D3.1 Key concept identification and clustering of similar content

Kalina Bontcheva, Danica Damljanovic, Niraj Aswani, Milan Agatonovic, James Sun (University of Sheffield)
Florence Amardeilh (Mondeca)

Abstract.
EU-IST Strategic Targeted Research Project (STREP) IST-2004-026460 TAO
Deliverable D3.1 (WP3)

This deliverable is concerned with developing algorithms and tools for semantic annotation of legacy software artefacts, with respect to a given domain ontology. In the case of non-textual content, e.g., screen shots and design diagrams we have applied OCR software, prior to Information Extraction. The results have been made available as a web service, which is in the process of being refined and integrated within the TAO Suite.

Keyword list: semantic annotation, concept identification, co-reference

Document Id.	TAO/2007/D3.1/v1.0
Project	TAO IST-2004-026460
Date	October 15, 2007
Distribution	Public
Reviewed By:	Farid Cerbah (Dassault Aviation)
Web links:	http://www.tao-project.eu

TAO Consortium

This document is part of a research project partially funded by the IST Programme of the Commission of the European Communities as project number IST-2004-026460.

University of Sheffield
Department of Computer Science
Regent Court, 211 Portobello St.
Sheffield S1 4DP
UK
Tel: +44 114 222 1891, Fax: +44 114 222 1810
Contact person: Kalina Bontcheva
E-mail: K.Bontcheva@dcs.shef.ac.uk

University of Southampton
Southampton SO17 1BJ
UK
Tel: +44 23 8059 8343, Fax: +44 23 8059 2865
Contact person: Terry Payne
E-mail: trp@ecs.soton.ac.uk

Atos Origin Sociedad Anonima Espanola
Dept Research and Innovation
Atos Origin Spain, C/Albarracin, 25, 28037 Madrid
Spain
Tel: +34 91 214 8835, Fax: +34 91 754 3252
Contact person: Jaime García Sáez
E-mail: jaime.2.garcia@atosorigin.com

Jozef Stefan Institute
Jamova 39
1000 Ljubljana
Slovenia
Tel: +386 1 4773 778, Fax: +386 1 4251 038
Contact person: Marko Grobelnik
E-mail: marko.grobelnik@ijs.si

Mondeca
3, cit Nollez
75018 Paris
France
Tel: +33 (0) 1 44 92 35 03, Fax: +33 (0) 1 44 92 02 59
Contact person: Jean Delahousse
E-mail: jean.delahousse@mondeca.com

Sirma Group Corp., Ontotext Lab
Office Express IT Centre, 5th Floor
135 Tsarigradsko Shose
Sofia 1784
Bulgaria
Tel: +359 2 9768, Fax: +359 2 9768 311
Contact person: Atanas Kiryakov
E-mail: naso@sirma.bg

Dassault Aviation SA
DGT/DPR
78, quai Marcel Dassault
92552 Saint-Cloud
Cedex 300
France
Tel: +33 1 47 11 53 00, Fax: +33 1 47 11 53 65
Contact person: Farid Cerbah
E-mail: Farid.Cerbah@dassault-aviation.com

Executive Summary

Content augmentation is a specific metadata generation task aiming to enable new information access methods. It enriches the text with semantic information, linked to a given ontology, thus enabling semantic-based search over the annotated content.

In the case of legacy software applications, important parts are the software code and documentation. While there has been a significant body of research on semantic annotation of textual content (in the context of knowledge management applications), only limited attention has been paid to processing legacy software artefacts, and in general, to the problem of semantic-based software engineering. This is one of the key areas addressed here.

This deliverable begins by providing an overview of content augmentation and breaks down the process into a number of tasks. The interactions with the ontology and the knowledge store is also defined here.

Next we investigate some general text analysis problems posed by software artefacts, namely tokenisation and sentence boundary detection. Implementational details of the source code tokeniser and JavaDoc sentence splitter are presented.

This is followed by an in-depth presentation of the key concept identification tools and the way they use the ontology as a dynamic source of lexical information.

The problem of information consolidation is compared against anaphora resolution and, then, we introduce our ontology-based consolidation method. An important distinguishing aspect of our work is that we do not perform ontology population directly, but instead, produce candidates for new instances in the ontology.

First experiments with content augmentation of non-textual software artefacts are also presented. We have evaluated some OCR tools on their ability to process software screen shots. The results of the TAO content augmentation tools are also presented and future improvements are discussed. We have also started collaborations with several speech recognition research groups, in order to experiment with applying ASR tools to tutorial movies. However, the state-of-the-art in this area is not as mature, as we had hoped for.

At the end, we draw some conclusions and plans for future work.

Contents

Chapter 1

Introduction

Until recently content augmentation with semantic information was perceived as a primarily manual task. However, the sheer volume of existing content and the symbiotic relationship between knowledge and documents has resulted in research on automatic tools based on Human Language Technology, and, more specifically, Information Extraction.

Information Extraction (IE) takes content (text, video, sound) as input and produces structured data as output. This data may be used directly for display to users, or may be stored in a semantic repository to power semantic-based search and browse and other intelligent access to content. IE is being applied in the context of Semantic Web and knowledge management to perform semantic annotation. Semantic annotation is a content augmentation process that links parts of text (e.g. a phrase) with classes and instances in an ontology, i.e. it assigns semantic metadata to content. Such semantically enriched text enables innovative methods of access and use, e.g. concept-based indexing and search, ontology-based categorisation, smooth traversal between content and knowledge.

Earlier work on semantic annotation focused primarily on textual content, e.g., S-Cream [HSC02], KIM [KPO$^+$04], perceptron-based IE [LBC07]. However, legacy content tends to be heterogeneous, including text, images, video, and structured data. In the context of the TAO project we consider the software-related documentation of the legacy applications, which contain text, images (screen shots), diagrams, videos (e.g. training materials). While there have been attempts to apply semantic annotation tools to multimedia data (e.g. news videos [DTCP05]), the approaches tend to be domain and application-specific and thus need to be developed further prior to being applied to software artefacts, such as screen shots, training videos, and software specifications.

Over the course of this and the following TAO deliverables from WP3, we will address the following challenges:

1. Given a domain ontology, develop algorithms for identification of key concepts mentioned in the software-related legacy content. For video, audio, and images third-party ASR and OCR tools will be applied prior to carrying out the content

augmentation task.

2. Clustering similar content, based on the identified key concepts, i.e., disambiguate and consolidate all mentions of concepts, instances, or properties (referred to as information consolidation phase).

3. Augmentation of the semantic annotations on the multimedia content by using those detected in textual sources, i.e. cross-media content augmentation.

4. Quantitative evaluation (using standard IE evaluation metrics) to compare the performance of semantic annotation on each content type in isolation and using cross-media augmentation.

5. Development of a user-friendly interface for semantic-based search of the augmented content and, if needed, for error correction. An existing semantic annotation and search tool for textual content will be extended with multimedia capabilities.

The first two challenges are adressed in this deliverable. The development of the cross-media content augmentation and end-user tools will be a two-stage process, where first versions are available at M24, then they are evaluated by M30, and improved versions are delivered at M36.

1.1 Relevance to Project Objectives

This deliverable contributes directly to TAO's second research objective, which is to develop tools for semantic augmentation and search of legacy content. In particular, here we have concentrated on addressing the first two of the five challenges mentioned above, and we also partially address the third one.

In the case of software applications, an important part of the legacy system is the software code and documentation. While there has been a significant body of research on semantic annotation of textual content (in the context of knowledge management applications), only limited attention has been paid to processing legacy software artefacts, and in general, to the problem of semantic-based software engineering. This is one of the key areas addressed in TAO, alongside the semantic web services dimension.

1.2 Relation to Other Workpackages

The research goals of WP3 are as follows:

- Develop semi-automatic techniques for semantic augmentation of legacy software content

- Deploy these as a web service for automatic content annotation
- Develop/integrate post-editing Web GUI for human correction of the automatic results
- Develop user tools for semantic-based search and browse of the augmented content

WP3 is dependent on the outcomes of WP2 (ontology learning), which *learns* the domain ontology from a sub-set of the legacy content (code, comments). In contrast, WP3 *uses* the ontology to semantically annotate all legacy content plus any new content. WP3 also has to deal with the dynamic document base, i.e., new documents come in all the time which need to be annotated semantically, e.g., from discussion forums, emails, etc.

The heterogeneous knowledge store (WP4) is used to access the ontology and also to store content augmentation results, via ontology population and metadata storage.

One of the outcomes of this deliverable is a web service for content augmentation, which is in the process of further refinement and integration within the TAO Suite (WP5). Its usage in various scenarios will be covered by the methodology (WP1).

The two case studies will use the results of this deliverable on their legacy content and provide feedback for further development. In addition, they may carry out some case study-specific customisations, if required.

1.3 Deliverable Outline

This deliverable is structured as follows.

Chapter 2 provides an overview of content augmentation and breaks down the process into a number of tasks. The interactions with the ontology and the knowledge stores is also defined here.

Chapter 3 investigates the general text analysis problems posed by software artefacts, namely tokenisation and sentence boundary detection. Implementational details of the source code tokeniser and JavaDoc sentence splitter are presented.

Next, Chapter 4 provides an in-depth presentation of the key concept identification tools and the way they use the ontology as a dynamic source of lexical information.

The problem of information consolidation is discussed in Chapter 5, where we define the task with respect to anaphora resolution problems and introduce our ontology-based consolidation method. An important distinguishing aspect of our work is that we do not perform ontology population directly, but instead, produce candidates for new instances in the ontology.

First experiments with content augmentation of non-textual software artefacts are presented in Chapter 6, where we evaluate some OCR tools on their ability to process software screen shots. The results of the content augmentation tools from the previous chapters are also presented and potential future improvements discussed. We have also started a collaboration with several speech recognition research groups, in order to experiment with applying ASR tools to tutorial movies. However, the state-of-the-art in this area is not as mature, as we hoped for.

At the end, we draw some conclusions and directions for future work.

Chapter 2

Content Augmentation Framework

Content augmentation is a specific metadata generation task aiming to enable new information access methods. It enriches the text with semantic information, linked to a given ontology, thus enabling semantic-based search over the annotated content.

The first task of content augmentation (often referred to as semantic annotation) could be seen as an advanced combination of a basic press-clipping exercise, a typical information extraction task, and automatic hyper-linking to an ontology. The resulting annotations represent basically a method for document enrichment and presentation, the results of which can be further used to enable semantic-based access methods.

The second task is concerned with storage and retrieval of the semantically augmented content. It can be considered as a modification of the classical IR task - documents are indexed and retrieved on the basis of relevance to semantic annotations instead of words.

In this deliverable we will focus only on aspects of the first task, i.e., on semantic annotation tools for automatic content augmentation of legacy software content. To complement the automatic approach, the forthcoming deliverable D3.2 will focus on tools for post-editing and manual correction of augmented content, including merging information from different content.

The second, semantic indexing and search task will be addressed in another forthcoming deliverable (D3.4), where a set of user tools will be developed to enable user-friendly semantic search and browse of the augmented content. The tools will show the ontology and the user will be able to construct queries in an intuitive manner (e.g., drag and drop). These tools will be the front-end to the heterogeneous knowledge stores, where the semantically augmented content and semantics will be stored.

2.1 Overview

Automatic semantic annotation tools are typically composed of the following main components (see Figure 2.1: the Information Extraction Module, the Information Consolidation Module and the Information Export Module. The first two modules are discussed in detail in the following sections. The Information Export Module, is responsible for exporting the semantic annotation metadata in the format required by the chosen semantic indexing and search solution, which in our case will be the heterogeneous knowledge store from WP4. On the other hand, by introducing this module, our approach will retain the flexibility to also export its results in other repositories such as ITM or Annotea. As part of the TAO Suite integration effort we are in the process of defining the data format of the export module, so augmented content can be stored and accessed easily via the HKS web service.

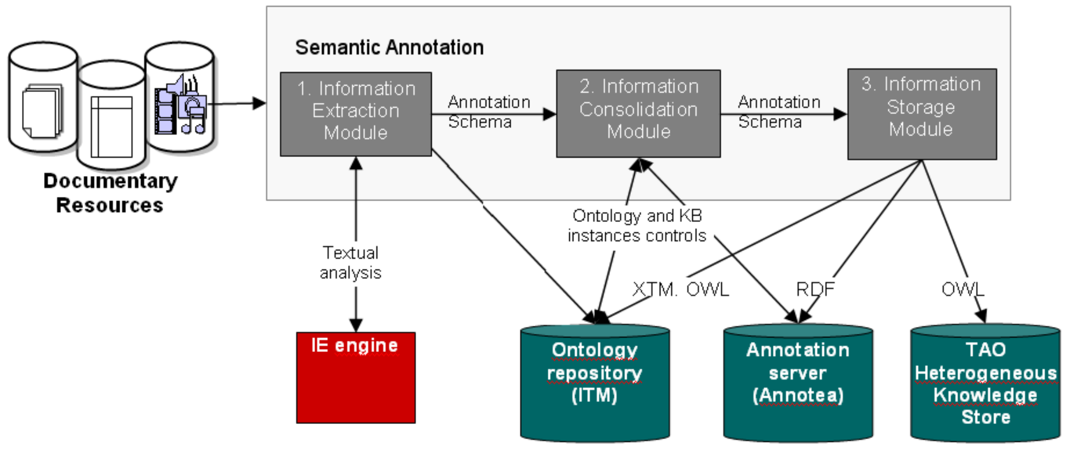

Semantic Annotation
Documentary Resources
1. Information Extraction Module
Annotation Schema
2. Information Consolidation Module
Annotation Schema
3. Information Storage Module
Ontology and KB instances controls
Textual analysis
XTM. OWL
RDF
OWL
IE engine
Ontology repository (ITM)
Annotation server (Annotea)
TAO Heterogeneous Knowledge Store

Figure 2.1: Architecture of a typical semantic annotation framework

The semantic annotation tools should take into account the following requirements:

- **Mapping the structure of the ontology and the structure of the linguistic extractions**, modelled in separate ways. Annotating a document and/or populating an ontology must not impose new constraints on the way the terminological and ontological resources are modelled nor on the format produced by the IE tools.
- **Completeness**. The approach must be able to map all information given by the IE tools.
- **Standardisation**. The approach must not be dependent on the IE tool used and it must produce Semantic Web compliant formats such as RDF and OWL.

- **Consistency**. The instances created in the knowledge base and the semantic annotations produced must be consistent with the ontology model.
- **Capacity to evolve**. The approach must be able to take into account evolutions of both the ontological resources and the IE tools.

2.2 The Information Extraction Module

The first step of the annotation workflow consists in extracting from a document all relevant information relating to the concerned domain. The Information Extraction Module connects to the chosen IE engine that analyzes the document according to its lexicons and its set of extractions patterns. It locates the information to be extracted in the document and tags it in order to generate a **conceptual tree** as the output. This term ”conceptual tree” describes the results of the IE engines, being the ones produced by our IE components for example, although they do not truly correspond to a ”tree of concepts” (in the ontological sense).

Consequently, one needs to map the semantic tags from the conceptual tree resulting of the linguistic analysis with the concepts, attributes and relations modelled in the domain ontology. Not only is it necessary to correctly interpret the semantics provided by the conceptual trees but also to take into account the gap that may exist between the two modes of knowledge representation.

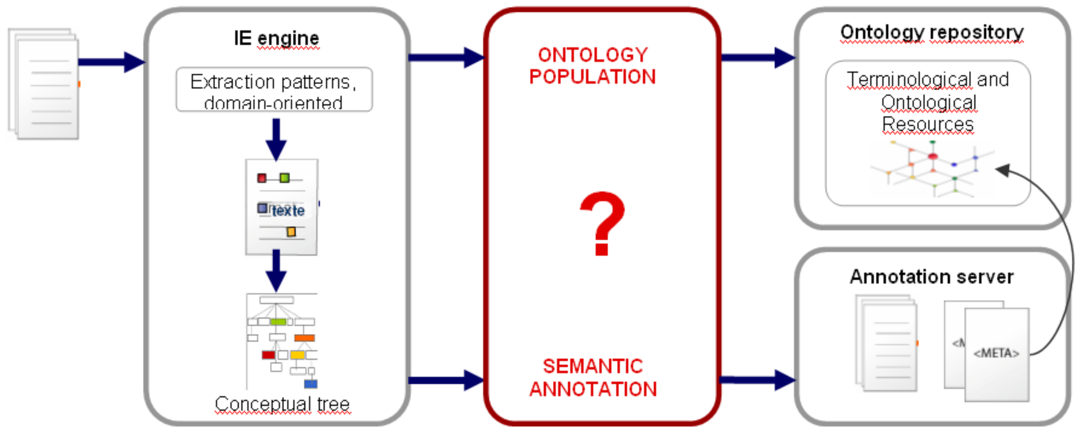


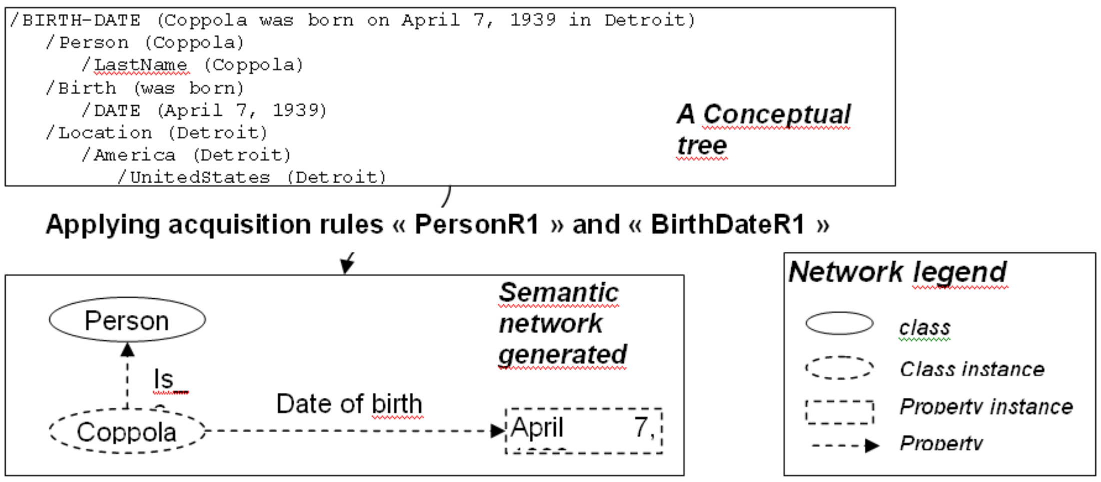
Figure 2.2: Bridging the gap from IE to Semantic Representation

In the existing solutions, annotation tools are closely linked and dependent on the mapping carried out between the two modes of knowledge representation. As an example, OntoMat, in its S-CREAM version [HSC02], recognizes that its integration with the Amilcare IE tool is made of ”ad hoc” and specific mapping rules. This mapping could not be used to integrate another IE tool. However, we want to emphasize the fact that a semantic annotation and/or ontology population tool should be able to easily plug in a new IE engine according to the target application's needs.

Decoupling the IE components within the semantic annotation process allows us to provide more flexibility and modularity for the target applications. But to do so, we need to find a generic solution to fill the existing gap as presented in Figure 2.2. It is thus necessary to design a gateway between these two representations.

One solution is to use declarative rules, called Knowledge Acquisition Rules (KAR) by [Ama06]. These rules map one or more semantic tags of a conceptual tree with an element (concept, attribute or relation) of the domain ontology. Concretely a rule identifies the semantic tag which will trigger the annotation or population process. It is also able to take into account the context of the semantic tag in the conceptual tree in order to solve a certain number of ambiguities.

Since a conceptual tree can be represented as an XML document, the Information Extraction Module makes use of the XML family of languages to compile and execute the KARs. As an example, if the linguistic analysis of the sentence ”Coppola was born on April 7, 1939 in Detroit” produces the conceptual tree located at the top of Figure 2.3, then the application of the created KARs defined for this application will create the semantic network located at the bottom of Figure 2.3. This semantic network associates the attribute ”Date of birth” having the value ”April 7, 1939” with the instance ”Coppola” of class ”Personality”.



```
/BIRTH-DATE (Coppola was born on April 7, 1939 in Detroit)
   /Person (Coppola)
      /LastName (Coppola)
   /Birth (was born)
      /DATE (April 7, 1939)
   /Location (Detroit)
      /America (Detroit)
         /UnitedStates (Detroit)
```

Figure 2.3: Applying Knowledge Acquisition Rules on a conceptual tree to produce the associated semantic network

From a methodological perspective, the Knowledge Acquisition Rules constitute the gateway foundations between the linguistic results and the semantic knowledge representation. From a software solution perspective, they are the essential ingredient to enable correct operation of the ontology population and semantic annotation processes.

Another solution to the gap problem is to make parts of the IE process ontology-based, so they take the domain ontology as an input and are thus capable of producing semantic annotations referring to the appropriate domain concepts from the ontology. This is what we refer to as *Ontology-Based Information Extraction* (OBIE) and it is the approach that we have experimented with in this deliverable.

OBIE approaches have similar methodologies to those used for traditional IE systems, but using an ontology rather than a flat gazetteer. For rule-based systems, this is relatively straightforward. For learning-based systems, however, this is more problematic because training data is required, but collecting such data has proved to be a large bottleneck. Unlike traditional IE systems for which training data exists in domains like news texts in plentiful form, thanks to efforts from MUC, ACE and other collaborative and/or competitive programs, there is a dearth of material currently available for training OBIE modules, particularly in specialised domains like ours. Consequently, if a learning approach is to be used, new training data needs to be created manually or semi-automatically, which is a time-consuming task.

2.3 The Information Consolidation Module

The second phase is what is often referred to as *Information Consolidation*[1].

As indicated in [AKM+03], rare are the tools for ontology population or semantic annotation which describe, or even mention, the consolidation phase in their workflows. However, this phase is extremely important to maintain the integrity and the quality of the application results. In fact, most of them rely only on manual validation to check the generated annotations or instances. Some annotation tools, like OntoMat [HSC02] or SMORE [KPS+05], have an ontology editor which allows the end-users to control the domain and range constraints on the created annotations. From the ontology population point of view, only one project, ArtEquAkt [AKM+03], was concerned by the consolidation phase and clearly specified it.

In this project, Alani et al. define four problems related to the integration of new instances in a knowledge base through ontology population: duplicated information, geographical consolidation, temporal consolidation and inconsistent information. Only some of these problems arise in the context of TAO, as, for example, software artefacts do not tend to have geographical information:

[1]In the original TAO Description of Work, this task is referred to as clustering of similar content, but due to the wide usage of this term meaning document clustering, we have decided to use information consolidation in order to help the reader in distinguishing between the two tasks.

- **Duplicated Information**: merging the instances with the same label, merging the instances if possessing a common set of attributes, merging the attributes when identical (name, value);
- **Geographical Consolidation**: using relations of synonymy and specialization in a geographical thesaurus, such as the Thesaurus of Geographic Names (TGN), clarifying location names using contextual analysis in the document content or associated semantic network;
- **Temporal Consolidation**: reasoning on the dates to identify them in a precise manner, clarifying the dates using contextual analysis in the document content or associated semantic network;
- **Inconsistent Information**: using frequency of extraction as a proof for precision.

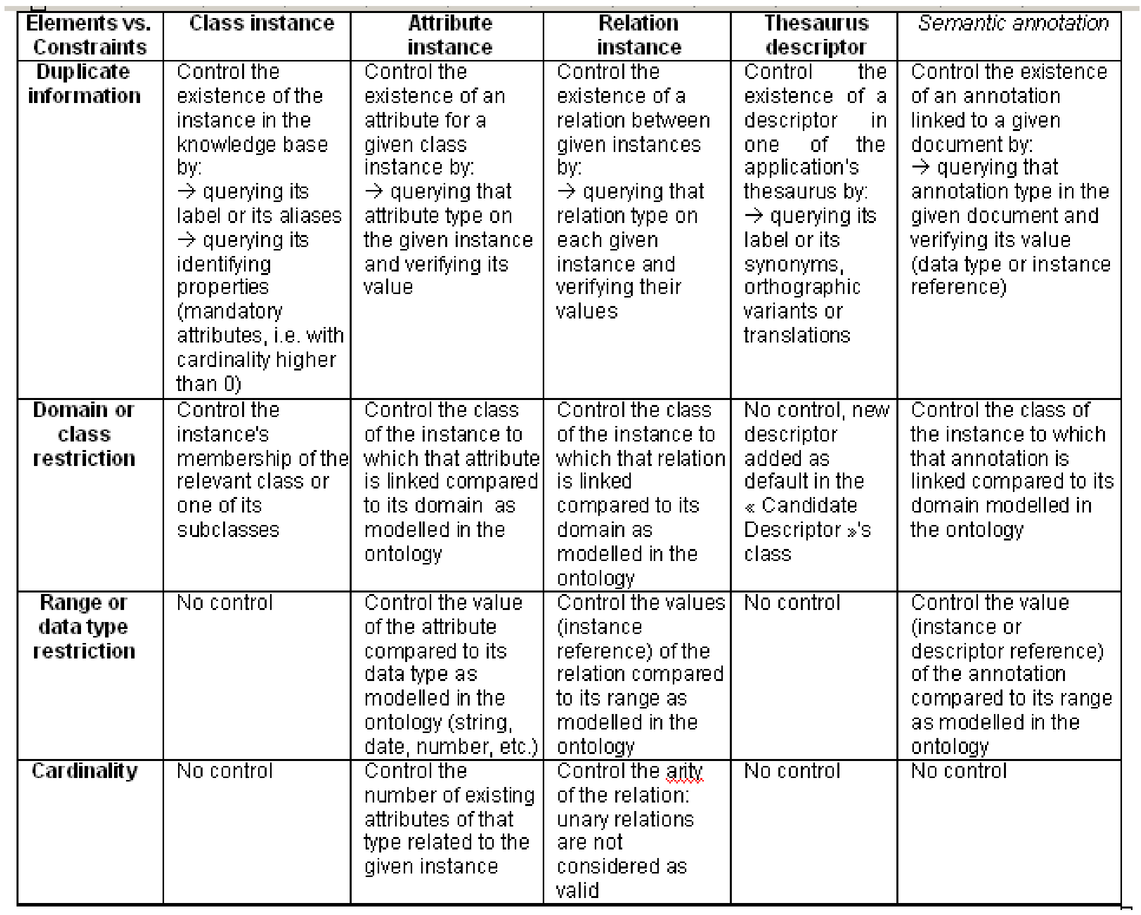
Their approach to solving these problems consists of instantiating the knowledge base with the information extracted from the documents and then applying a consolidation algorithm based on a set of heuristics and methods of terminological expansion. This algorithm uses WordNet in order to automate the process performed on the instances of the knowledge base.

In our view, in order to preserve the integrity of the knowledge base, this consolidation phase must be carried out before the creation of the instances in the knowledge base. As we said, the semantic network and the semantic annotations resulting from the linguistic analysis need to be analysed in depth to remove any ambiguity, any inconsistency or any conflict with already existing information. Thus, only new and consistent information is created, thus preserving the integrity of the referential and improving the quality of the augmented content.

We studied the various possible cases of instances and annotation creation. We deduced two axes of consolidation:

- the first axis defines the ontological element concerned, i.e. an instance of a class, of an attribute, of a relation, a thesaurus descriptor or a semantic annotation;
- the second axis defines the constraints to be checked, i.e. non redundancy, the domain and range restrictions and the element's cardinality.

The second axis must be adapted according to the ontological element consolidated. Indeed for an instance of class as for a thesaurus descriptor, it is not necessary to control the domain and range restrictions. But, rather the domain restriction on a class instance could be regarded as the correct class attribution to that instance in the ontology. The same way, the range restriction for an attribute can be understood as checking the data type awaited by the knowledge base: is it a character string, a numeric, an address URL or a date? According to these axes, we define all the recommended operations of consolidation, cf. Table 2.4.

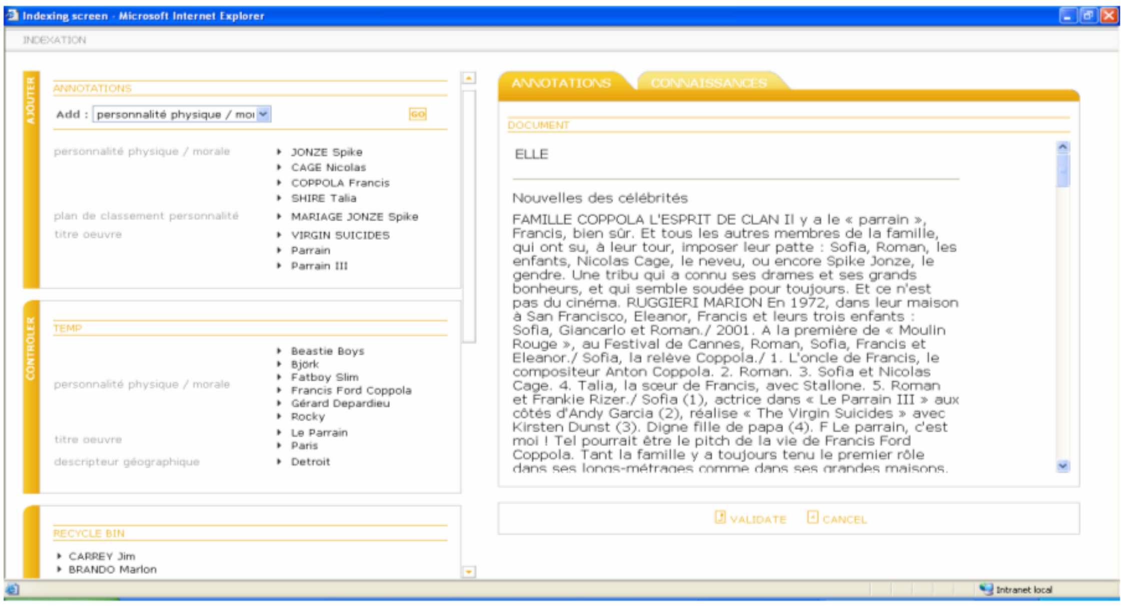


Elements vs. Constraints	Class instance	Attribute instance	Relation instance	Thesaurus descriptor	*Semantic annotation*
Duplicate information	Control the existence of the instance in the knowledge base by: → querying its label or its aliases → querying its identifying properties (mandatory attributes, i.e. with cardinality higher than 0)	Control the existence of an attribute for a given class instance by: → querying that attribute type on the given instance and verifying its value	Control the existence of a relation between given instances by: → querying that relation type on each given instance and verifying their values	Control the existence of a descriptor in one of the application's thesaurus by: → querying its label or its synonyms, orthographic variants or translations	Control the existence of an annotation linked to a given document by: → querying that annotation type in the given document and verifying its value (data type or instance reference)
Domain or class restriction	Control the instance's membership of the relevant class or one of its subclasses	Control the class of the instance to which that attribute is linked compared to its domain as modelled in the ontology	Control the class of the instance to which that relation is linked compared to its domain as modelled in the ontology	No control, new descriptor added as default in the « Candidate Descriptor »'s class	Control the class of the instance to which that annotation is linked compared to its domain modelled in the ontology
Range or data type restriction	No control	Control the value of the attribute compared to its data type as modelled in the ontology (string, date, number, etc.)	Control the values (instance reference) of the relation compared to its range as modelled in the ontology	No control	Control the value (instance or descriptor reference) of the annotation compared to its range as modelled in the ontology
Cardinality	No control	Control the number of existing attributes of that type related to the given instance	Control the arity of the relation: unary relations are not considered as valid	No control	No control

Figure 2.4: Operations of consolidation performed accordingly to the two axes defined

If the instances, the descriptors or the annotations are rejected by the consolidation phase, they can either be rejected through deletion or be saved in a ”buffer” in order to be subsequently proposed to the end-user for correction and validation. We consider that the most flexible approach is to regard any knowledge as exploitable even if it requires human intervention. Nevertheless, the knowledge that is not conforming to the ontology model should not make the knowledge base inconsistent. This is why it needs either to be deleted, or ideally, kept separate from the valid instances and annotations.

In the case of a semi-automated usage of the semantic annotation tools, the end-user has to validate the results generated by the automatic process in order to verify its performance and quality. A single user interface, such as that of ITM (see Figure 2.5), enables the validation of both the semantic annotations and created instances simultaneously. The user can edit, modify, add new ones or remove wrong ones. Each of these actions is constrained by the ontology model so that the user cannot add inconsistencies to the knowledge base or to the semantic annotations. The annotations and instances that were rejected by the automated consolidation process are also presented to the user. They can easily accept new information considered as relevant that the consolidation process did not succeed in solving automatically. This information can also be merged with existing instances or annotations.



ANNOTATIONS

Add : personnalité physique / moı

personnalité physique / morale	JONZE Spike CAGE Nicolas COPPOLA Francis SHIRE Talia
plan de classement personnalité	MARIAGE JONZE Spike
titre oeuvre	VIRGIN SUICIDES Parrain Parrain III

TEMP

personnalité physique / morale	Beastie Boys Björk Fatboy Slim Francis Ford Coppola Gérard Depardieu Rocky
titre oeuvre	Le Parrain Paris
descripteur géographique	Detroit

RECYCLE BIN

- CARREY Jim
- BRANDO Marlon

DOCUMENT

ELLE

Nouvelles des célébrités

FAMILLE COPPOLA L'ESPRIT DE CLAN Il y a le « parrain », Francis, bien sûr. Et tous les autres membres de la famille, qui ont su, à leur tour, imposer leur patte : Sofia, Roman, les enfants, Nicolas Cage, le neveu, ou encore Spike Jonze, le gendre. Une tribu qui a connu ses drames et ses grands bonheurs, et qui semble soudée pour toujours. Et ce n'est pas du cinéma. RUGGIERI MARION En 1972, dans leur maison à San Francisco, Eleanor, Francis et leurs trois enfants : Sofia, Giancarlo et Roman./ 2001. A la première de « Moulin Rouge », au Festival de Cannes, Roman, Sofia, Francis et Eleanor./ Sofia, la relève Coppola./ 1. L'oncle de Francis, le compositeur Anton Coppola. 2. Roman. 3. Sofia et Nicolas Cage. 4. Talia, la sœur de Francis, avec Stallone. 5. Roman et Frankie Rizer./ Sofia (1), actrice dans « Le Parrain III » aux côtés d'Andy Garcia (2), réalise « The Virgin Suicides » avec Kirsten Dunst (3). Digne fille de papa (4). F Le parrain, c'est moi ! Tel pourrait être le pitch de la vie de Francis Ford Coppola. Tant la famille y a toujours tenu le premier rôle dans ses longs-métrages comme dans ses grandes maisons.

Figure 2.5: The “Annotations” tab in ITM’s Validation user interface

To sum up, the consolidation phase consists of:

- controlling the instances and semantic annotations according to the ontology model (domain and range restrictions, cardinalities), to the knowledge base, and to controlled vocabularies such as a thesaurus or reference tables;
- providing a user interface for validating the results obtained automatically.

In chapter 5 of this deliverable we will focus on the first task, i.e., algorithms for information consolidation, whereas the user validation interface will be developed as part of the forthcoming D3.2.

2.4 Accessing and Modifying Ontologies for Content Augmentation

As shown in Figure 2.1, content augmentation modules need to access knowledge from the ontology in order to be able to use it as a knowledge source during all semantic annotation phases: information extraction, consolidation, and result storage. In addition, in the last phase, it is often necessary to carry out ontology population by storing results into the ontology, which in our context is stored and managed in OWLIM. Consequently, we created an ontology web service which provides access to ontologies in OWLIM, with fine-grained methods, such as obtaining the sub-classes of a given class, the properties of a given instance, etc.

In addition, many typical content augmentation scenarios, including the TAO aviation case study (see TAO D7.1), have requirements that the user is able to access visually the content of the ontology, add new instances and properties, and even classes, all of this as part of the semantic annotation process. In other words, what is required is a seamless switch between documents and ontology editing. In order to support this requirement, we developed also a simple ontology browsing and editing component, which we plan to integrate in the user validation interface in D3.2 and the TAO Suite.

Further details are provided in Appendix A.

Chapter 3

Text Processing of Software Artefacts

Software artefacts present a challenge for general purpose language processing tools, such as tokenisers and sentence splitters, because they are semi-structured and contain variable names, which internally consist of one or more words (e.g., getDocumentName). Consequently, in order to enable appropriate processing of such texts, one needs to customise such generic tools accordingly.

In particular, this chapter discusses how the generic, open-source ANNIE English Tokeniser and Sentence Splitter were customised for analysing Java source code and JavaDoc files[1].

3.1 Tokenisation of source code/JavaDoc

The tokenisation is a pre-processing step of content augmentation and it splits the text into very simple tokens such as numbers, punctuation and words of different types. For example, tokenisers distinguish between words in uppercase and lowercase, and between certain types of punctuation. Typically, tokenisation takes place by using spaces and punctuation marks as token delimiters.

However, as already mentioned above, each programming language and software project tends to have naming conventions and these need to be considered during tokenisation, in order to enable searching within method and variable names.

Consequently, we had to modify a generic English tokeniser (the ANNIE English Tokeniser) so that it *separates variable and method names* into their constituent words, i.e., `getDocumentName` should be separated into *get*, *Document*, and *Name* tokens, prior to being submitted as input to the subsequent content augmentation algorithms.

The generic tokeniser uses a set of rules, where a rule has a left hand side (LHS) and

[1]JavaDoc are documentation files created automatically from Java source code and the comments inside it. For an example, see http://www.gate.ac.uk/releases/gate-4.0-build2752-ALL/doc/javadoc/index.html

a right hand side (RHS). The LHS is a regular expression which has to be matched on the input text and contains Unicode character classes. The RHS describes the tokens to be created. The LHS is separated from the RHS by '>'. The traditional Klene operators can be used on the LHS: —, *, ?, +.

The RHS uses ';' as a separator, and has the following format:

```
{LHS} > {Annotation type};{attribute1}={value1};...;{attribute
n}={value n}
```

The following tokeniser rule is for a word beginning with a single capital letter:

```
"UPPERCASE_LETTER" "LOWERCASE_LETTER"* >
  Token;orth=upperInitial;kind=word;
```

It states that the character sequence must begin with an uppercase letter, followed by zero or more lowercase letters. This sequence will then be annotated as type "Token". The attribute "orth" (orthography) has the value "upperInitial"; the attribute "kind" has the value "word".

In the generic tokeniser, a word is defined as any set of contiguous upper or lowercase letters. A word token is given the attribute "orth", for which four values are possible:

- upperInitial - initial letter is uppercase, rest are lowercase
- allCaps - all uppercase letters
- lowerCase - all lowercase letters
- mixedCaps - any mixture of upper and lowercase letters not included in the above categories

Consequently, when variable and method names are tokenised in a generic fashion, they are marked as word tokens with orthography mixedCaps.

In order to address this problem, we added a post-processing step to the generic English tokeniser, which iterates through all mixedCaps tokens, splits them as necessary, deletes the original mixedCap token, and adds tokens for each of the sub-parts. For instance, getDocumentName is split into the three respective tokens: get, Document, and Name.

The token splitting is generally done when the case of the letters changes, e.g., from lowercase to uppercase, or when a dash or underscore is encountered (e.g. get-document-name). The only exception is when we have a sequence of uppercase letters (e.g., ANNIETokeniser) and in that case, tokenisation leaves the last uppercase letter for the next token, i.e., ANNIE and Tokeniser.

3.2 Sentence segmentation of JavaDoc

Another required task is segmenting software artefacts into sentences, so it is possible during semantic search to present only the relevant snippet of information, rather than the entire document (although the user would also be able to browse the entire document, if interested).

The generic ANNIE sentence splitter is a cascade of finite-state transducers which segments text into sentences. It uses a gazetteer list of abbreviations to help distinguish sentence-marking full stops from other kinds. However, it suffers from the problem that it has been designed to deal only with regular, well-formatted text.

We have found in TAO that it works well for discursive software artefacts, such as user and programmer guides. Java documentation, however, is generated automatically from comments in the source code and comes in HTML format. The problem is that programmers writing comments do not always provide punctuation marks, which means that the generic sentence splitter would tend to lump together entries about several methods into one sentence.

For instance, the text:

```
AnnotationSet get ( String type, FeatureMap constraints, Long offset)
Select annotations by type, features and offset

AnnotationSet get ( Long offset)
Select annotations by offset. This returns the set of annotations
whose start node is the least such that it is greater than or equal
to offset. If a positional index doesn't exist it is created.
```

would be segmented wrongly as three sentences, the first one covering two methods and some of the comments:

```
<Sentence>
AnnotationSet get ( String type, FeatureMap constraints, Long offset)
Select annotations by type, features and offset

AnnotationSet get ( Long offset) Select annotations by offset.
</Sentence>
<Sentence>
This returns the set of annotations whose start node is the least such
that it is greater than or equal to offset.
</Sentence>
<Sentence>
If a positional index doesn't exist it is created.
</Sentence>
```

Consequently, we extended the generic sentence splitter with new grammars that take into account the HTML formatting tags and break sentences not only at full stops, but also on table cell boundaries, headers, titles, definition terms and descriptions, list items, etc.

This was achieved first by creating a new grammar that takes as input the HTML markup of the JavaDoc and produces candidate sentence split annotations, which are indicators of a potential sentence boundary. These splits are then combined with those created on the basis of punctuation and abbreviations by the default English splitter. Finally, Sentence annotations are created based on the final set of sentence splits.

For our example, the result is now as required:

```
<Sentence>
AnnotationSet get ( String type, FeatureMap constraints, Long offset)
</Sentence>
<Sentence>
Select annotations by type, features and offset
</Sentence>
<Sentence>
AnnotationSet get ( Long offset)
</Sentence>
<Sentence>
Select annotations by offset.
</Sentence>
<Sentence>
This returns the set of annotations whose start node is the least
such that it is greater than or equal to offset.
</Sentence>
<Sentence>
If a positional index doesn't exist it is created.
</Sentence>
```

3.3 Discussion and Future Work

In this chapter we presented how we adapted two of the basic NLP processing tools – tokeniser and sentence splitter – to the specialised formatting and token conventions of software artefacts.

Part-Of-Speech (POS) tagging is another basic text analysis stage, which given a set of tokens, assigns their part of speech, e.g., verb in past tense, proper noun. POS taggers are typically trained on large, human-annotated corpora, and, in our experience so far, tend to be sufficiently accurate when tagging software artefacts. Consequently, for the time being, we have decided against re-training a generic tagger, specifically on software artefacts, as this would require a substantial manual annotation effort.

Another generic component which we reuse without modification is an English morphological analyser. It takes as input a tokenized document and iterates through each token and its part of speech tag, assigning its lemma and an affix.

We have now commenced formal performance evaluation experiments as part of the evaluation deliverable D3.3 in this workpackage. In particular, we will compare the performance levels of the generic English tokeniser, sentence splitter and POS tagger against those developed by us for software artefacts. Further improvements in these components, following the evaluation, will also be reported there.

Chapter 4

Key Concept Identification

Identifying key concepts from software-related legacy content can improve the process of information retrieval and search for particular documents. However, the identification is usually a time consuming process as it is mostly performed manually. Describing this content in a more structured way i.e. developing a domain specific ontology to describe this content is a step towards the possibility to identify key concepts automatically.

This chapter presents the Key Concept Identification Tool (KCIT) for automatic retrieval of key concepts from software-related legacy content w.r.t. a domain ontology. KCIT is combining the features of several generic language analysis components (e.g., sentence splitter, tokeniser, and GATE's Flexible Gazetteer) with some newly developed ones, the main one being the OntoRoot Gazetteer. The OntoRoot Gazetteer is using the features of the generic language analysers such as the gazetteer and morphological analyzer in order to achieve effectiveness and robustness when identifying the key concepts.

In the following sections we provide details of the implementation and give examples to illustrate how this tool works. Finally, we compare our work against other similar tools that exist up to date and propose ideas for our future work.

4.1 Key Concept Identification Tool

Semantic annotation is usually the first mandatory step when performing some more important tasks such as semantic indexing, searching, keyword extraction, ontology population and others. For cases when a domain ontology is already developed, identifying key concepts in legacy software documents would be possible by linking the appropriate parts of the documents to the particular ontology resources. Identified, semantically enriched content can further be used to enhance process of semantic indexing and search. However, the process of producing ontology-aware annotations automatically is not trivial, as the language used to describe concepts and relations in ontologies can differ from the language appearing in legacy software content. Additionally, the natural human language itself present in a software documentation is well known for ambiguity and complexity.

Many available tools for producing ontology-aware annotations exist nowadays. However, most of them use static lists for a gazetteer and match only exact text in documents from that in the list. Our approach differs in that of matching all morphological inflections of the relevant terms by using a morphological analyzer in the dynamic construction of the gazetteer lists from the ontologies. We developed the Key Concept Identification Tool (KCIT) to automatically retrieve key concepts from legacy documents by creating ontology-aware annotations over them. These annotations are created based on the assumption that a specific part of a document is referring to a particular resource residing inside the ontology if the lemmas[1] of the two match. A particular ontology resource is identified mostly by its URI, labels, or by a value of some set properties. Annotations contain link to the ontology resources they refer to so that they can be used for performing some other tasks later on.

The KCIT process can be broken down into several steps:

1. **Building a list of relevant terms**. For this step we developed a new component *the Ontology Resource Root Gazetteer (ORRG)*. Given an ontology, ORRG extracts and lemmatizes the lexicalisations of all ontological resources (classes, instances, and properties) and creates a gazetteer list.

2. **Annotating the legacy content**. The legacy content that is being processed is first lemmatized with a morphological analyser. It is then matched against the gazetteer list created in the previous step. For this purpose we are using a Flexible Gazetteer module that uses ORRG from the previous step coupled with some other language analysis components (the TAO tokeniser, TAO sentence splitter, a generic POS Tagger, a generic morphological analyser), which all together comprise the *the Ontology Resource Finder (ORF) Application.*

3. **Resolving conflicts**. This step includes solving ambiguity problems such as identifying the same part of content with concepts of different meaning.

Following sections describe each step in detail.

4.1.1 Building a Dynamic Gazetteer from the Ontology

To produce ontology-aware annotations i.e. annotations that link to the specific concepts/relations from the ontology, it is essential to pre-process the Ontology Resources (e.g., Classes, Instances, Properties) and extract their human-understandable lexicalisations. As *rdf:label* property is meant to have a human-understandable value [Cha01], it is

[1]Lemma is the canonical form of a lexeme. Lexeme refers to the set of all the forms that have the same meaning, and lemma refers to the particular form that is chosen by convention to represent the lexeme.The process of determining the lemma for a given word is called lemmatisation.

a good candidate for the gazetteer. Additionally, labels contain multilingual values, which means that the same tool can be used over the documents written in different languages - as long as that language is supported by the ontology.

However, the part of the Unique Resource Identifier (URI) itself is sometimes very descriptive, making it a good candidate for the gazetteer as well. This part is called *fragment identifier*[2].

As a precondition for extracting human-understandable content from the ontology we created a list of the following:

- names of all ontology resources i.e. fragment identifiers and
- values of all set properties for all ontology resources (e.g., values of labels, values of datatype properties, etc.)

Each item from this list is analysed separately by the Onto Root Application (ORA) on execution (see figure 4.1). The Onto Root Application first tokenises each linguistic term, then assigns part-of-speech and lemma information to each token.

As a result of that pre-processing, each token in the terms will have additional feature named 'root', which contains the lemma as created by the morphological analyser. It is this lemma or a set of lemmas which are then added to the dynamic gazetteer list, created from the ontology.

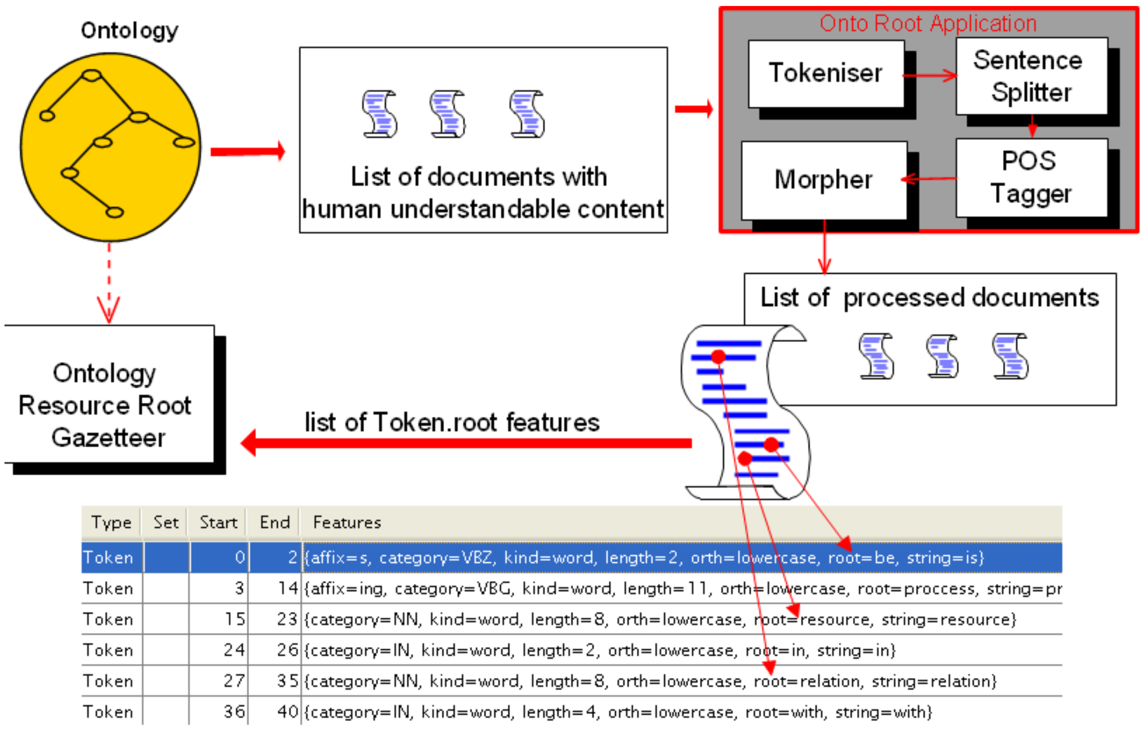
For instance, if there is a resource with a short name (i.e., fragment identifier) *ANNIEJapeTransducer*, with assigned property *rdf:label* with values *Jape Transducer* and *ANNIE Jape Transducer*, and with assigned property *rdf:comment* with value *A module for executing Jape grammars*, the created list before executing the OntoRoot gazetteer collection will contain following the strings:

- '*ANNIEJapeTransducer*',
- '*Jape Transducer*',
- '*ANNIE Jape Transducer*' and
- '*A module for executing Jape grammars*'.

Each of the items from the list is then analysed separately and the results would be:

- For '*ANNIEJapeTransducer*', '*Jape Transducer*', and '*ANNIE Jape Transducer*' the output will be the same as the input, as the lemmas are the same as the input tokens.

[2]An ontology resource is usually identified by URI concatenated with a set of characters starting with '#'. This set of characters is called *fragment identifier*. For example, if the URI of a class representing *GATE POS Tagger* is: 'http://gate.ac.uk/ns/gate-ontology#POSTagger', the fragment identifier will be 'POSTagger'.


Ontology
List of documents with human understandable content
Onto Root Application
Tokeniser
Sentence Splitter
Morpher
POS Tagger
List of processed documents
Ontology Resource Root Gazetteer
list of Token.root features

Type	Set	Start	End	Features
Token		0	2	{affix=s, category=VBZ, kind=word, length=2, orth=lowercase, root=be, string=is}
Token		3	14	{affix=ing, category=VBG, kind=word, length=11, orth=lowercase, root=proccess, string=pr
Token		15	23	{category=NN, kind=word, length=8, orth=lowercase, root=resource, string=resource}
Token		24	26	{category=IN, kind=word, length=2, orth=lowercase, root=in, string=in}
Token		27	35	{category=NN, kind=word, length=8, orth=lowercase, root=relation, string=relation}
Token		36	40	{category=IN, kind=word, length=4, orth=lowercase, root=with, string=with}

Figure 4.1: Building Ontology Resource Root Gazetteer from the Ontology

- For '*A module for executing Jape grammars*' the output will be the set of lemmas from the input resulting in '*A module for execute Jape grammar*'.

In this way, a dynamic gazetteer list is created directly from the ontology resources and is then used by the subsequent components to annotate mentions of classes, instances, and properties in the legacy content. It is essential that the gazetteer list is created on the fly, because it needs to be kept in sync with the ontology, as the latter changes over time.

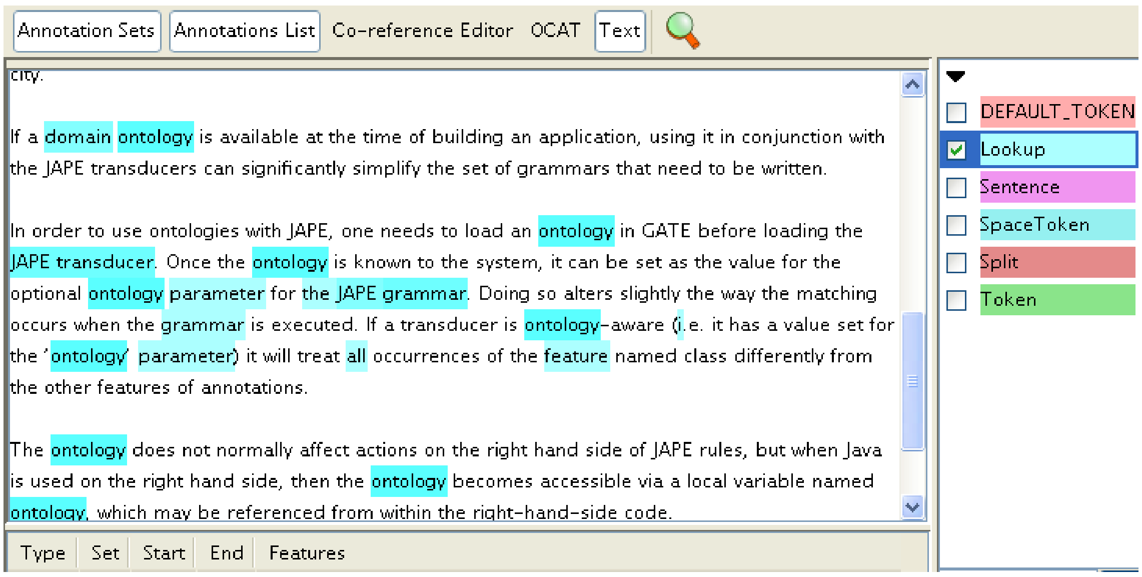
4.1.2 Annotating the Legacy Content

As we created the list of relevant terms explained in the previous section, it is feasible to perform a direct gazetteer lookup against this list. By default, a Gazetteer is a language processing component that matches a list of entries against the document content, if they appear in the text in the exact form as they are in the Gazetteer list. Due to morphological variations in English and many other languages, the default behavior is not always sufficient to provide the required flexibility and match all morphological inflections of the relevant terms.

To enable considering lemmas when annotating documents against the gazetteer of ontology terms, we use a Flexible Gazetteer. The most important difference between a default Gazetteer and a flexible one is that the latter matches against document annotations, not against the document content itself. In effect, the Flexible Gazetteer performs lookup based on the values of a given feature of an arbitrary annotation type, by using an externally provided gazetteer [CMB+05].

In KCIT we use the ORRG gazetteer (created in the previous step) as the external gazetteer. The output of the morphological analyzer creates features called 'root' and add them to the document tokens (which are annotations of type 'Token'). Consequently, we set the Flexible Gazetteer to use the values of the Token.root features during the annotation process.

To illustrate the advantage of using the Flexible Gazetteer over the default one, we run them against the same ontology (GATE domain ontology) and over the same document (GATE User manual). The results are shown on Figure 4.2 and Figure 4.3 respectively.



city.

If a domain ontology is available at the time of building an application, using it in conjunction with the JAPE transducers can significantly simplify the set of grammars that need to be written.

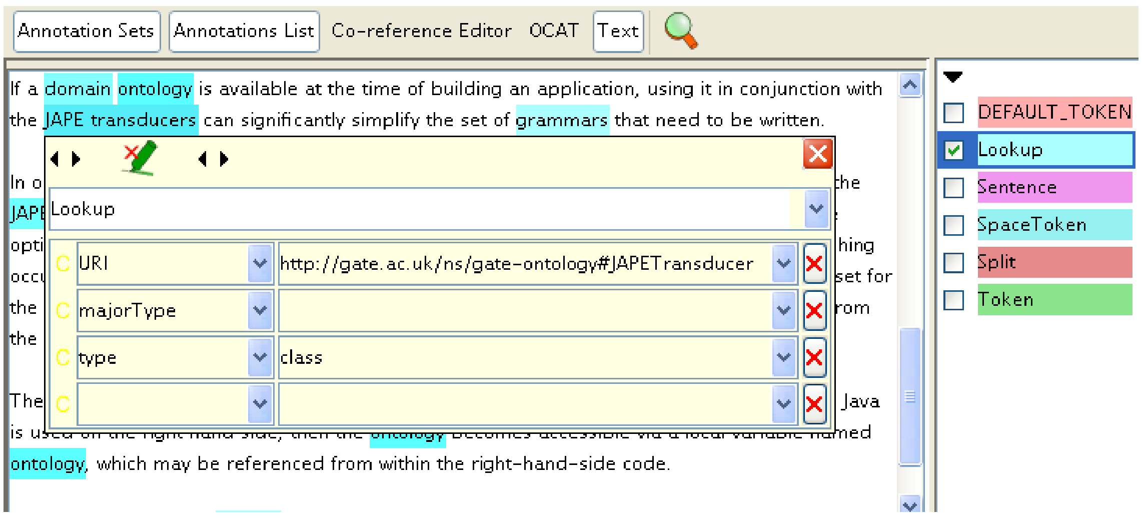
In order to use ontologies with JAPE, one needs to load an ontology in GATE before loading the JAPE transducer. Once the ontology is known to the system, it can be set as the value for the optional ontology parameter for the JAPE grammar. Doing so alters slightly the way the matching occurs when the grammar is executed. If a transducer is ontology-aware (i.e. it has a value set for the 'ontology' parameter) it will treat all occurrences of the feature named class differently from the other features of annotations.

The ontology does not normally affect actions on the right hand side of JAPE rules, but when Java is used on the right hand side, then the ontology becomes accessible via a local variable named ontology, which may be referenced from within the right-hand-side code.

Figure 4.2: Results of running the default Gazetteer over the GATE User manual: only the exact matches from the ORRG are annotated, resulting in skipping most of the plural forms such as *'annotations'* or *'Jape Tranducers'*

As discussed above, a precondition for using a Flexible Gazetteer over a document is to have done some basic pre-processing first. The analysis pipeline (also refered to as *Ontology Resource Finder (ORF) Application*) includes the following language processing components (see figure 4.4):

- Tokeniser
- Sentence Splitter



If a domain ontology is available at the time of building an application, using it in conjunction with the JAPE transducers can significantly simplify the set of grammars that need to be written.

Lookup

URI	http://gate.ac.uk/ns/gate-ontology#JAPETransducer
majorType	
type	class

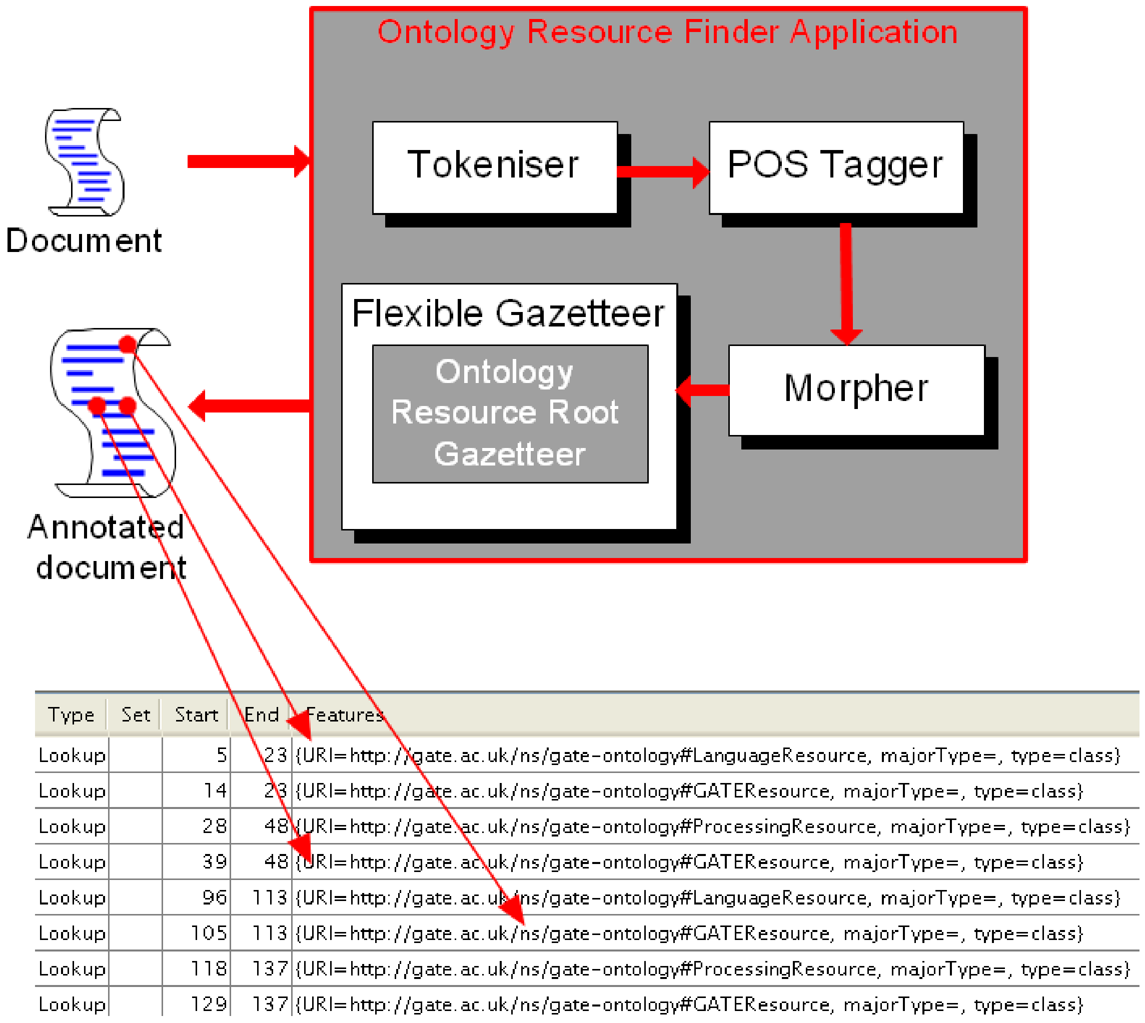
ontology, which may be referenced from within the right-hand-side code.

Figure 4.3: Results of running the Flexible Gazetteer over the GATE User manual: *Jape Transducers* (plural) is annotated although the list of relevant terms created in the previous section and added to the ORRG contains singular form - *Jape Transducer*

- POS Tagger
- Morphological Analyzer
- Flexible Gazetteer
- (optionally) OntoRes Annotator

The input for ORF Application is a set of documents that will be annotated w.r.t. the domain ontology. The output is the documents with annotations of type **Lookup**, each of which contains features 'URI' identifying the URI of the ontology resource they refer to, and 'type' identifying the type of the ontology resource (i.e., class, instance or property).

As **Lookup** annotations are created by running any Gazetteer, we created a new component, called OntoResAnnotator, which renames all annotations of type **Lookup** to **OntoRes**, if they are created by ORRG. This differentiation is important as gazetteers are used frequently in information extraction pipelines, and if one adds for example, another Gazetteer to annotate some key phrases such as *is a kind of* or *is a*, they would also be marked as **Lookup** annotations. However, if no other gazetteers are used, then the use of the OntoResAnnotator is optional. Figure 4.5 illustrates running the application with it over the same document shown on figure 4.3.



Type	Set	Start	End	Features
Lookup		5	23	{URI=http://gate.ac.uk/ns/gate-ontology#LanguageResource, majorType=, type=class}
Lookup		14	23	{URI=http://gate.ac.uk/ns/gate-ontology#GATEResource, majorType=, type=class}
Lookup		28	48	{URI=http://gate.ac.uk/ns/gate-ontology#ProcessingResource, majorType=, type=class}
Lookup		39	48	{URI=http://gate.ac.uk/ns/gate-ontology#GATEResource, majorType=, type=class}
Lookup		96	113	{URI=http://gate.ac.uk/ns/gate-ontology#LanguageResource, majorType=, type=class}
Lookup		105	113	{URI=http://gate.ac.uk/ns/gate-ontology#GATEResource, majorType=, type=class}
Lookup		118	137	{URI=http://gate.ac.uk/ns/gate-ontology#ProcessingResource, majorType=, type=class}
Lookup		129	137	{URI=http://gate.ac.uk/ns/gate-ontology#GATEResource, majorType=, type=class}

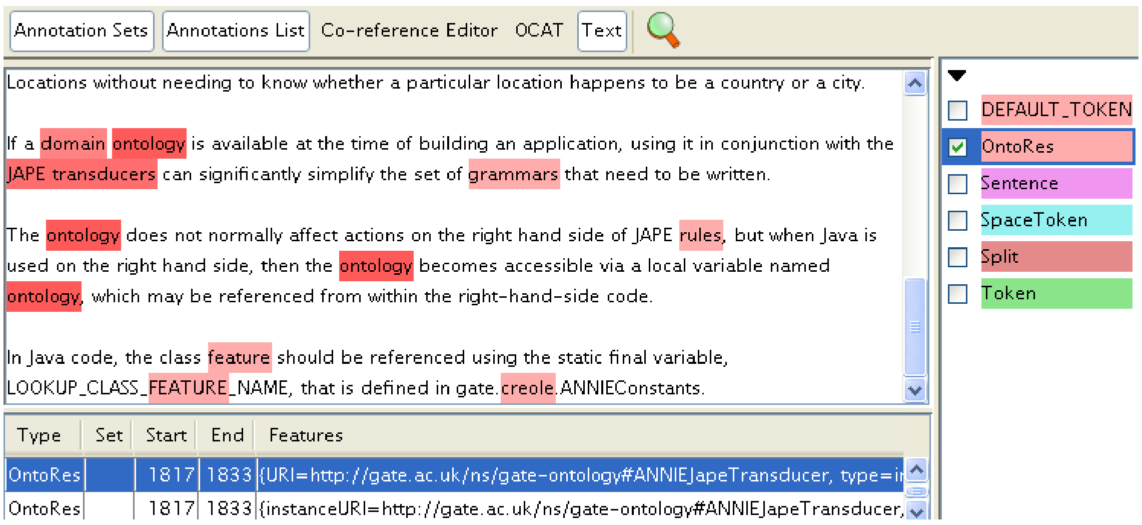
Figure 4.4: Running Ontology Resource Finder Application

4.1.3 Resolving Conflicts - A Challenging Ambiguity Problem

Human language itself is well-known for its ambiguity [CP82]. It is possible to use the same expression in different context and express the totally different meaning. Running the ORF analysis pipeline can result in more than one annotation over the same token or a set of tokens, which need to be disambiguated.

As we do not use any filtering during the process of annotating the documents, it needs to be done in a later stage. The most common disambiguation rule is to give priority to the longest matching annotations. We consider an annotation longer than the other one when

- the start offset node is equal or smaller than the start offset node for the other one



Locations without needing to know whether a particular location happens to be a country or a city.

If a domain ontology is available at the time of building an application, using it in conjunction with the JAPE transducers can significantly simplify the set of grammars that need to be written.

The ontology does not normally affect actions on the right hand side of JAPE rules, but when Java is used on the right hand side, then the ontology becomes accessible via a local variable named ontology, which may be referenced from within the right-hand-side code.

In Java code, the class feature should be referenced using the static final variable, LOOKUP_CLASS_FEATURE_NAME, that is defined in gate.creole.ANNIEConstants.

Type	Set	Start	End	Features
OntoRes		1817	1833	{URI=http://gate.ac.uk/ns/gate-ontology#ANNIEJapeTransducer, type=i
OntoRes		1817	1833	{instanceURI=http://gate.ac.uk/ns/gate-ontology#ANNIEJapeTransducer,

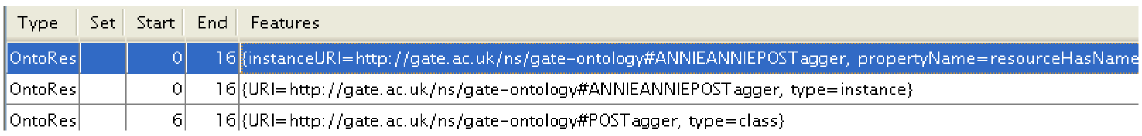
Figure 4.5: Running Ontology Resource Finder Application with additional Processing Resource - OntoResAnnotator. All annotations previously annotated as a **Lookup** type are now transformed to the new annotation type - **OntoRes**.

and

- when the end offset node is greater than or equal as the end offset node for the second annotation.

For example, there is an instance with assigned label with value *ANNIE POS Tagger* inside the GATE domain ontology. This expression comprises the label for the class *POS Tagger* as well, as the class has assigned label *POS Tagger*.

When a document contains the text 'ANNIE POS Tagger', then there will be several annotations of type **OntoRes** indicating that there is more than one resource in the ontology with this name. In a graphical viewer, they will appear as overlapped markup (see Figure 4.6)



Type	Set	Start	End	Features
OntoRes		0	16	{instanceURI=http://gate.ac.uk/ns/gate-ontology#ANNIEANNIEPOSTagger, propertyName=resourceHasName
OntoRes		0	16	{URI=http://gate.ac.uk/ns/gate-ontology#ANNIEANNIEPOSTagger, type=instance}
OntoRes		6	16	{URI=http://gate.ac.uk/ns/gate-ontology#POSTagger, type=class}

Figure 4.6: Annotations of type **OntoRes** for input string 'ANNIE POS Tagger'

As the annotation referring to 'ANNIE POS Tagger' text inside the document has the start offset smaller than the start offset for the annotation referring to 'POS Tagger' text, and the same end offset, we consider it longer and give it a priority. Inside the GATE domain ontology, *ANNIE POS Tagger* is an instance of the class *POS Tagger* and *POS Tagger* is a class with four instances, one of them being the *ANNIE POS Tagger*. Therefore, in this case, it is possible to disambiguate the mentions to that of the correct instance.

This disambiguation rule is based on the heuristic that longer names usually refer to the more specific concepts whereas shorter ones usually refer to the more generic term. However, as this might be domain-specific, it is therefore left in a separate, optional, filtering phase, which can be disabled easily.

4.2 An Example of Running the Key Concept Identification Tool

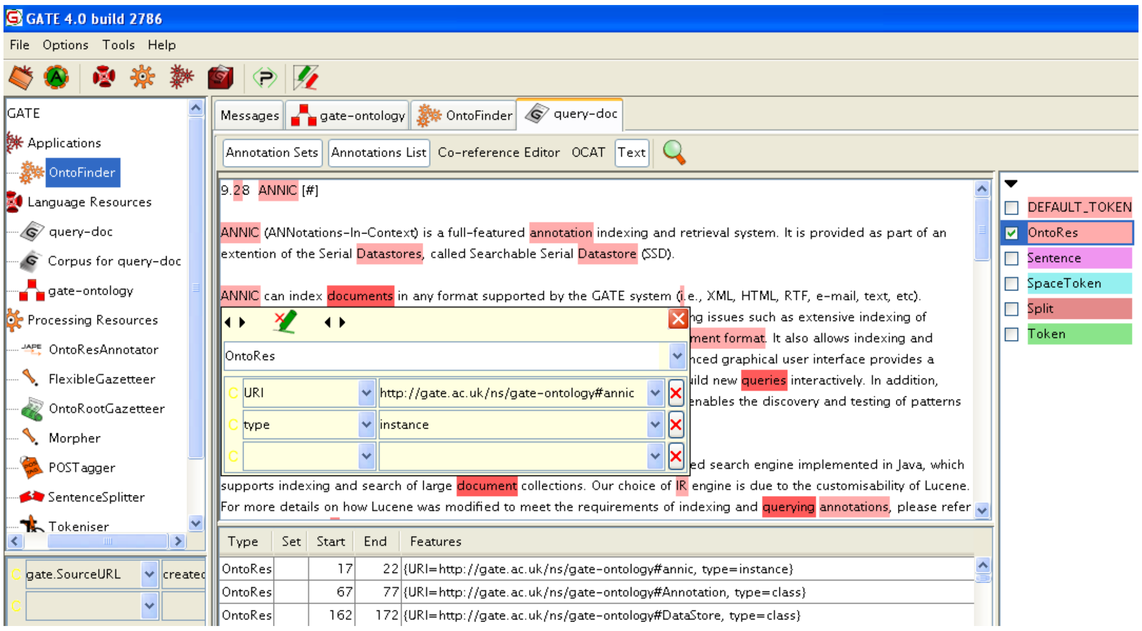
As KCIT is implemented as a pipeline within GATE, we will demonstrate it running with the GATE GUI environment, so that the results are visible inside the Annotation Editor of GATE. The example is using the GATE Domain Ontology, used to annotate the GATE User Manual document (see Figure 4.7).

The GATE Domain Ontology describes concepts and relations regarding the GATE legacy software, and also includes some of the terms that are in a way related to GATE, such as GATE developers, publications, and the like. This ontology was created as part of the GATE case study in TAO (WP6) and will be defined in detail in deliverable D6.2. It is available at: http://gate.ac.uk/ns/gate-ontology.

Running KCIT against this ontology and over a relevant document such as GATE User Manual available online at http://gate.ac.uk/sale/tao/index.html will result in documents that are annotated with key concepts that are GATE domain specific.

On the right side of the picture there is a list of the annotation names that is created after running KCIT. The most important ones are the **OntoRes** annotations, all of which have features *URI* and *type*. URI refers to the actual URI in the ontology, and 'type' refers to the type of the resource inside the ontology, e.g., an instance, a class, a property. Some of the annotations with appropriate features are visible inside the table in the lower part of Figure 4.7.

Apart from running KCIT within GATE, it can also be used as a stand-alone, batch process from the command line. However, for ease of integration with the TAO Suite, we have focused our efforts on delivering KCIT and all other content augmentation components as web services. The first prototype of these services is already running at `http://gate.ac.uk/ca-service/services/CAService` and the WSDL can be obtained from `http://gate.ac.uk/ca-service/services/CAService?wsdl`.



9.28 ANNIC [#]

ANNIC (ANNotations-In-Context) is a full-featured annotation indexing and retrieval system. It is provided as part of an extention of the Serial Datastores, called Searchable Serial Datastore (SSD).

ANNIC can index documents in any format supported by the GATE system (i.e., XML, HTML, RTF, e-mail, text, etc).
ng issues such as extensive indexing of
ment format. It also allows indexing and
nced graphical user interface provides a
ild new queries interactively. In addition,
enables the discovery and testing of patterns

ed search engine implemented in Java, which
supports indexing and search of large document collections. Our choice of IR engine is due to the customisability of Lucene.
For more details on how Lucene was modified to meet the requirements of indexing and querying annotations, please refer

OntoRes	
URI	http://gate.ac.uk/ns/gate-ontology#annic
type	instance
gate.SourceURL	created

Type	Set	Start	End	Features
OntoRes		17	22	{URI=http://gate.ac.uk/ns/gate-ontology#annic, type=instance}
OntoRes		67	77	{URI=http://gate.ac.uk/ns/gate-ontology#Annotation, type=class}
OntoRes		162	172	{URI=http://gate.ac.uk/ns/gate-ontology#DataStore, type=class}

Figure 4.7: Running Key Concept Identification Tool against GATE Domain Ontology over the GATE User Manual

4.3 Related work

Semantic annotation is extensively performed by the knowledge management platforms that are developed up to date. These management platforms use the process of semantic annotation as a precondition to performing some other tasks (e.g., a knowledge base enrichment). The process itself is performed: manually, automatically, or as a combination of the two - usually referred to as semi-automatically. As KCIT is a tool for producing ontology-aware annotations over legacy software content we give an overview of the similar tools: 1) for performing Content Augumetation task with regards to a domain ontology 2) applied to software systems/software engineering tasks.

The most similar already developed tool to the KCIT is Apolda - GATE plugin for producing ontology-aware annotations. Apolda (Automated Processing of Ontologies with Lexical Denotations for Annotation) annotates a document on the very similar way to the gazetteer, with a difference of taking the terms from an ontology and not from a list [WBGH07]. During annotation process Apolda considers set annotation properties on concepts. Our approach differs in that of considering not only concepts but also relations between them. We also consider values of all set properties for all existing resources. Our approach is more generic than that of Apolda, as we use a Morphological Analyzer twice: 1) to lemmatize the extracted content from ontology resources 2) to lemmatize the document content when running the tool over the document. With Apolda the use of a

Morphological Analyzer is possible only once - for lemmatizing document content.

MagPie [DDM04] is a tool for interpretation of web pages and is used as a plugin within a standard web browser. Magpie automatically associates an ontology-based semantic layer to web resources. However, it is not possible to use it on the documents not supported by a Web browser (e.g., Word format). Considering content augumentation process our approach is more flexible as they do not lemmatize the content at all. However, they focus more on some other tasks such as using the results for employing semantic web services.

KIM [KPO$^+$04] performs semantic annotation automatically in respect to their ontology, by identifying the Key Phrases and Named Entities. As a Name Entity(NE) they consider people, organizations, locations, and others referred to by name. They use GATE for Name Entity Recognition, where for this task they use the ontology to link an arbitrary Token(or a set of Tokens) to the particular URI. In the next stages, they use created annotations for semantic indexing and retrieval, co-occurence, and popularity trend analysis. To extend their scope out of already existing concepts supported by their ontology, it is a mandatory to extend the ontology they extended, namely PROTON ontology (http://proton.semanticweb.org). KIM's approach differs from ours in that of using the exact names without any morphological analysis, and also in considering only labels associated with classes that are developed inside their ontology for representing the names (e.g., class *Alias*).

At the Helsinki University of Technology in Finland they developed Poka - a framework for automatic annotation [VHA06]. They use this framework to develop a domain specific tools. Poka extracts ontological concepts and person names from the input text. They use Finnish General Upper Ontology YSO (http://www.seco.tkk.fi/ontologies/yso/) based on the widely used Finnish General Thesaurus maintained by The National Library of Finland. They consider lemmatized extraction of ontology resources, but it is limited to persons, places and common nouns. In comparison to theirs, our tool is more portable and generic as it can be used with any ontology, as long as it is populated with relevant data, e.g.,values of *rdf:label* property for ontology resources - without any further intervention.

Some of existing tools address a similar problem to ours, applying a slightly different approach. Dhruv [ASHK06] is a prototype semantic web system developed for Open Source Software Community to support bug resolution. The main differences of their approach to ours is that of using general purpose ontologies, whereas in TAO we focus on developing application-specific ontology. Additionally, in Dhruv they only populate the ABox (i.e., instances) whereas we focus on populating both ABox and TBox (i.e., ontology). Finally, Dhruv is aimed to be used by developers, whereas in TAO we focus at higher/component level.

In [WZR07] they are focusing on reducing the conceptual gap between source code and software documentation by integrating them into a formal ontological representation. This representation assist maintainers for performing typical software maintenance tasks. Their work differs from ours in that of having already developed generic ontologies that are further being automatically populated by application-specific data. In TAO we create ontologies for each software separately and populate them semi-automatically - after we recognize candidates for ontology population (i.e. instances) they need to be verified by a domain expert in order to be included. In [WZR07] they use Ontogazetteer to perform lookup over software code and documentation. Ontogazetteer is a language processing component developed as a plugin for GATE [CMB$^+$06]. It provides the possibility to produce automatic annotations against an ontology, and requires maintaining lists where the data about the appropriate links to ontology resources are stored. The main difference of ORRG gazetteer used in KCIT tool and Ontogazetteer is that of performing everything dynamically in ORRG: the list is created on-fly, contains lemmatized content, and stored in memory for a better performance.

4.4 Future work

There is a room for future improvements in KCIT:

1. **Better analysis of longer relevant terms**. For some ontology resources it would be more efficient if the value of their properties are analyzed so that they are included in the dynamic gazetteer list only partially. For example, values of rdf:comment property usually contain long explanations of what is the resource about. The value of this property could be analyzed and included only in part, as currently the whole value of this property is lemmatized and included in the gazetteer list.

2. **Configurability**. Enabling configuration of the tool would be of a great importance. At the moment, KCIT performs automatically Content Augmentation task without the possibility to set whether one wants to include a resource URI or not, property values, and the like. Providing the possibility to use specific properties, or a specific type of resources (e.g., only classes, only properties, or only a specific property) would result in a greater configurability of this tool.

3. **Detecting spelling errors**. Legacy documentation, especially the one created by OCR tools and the like can contain spelling errors. Using some of the available similarity metrics for detecting the similarity between legacy content and the one that appears in the ontology can help in detecting the spelling errors. This would make the KCIT more effective.

4. **Matching synonims**. Coupling KCIT with some of the available tools for matching synonyms (e.g., using WordNet [Fel98] or Google distance [GtKAvH07]). This would lead to the possibility to annotate the words that are not extracted from the ontology resources, but are in relation with them. For example, if the ontology contains concept with the name *desk*, if the word *table* appears in the document it would be annotated, based on the synonym relationship with *desk*.

We will include the first two features in the future work of TAO while for the last one we might do some experiments using term service provided by JSI.

Chapter 5

Information Consolidation

As defined in Section 2.3, the information consolidation is the process during which semantic annotations, created in the concept identification stage are analysed, all remaining ambiguities are removed, and, where applicable, new instances and properties are identified for ontology population.

The information consolidation tools that we implemented are as follows:

New mention discovery: discover un-annotated mentions which could be either new instances in the ontology or nominal and pronominal coreference mentions of instances already in the ontology.

Reference resolution: determine the URI of coreferent new mentions or flag as new candidate instance to be added to the ontology via ontology population. Such candidate instances can then either be added automatically or shown to the user for verification. The choice between these two strategies is application dependent and TAO D3.2 will provide tools to support the manual verification step.

5.1 New Mention Discovery

The concept identification tools, described in the previous chapter, are designed to only discover mentions of resources from the domain ontology based on their lexicalisation. In addition to that, one also needs to discover other mentions, such as new instances and also referring expressions, not already annotated in ealier stages. For instance, the expression ”the parser” can refer to any of the several syntactic parsers in GATE, but it might not have been matched during concept identification because it is unclear from the phrase itself which of the instances it refers to.

5.1.1 Identifying New Candidates for the Ontology

The identification of candidates for new instances in the ontology is carried out using the following patterns. In the patterns, OntoTerm denotes the annotation type produced by KCIT tools, which contains two features – URI and type (class, instance, property). NN and NNS are part-of-speech-tags denoting noun and plural noun respectively, whereas NNP and NNPS are the tags for proper noun and proper noun in plural.

1. `OntoTerm.type == class NN|NNS`: matches a mention of a class from the ontology followed by a noun, e.g., gazetteer lists, ontology viewer. These are marked as candidates for a new class.

2. `OntoTerm.type == instance NN|NNS`: matches a mention of an instance from the ontology, follwed by a noun, e.g., ANNIE application. These are marked as candidates for new instances.

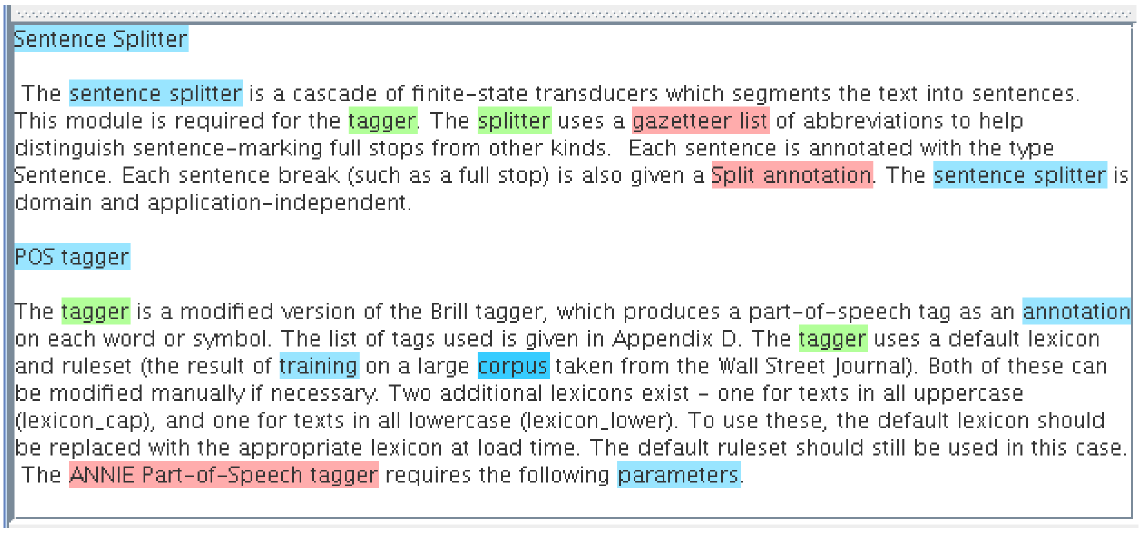
3. `NN|NNS|NNP|NNPS OntoTerm.type == class` : matches a noun or a proper noun followed by a mention of a class from the ontology. These are marked as candidates for new instances, e.g., “HipHep tagger”.

4. `OntoTerm OntoTerm.type == class`: two mentions one after another. This is marked as a candidate, but at this stage it is left open whether it’s candidate instance, sub-class of the second class, or simply a new lexicalisation of an existing instance/class. Therefore, it will be investigated further during the coreference step. If the first OntoTerm is of type instance, it is almost certainly a new lexicalisation, but if fuzzy matching fails, then it will be proposed as a new instance.

5. `''the'' OntoTerm.type == class`: it is marked as a reference resolution candidate, because expressions such as ”the parser” often can be disambiguated as referring to an actual instance of that class, mentioned earlier in the text (see Figure 5.1).

Figure 5.1 shows a portion of the GATE User manual, which is annotated with mentions of ontology resources (in blue). In addition, terms that need disambiguation (e.g., the ”splitter”) are marked in green. Following reference resolution such candidate terms will either be disambiguated as pointing to an existing instance or class in the ontology (identified with a URI), or they will be flagged as candidates for new instances in the ontology.

Candidates for new classes and instances are highlighted in red. For instance, “gazetteer list” is a candidate class, “Split annotation” – a candidate instance, and “ANNIE Part-of-speech Tagger” - a candidate instance as well. In the case of the latter, during reference resolution phase it will be changed as a reference to an already existing instance *ANNIE POS Tagger*, due to their linguistic similarities (i.e., one is abbreviation of the other).



Sentence Splitter

The sentence splitter is a cascade of finite-state transducers which segments the text into sentences. This module is required for the tagger. The splitter uses a gazetteer list of abbreviations to help distinguish sentence-marking full stops from other kinds. Each sentence is annotated with the type Sentence. Each sentence break (such as a full stop) is also given a Split annotation. The sentence splitter is domain and application-independent.

POS tagger

The tagger is a modified version of the Brill tagger, which produces a part-of-speech tag as an annotation on each word or symbol. The list of tags used is given in Appendix D. The tagger uses a default lexicon and ruleset (the result of training on a large corpus taken from the Wall Street Journal). Both of these can be modified manually if necessary. Two additional lexicons exist - one for texts in all uppercase (lexicon_cap), and one for texts in all lowercase (lexicon_lower). To use these, the default lexicon should be replaced with the appropriate lexicon at load time. The default ruleset should still be used in this case. The ANNIE Part-of-Speech tagger requires the following parameters.

Figure 5.1: Highlighted in green are mentions for reference disambiguation, in red – new candidate ontology resources, in blue – ontology resources annotated by KCIT

5.2 Reference Resolution for Ontology Population

The reference resolution task consists of assigning the most appropriate URI from the given domain ontology to any candidate term, which does not have one already. It also analyses class mentions to check whether they should be changed into instance mentions, if they are part of a nominal refering expression, e.g., "this tokeniser" might initially be assigned a URI of the tokeniser class, but from the context it needs to be disambiguated to one of the two tokeniser instances: English tokeniser or default (Unicode) tokeniser.

This task, while bearing similarities to anaphora resolution, is somewhat different, because it uses knowledge from the ontology and also disambiguates with respect to the ontology. In this research we focused on resolving definite noun phrases by assigning the URI of the correct ontology resource. We have not yet considered the resolution of *it* and other similar pronouns, largely because they are not as prevalent as definite noun phrases.

Our approach to reference resolution is similar to the class of "knowledge poor" anaphora resolution approaches. Such methods are intended to provide inexpensive and fast implementations that do not rely on complex linguistic knowledge, yet they work with sufficient success rate for practical tasks (e.g., [Mit98]).

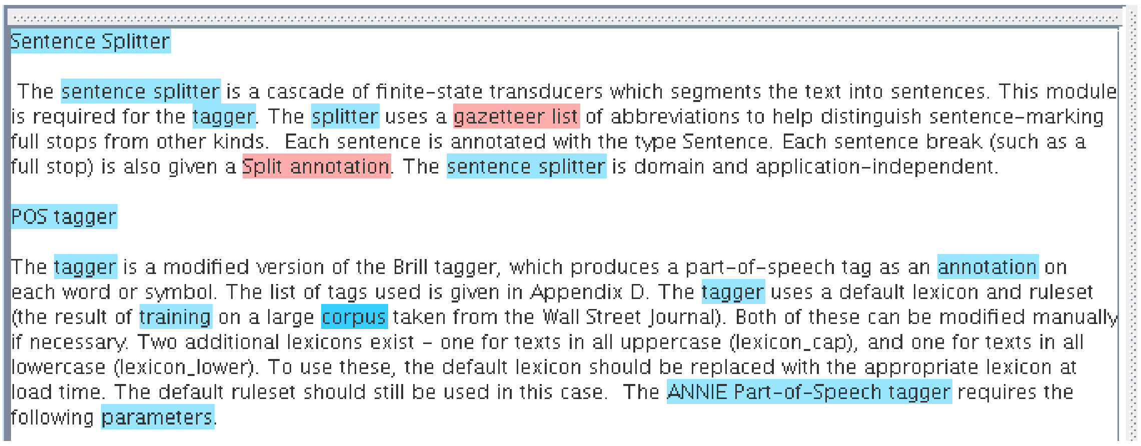
The method is similar to other salience-based approaches, which perform resolution following the steps:

- identification of all antecedents and organising them in a stack structure, so at any given point one can find the most recent, compatible antecedent of a given ontological class/instance

- inspecting the context for candidate antecedents that satisfy a set of consistency restrictions based on the ontology
- selection of the most salient, i.e., most recent compatible antecedent on that basis
- assignment of the appropriate URI from the domain ontology

As we aim to process large amounts of text efficiently, we do not employ any syntactic parsing or discourse analysis to identify deeper relationships between candidates and the set of compatible antecedents.

The actual implementation is very similar to our algorithm for pronoun resolution [DBCM05], the difference being that here the antecedents are not named entities from mentions of ontological resources, and also that we carry out disambiguation of noun phrases, instead of pronouns.


Sentence Splitter

The sentence splitter is a cascade of finite-state transducers which segments the text into sentences. This module is required for the tagger. The splitter uses a gazetteer list of abbreviations to help distinguish sentence-marking full stops from other kinds. Each sentence is annotated with the type Sentence. Each sentence break (such as a full stop) is also given a Split annotation. The sentence splitter is domain and application-independent.

POS tagger

The tagger is a modified version of the Brill tagger, which produces a part-of-speech tag as an annotation on each word or symbol. The list of tags used is given in Appendix D. The tagger uses a default lexicon and ruleset (the result of training on a large corpus taken from the Wall Street Journal). Both of these can be modified manually if necessary. Two additional lexicons exist - one for texts in all uppercase (lexicon_cap), and one for texts in all lowercase (lexicon_lower). To use these, the default lexicon should be replaced with the appropriate lexicon at load time. The default ruleset should still be used in this case. The ANNIE Part-of-Speech tagger requires the following parameters.

Figure 5.2: The disambiguated mentions of ontological resources are in blue, whereas the new candidate classes and instances appear in red

Figure 5.2 is showing the results from the noun phrase disambiguation stage, where definite noun phrases have been assigned URIs of the respective ontological resources. In addition, the proposed new instances and classes are highlighted in red.

For the time being, we have made a design decision not to add these to the ontology automatically, but to present them to the user. Consequently, the result of the consolidation phase are two sets of metadata:

- List of all instances mentioned in the given document content, with their URI and information about the place in the text where they appear (as offsets). For the example above these would be the URIs of “ANNIE POS Tagger” and “Sentence Splitter”, with the offsets of all their mentions in the text.
- List of all newly proposed instances and classes. In our case, these are: “gazetteer list” (class) and “Split annotation” (instance).

5.3 Discussion and Future Work

We have created a prototype information consolidation tool, which has been experimented with on a subset of the GATE manuals. During the evaluation task, we plan to undertake quantitative evaluation of its effectiveness and improve the algorithms accordingly.

We also plan to investigate the interaction with ontology learning approaches, both those developed within TAO, i.e., LATINO and ONTOGEN, but also others. For instance, using Hearst patterns, as proposed in the Text2Onto approach [CV05].

Chapter 6

First Experiments with Non-textual Legacy Content

Legacy software systems consist primarily of textual content, i.e., source code, code documentation (JavaDoc), user guide, postings on online forums, etc. Nevertheless, there are also plenty of images, which are also very important for the understanding of the software application, e.g., dataflow diagrams, UML diagrams, architecture diagrams, and screen shots. In order to apply the content augmentation tools to these images, first one needs to extract the relevant textual content via OCR (Optical Character Recognition).

OCR is a fairly mature and widely used technology, which however has mainly been developed and tested to support the automatic conversion of scanned documents into text. However, our findings (see Section 6.1) have proven that images in software applications are rather different and far more challenging:

- the layout, shapes, and arrows in the charts and the richness of the screen shots are hard to interpret by the current layout algorithms, which are mostly geared towards well-formatted texts and tables;
- in the screen shots some text is highlighted, which means that it is light-coloured text on dark background, but the rest of the text is as usual (i.e., in dark font), which creates problems for the OCR tools;
- in order to keep their size down, the resolution of many of the images is lower than 300dpi, which is the minimum resolution recommended by some OCR tools;
- depending on the specificity of the software application, the terms appearing in the images might not be present in the vocabulary of the OCR system, which in some cases leads to degraded performance.

The task which we address here is to identify automatically a list of ontology resources (classes, instances, properties) which are mentioned in the images, i.e., flickr-style image annotation. An even more challenging task would be assign them to a region of the image, where they appear, but as we are using off-the-shelf OCR tools for pre-processing, that information has not been made available from these tools.

6.1 OCR Pre-processing: Tool Evaluation and Recommendations

6.1.1 The Test Data

In order to promote repeatability and avoid copyright problems, we chose a set of 12 images from the GATE online user guide – 4 diagrams and 8 screen shots. Figure 6.1 shows two of the diagrams – the left one is a workflow diagram, describing some system components, wheareas the right one is a standard UML diagram.

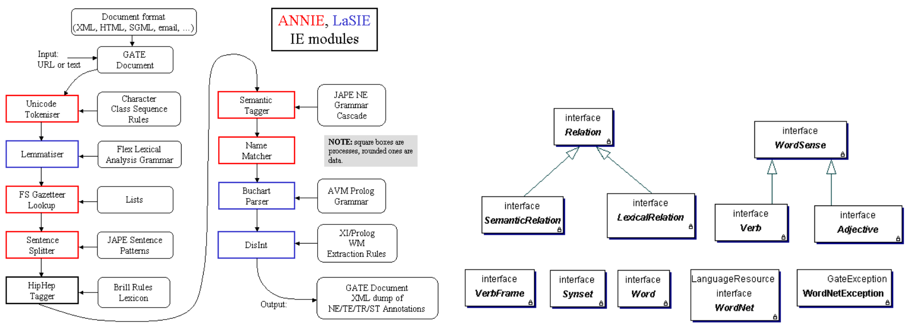
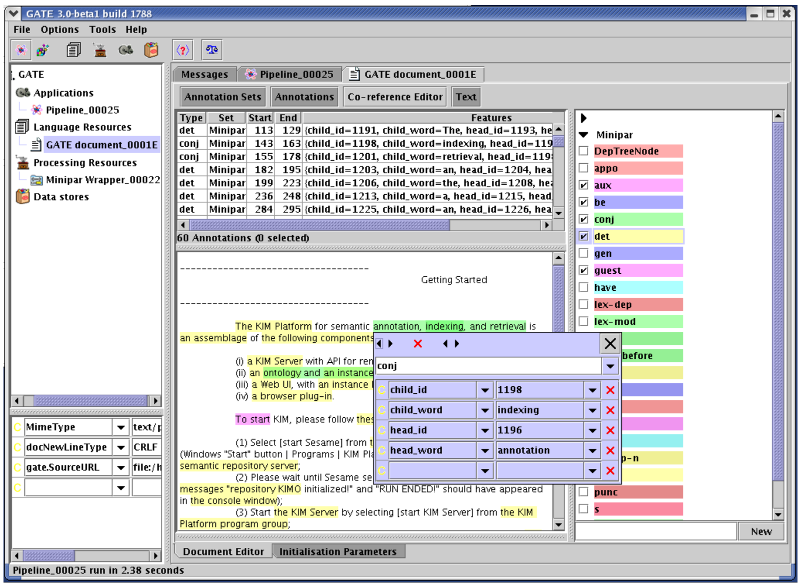


Figure 6.1: Test data: a workflow and a UML diagram

A sample screen shot of the main user interface and some language processing results appear in Figure 6.2. There are several screen shots of the main user interface, all demonstrating different functionalities. The challenging aspect here is to recognise the GATE-specific terms, especially as they are likely to be out-of-vocabulary words for the OCR tools and also, some of them are immediately followed by numeric identifiers (e.g. GATE document_0003E).

Figure 6.3 shows a screen shot of one of GATE's tree-like data viewers, which are even harder for the OCR tools, as they combine graphics and text quite close to each other. Also, the names of some of the GATE terms are slightly truncated in the screen shot itself (e.g., Processing Res(ource)), which makes their correct OCR recognition even harder. This problem is not specific to this screen shot only and is due to the author's effort to keep the images as small as possible while still showing all relevant information.

All test data was originally in PNG format, however, we had to transform it into TIFF, due to problems with formats supported by some of the OCR tools. This conversion step,



Type	Set	Start	End	Features
det	Minipar	113	129	{child_id=1191, child_word=The, head_id=1193, he
conj	Minipar	143	163	{child_id=1198, child_word=indexing, head_id=119
conj	Minipar	155	178	{child_id=1201, child_word=retrieval, head_id=119
det	Minipar	182	195	{child_id=1203, child_word=an, head_id=1204, hea
det	Minipar	199	223	{child_id=1206, child_word=the, head_id=1208, hea
det	Minipar	236	248	{child_id=1213, child_word=a, head_id=1215, head
det	Minipar	284	295	{child_id=1225, child_word=an, head_id=1226, hea

60 Annotations (0 selected)

```
-----------------------------------
                    Getting Started
-----------------------------------

          The KIM Platform for semantic annotation, indexing, and retrieval is
an assemblage of the following components

          (i) a KIM Server with API for ren
          (ii) an ontology and an instance
          (iii) a Web UI, with an instance
          (iv) a browser plug-in.

          To start KIM, please follow thes

          (1) Select [start Sesame] from t
(Windows "Start" button | Programs | KIM Pla
semantic repository server;
          (2) Please wait until Sesame se
messages "repository KIMO initialized!" and "RUN ENDED!" should have appeared
in the console window);
          (3) Start the KIM Server by selecting [start KIM Server] from the KIM
Platform program group;
```

conj

child_id	1198
child_word	indexing
head_id	1196
head_word	annotation

MimeType	text/p
docNewLineType	CRLF
gate.SourceURL	file:/h

Pipeline_00025 run in 2.38 seconds

Figure 6.2: Test data: A screen shot of the main user interface

when necessary, can be automated by means of a script. We carried it out on Ubuntu Linux version 7.04, using netpbm tools for image format conversions.

6.1.2 Open-source OCR Tools

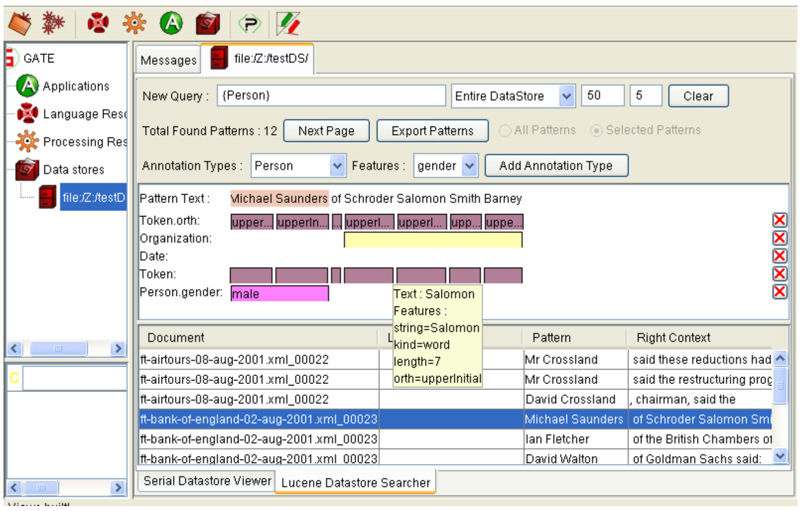
First we experimented with two open-source OCR tools – GOCR[1] and Tesseract OCR[2] – which were chosen because they have excellent cross-platform support and good user documentation. Both are command-line tools, so, if appropriate, can easily be made available as a web service. Tesseract was formerly developed by Hewlett Packard and was among the top 3 engines in the 1995 UNLV Accuracy test, then had little development until 2006, when it was picked up by Google.

Our tests were carried out on Ubuntu Linux 7.04 and we installed the two tools via the Synaptic package manager (the package names are gocr and tesseract-ocr respectively).

We tested the performance on the test images, after they were converted in the required formats: gocr requires p*m (e.g., pbm, pgm), whereas tesseract supports tiff. We also experimented with colour and black& white versions of the images.

Overall, the results were very unsatisfactory with very few of the words recognised correctly. In general, both tools performed slightly better on the black& white versions, than on the colour ones. Diagrams were also handled better with at least some words recognised on the workflow diagrams by both tools.

[1]http://jocr.sourceforge.net/
[2]http://code.google.com/p/tesseract-ocr/



New Query : {Person}   Entire DataStore   50   5

Total Found Patterns : 12

Annotation Types : Person   Features : gender

Pattern Text : Michael Saunders of Schroder Salomon Smith Barney

Token.orth: upper... upperIn... ... upperI... upperI... upp... uppe...

Organization:

Date:

Token:

Person.gender: male

Text : Salomon
Features :
string=Salomon
kind=word
length=7
orth=upperInitial

Document	L	Pattern	Right Context
ft-airtours-08-aug-2001.xml_00022		Mr Crossland	said these reductions had
ft-airtours-08-aug-2001.xml_00022		Mr Crossland	said the restructuring proc
ft-airtours-08-aug-2001.xml_00022		David Crossland	, chairman, said the
ft-bank-of-england-02-aug-2001.xml_00023		Michael Saunders	of Schroder Salomon Smi
ft-bank-of-england-02-aug-2001.xml_00023		Ian Fletcher	of the British Chambers of
ft-bank-of-england-02-aug-2001.xml_00023		David Walton	of Goldman Sachs said:

Figure 6.3: Test data: screen shot of a specialised data viewer with some truncated GATE terms

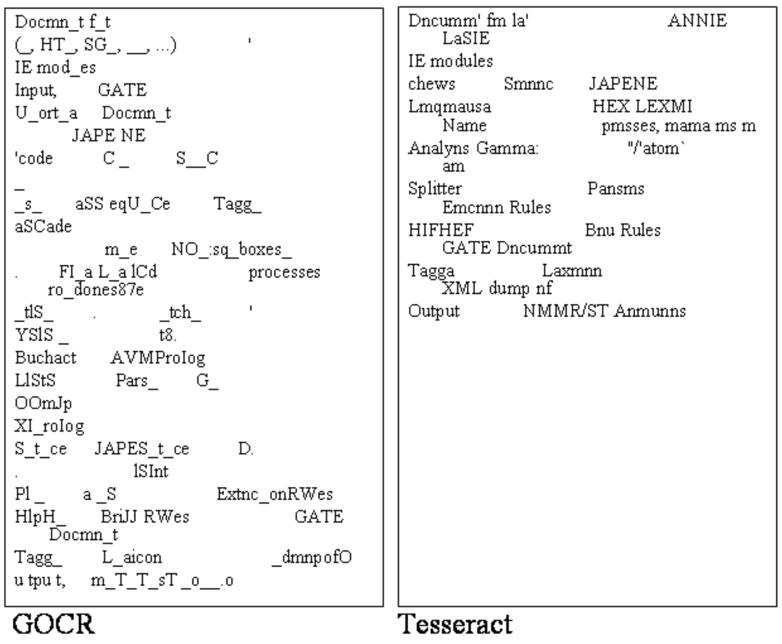
For example, Figure 6.4 shows the results on the ANNIE workflow diagram shown in Figure 6.1 above. However, tesseract had problems with the UML diagram with no legible words produced, whereas GOCR performed better, although most recognised words were not complete, i.e., had some characters replaced with underscores (e.g., inte_ace, _anquaqeResource). This poorer performance of tesseract is due to the fact that it does not recognise page layout or images, which is absolutely vital in our case. So while previous experiments[3] on pure text have shown tesseract to outperform GOCR, in fact, on our image diagrams GOCR is clearly better.

The latest version 2.0 of tesseract also allows users to extend its lexicons with new words, so we experimented with adding all gate terms from the ontology to tesseract's user lexicon, but this did not result in a substantial improvement.

With respect to processing of screen shots, both gocr and tesseract had problems with identifying the zones containing text and only processing those. The results improved when the images were cropped to contain only the textual zone relevant to the topic of the image, however, this is time-consuming and cannot be automated. In general, if manual cropping is required, it will actually be faster for the users to annotate the images with the relevant concepts from the GATE ontology, rather than pre- and post-processing them to improve the OCR results.

The overall conclusion is that, at the time of writing, open-source OCR tools do not deal sufficiently well with screen shots and software diagrams, due to problems with layout, colour, low resolution, and unknown terminology. Therefore, their integration in TAO's content augmentation tools is considered undesirable, due to the low quality of

[3]See article at: http://www.linux.com/articles/57222



```
Docmn_t f_t
(_, HT_, SG_, __, ...)             '
IE mod_es
Input,       GATE
U_ort_a    Docmn_t
        JAPE NE
'code        C _        S__C
_
_s_     aSS eqU_Ce       Tagg_
aSCade
             m_e      NO_:sq_boxes_
.      FI_a L_a lCd               processes
     ro_dones87e
_tlS_        .        _tch_        '
YSlS _                t8.
Buchact      AVMProlog
LlStS         Pars_      G_
OOmJp
XI_rolog
S_t_ce    JAPES_t_ce        D.
.                  lSInt
Pl _      a _S                Extnc_onRWes
HlpH_     BriJJ RWes                 GATE
     Docmn_t
Tagg_      L_aicon                _dmnpofO
u tpu t,    m_T_T_sT _o__.o
```

GOCR

```
Dncumm' fm la'                    ANNIE
     LaSIE
IE modules
chews        Smnnc       JAPENE
Lmqmausa                  HEX LEXMI
     Name                   pmsses, mama ms m
Analyns Gamma:                "/'atom`
     am
Splitter                  Pansms
     Emcnnn Rules
HIFHEF                    Bnu Rules
     GATE Dncummt
Tagga              Laxmnn
     XML dump nf
Output          NMMR/ST Anmunns
```

Tesseract

Figure 6.4: GOCR and Tesseract OCR results on the ANNIE workflow diagram

their results.

However, both gocr and tesseract are being developed actively, which in a few years' time is likely to lead to substantial improvements. For instance, tesseract's development roadmap includes integration of two layout engines (OCRopus and Leptonica), which will most likely address some of the problems reported above.

6.1.3 Commercial OCR Tools

There are a large number of commercial OCR tools amongst which one can choose, but our goal here was to experiment with some widely used ones and measure whether they perform significantly better at extracting text from screen shots and software diagrams. As they require each user to purchase their own license, they cannot be provided as part of the TAO Suite or within a TAO content augmentation service. Instead, the user will have to pre-process their images, extract the text and then supply them to the TAO text CA services to index them with respect to the domain ontology.

The ReadIRIS OCR tool[4] was tested under Windows XP, as it only supports Windows and Macintosh platforms. We chose it because it is distributed bundled together with many scanners so even a small company might have a licensed copy with which they can extract the text, without incurring extra costs.

[4]http://www.irislink.com/c2-532/OCR-Software—Product-list.aspx

We also experimented with another widely used commercial OCR tool – ABBYY FineReader[5], which only supports Windows platforms and comes in professional, corporate, large enterprise, and server versions.

ReadIRIS was not able to open PNG files, whereas the layout manager in FineReader did not process them as well as other formats, mostly by having problems locating the text zones correctly. When used on the tiff versions of the images, ReadIRIS did not encounter any problems. FineReader worked better than with the PNG files, but it had problems opening some of the tiff images, so ulimately all images had to be converted to JPEG and then FineReader worked extremely well.

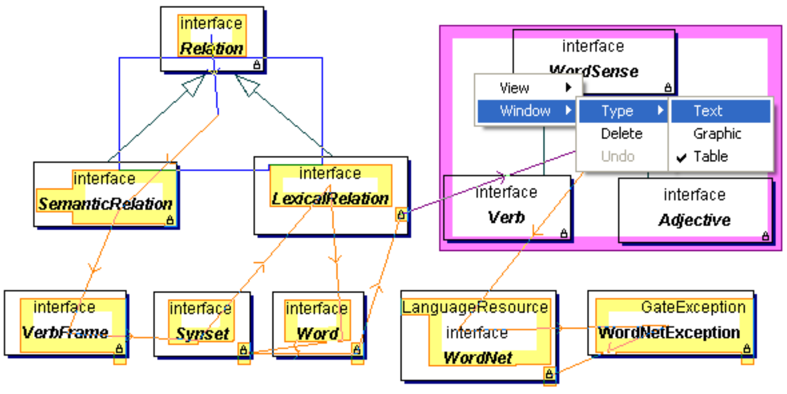


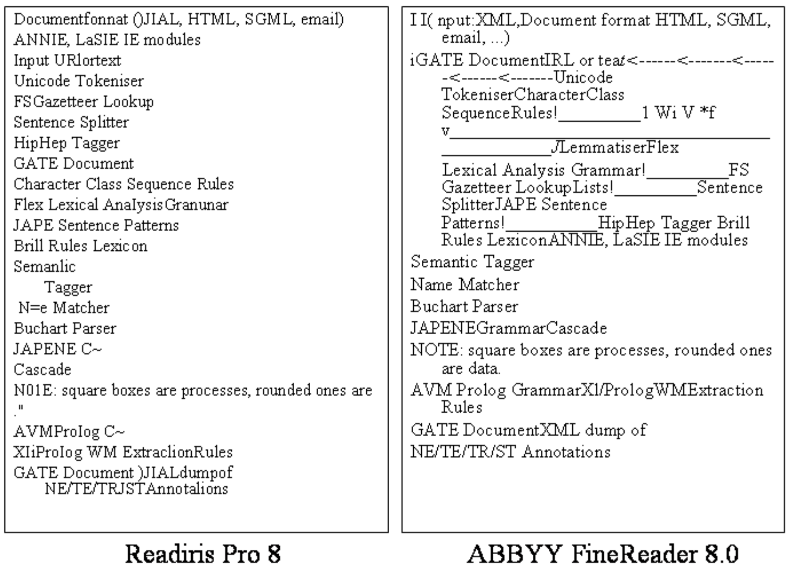
Figure 6.5: Layout recognition step in ReadIRIS

Both ReadIRIS and FineReader performed a layout recognition step during which they divided the screen shots and the diagrams into text, table, and image zones (see Figure 6.5). The automatic results can be corrected easily by the user, as can be seen in the Figure 6.5. However, we chose to run both tools in fully automatic mode, as again, the time spent on manual correction of the layout and OCR results would be at least as long as the time required to tag the images manually with the 5 to 10 relevant domain concepts and properties.

Overall, FineReader performed substantially better than ReadIRIS, both on software diagrams and on screen shots, when the images were supplied in JPEG format, but not in tiff. Both tools cannot be used on Linux platforms, while only ReadIRIS can be used on Macintosh. Diagrams were handled much better than screen shots and neither tool had a problem dealing with the colour versions of the images. The commercial tools also outperformed significantly their open-source counterparts on both kinds of data: screen shots and diagrams.

For example, Figure 6.6 shows the results of both tools on the ANNIE workflow diagram shown in Figure 6.1 above. FineReader is capable of recognising more of the GATE

[5]http://www.abbyy.com/finereader8/?param=44782#f8



```
Documentfonnat ()JIAL, HTML, SGML, email)
ANNIE, LaSIE IE modules
Input URlortext
Unicode Tokeniser
FSGazetteer Lookup
Sentence Splitter
HipHep Tagger
GATE Document
Character Class Sequence Rules
Flex Lexical AnalysisGranunar
JAPE Sentence Patterns
Brill Rules Lexicon
Semanlic
     Tagger
 N=e Matcher
Buchart Parser
JAPENE C~
Cascade
N01E: square boxes are processes, rounded ones are
."
AVMProIog C~
XIiProIog WM ExtraclionRules
GATE Document )JIALdumpof
     NE/TE/TRJSTAnnotalions
```

Readiris Pro 8

```
I I( nput:XML,Document format HTML, SGML,
     email, ...)
iGATE DocumentIRL or teat<------<-------<-----
     -<------<-------Unicode
     TokeniserCharacterClass
     SequenceRules!__________1 Wi V *f
     v_______________________________________
     _____________/LemmatiserFlex
     Lexical Analysis Grammar!__________FS
     Gazetteer LookupLists!__________Sentence
     SplitterJAPE Sentence
     Patterns!___________HipHep Tagger Brill
     Rules LexiconANNIE, LaSIE IE modules
Semantic Tagger
Name Matcher
Buchart Parser
JAPENEGrammarCascade
NOTE: square boxes are processes, rounded ones
     are data.
AVM Prolog GrammarXl/PrologWMExtraction
     Rules
GATE DocumentXML dump of
NE/TE/TR/ST Annotations
```

ABBYY FineReader 8.0

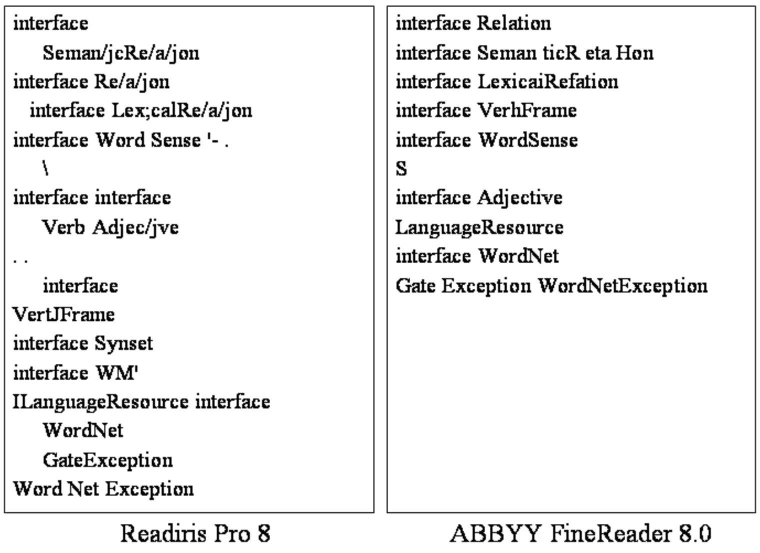
Figure 6.6: ReadIRIS and FineReader results on the ANNIE workflow diagram

terms than ReadIRIS and both tools perform significantly better than the open-source ones (see Figure 6.4).

When run on UML diagrams both tools recognised most of the text, with FineReader making very few errors overall (see Figure 6.7).

The most substantial difference between the two commercial tools appeared on screen shots, where FineReader was capable of identifying much larger portions of the text, including window captions, error messages, and mixed graphics and text. Nevertheless, where GATE-specific terms appeared (e.g., Minipar), both tools had difficulties in recognising these correctly. For instance, the words ”Minipar Wrapper” from the screen shot in Figure 6.2 were recognised as ”Minipai Wiappei” by FineReader and as ”MiniPa,w,appe,” by ReadIRIS. In comparison, ”GATE Applications” on the same image was recognised correctly by both tools, most probably due to both words being in common use and appearing in their dictionaries. Another complication with this particular screen shot is that it has been taken on a Linux platform, which has slightly different fonts and all OCR tools made significantly more mistakes on this image than on all other screen shots, which were taken on Windows machines.

The overall conclusion is that ABBYY FineReader was capable of recognising correctly substantial parts of the relevant text on both screen shots and software diagrams. ReadIRIS was less successful, but if already licensed by a user, it can still be used especially on the software diagrams. We decided to experiment with running the content augmentation tools on the output of both systems, so we can then measure quantitatively how well can the images be annotated semantically, based on the text extracted via OCR.



```
interface
    Seman/jcRe/a/jon
interface Re/a/jon
  interface Lex;calRe/a/jon
interface Word Sense '- .
    \
interface interface
    Verb Adjec/jve
. .
    interface
VertJFrame
interface Synset
interface WM'
ILanguageResource interface
    WordNet
    GateException
Word Net Exception
```

Readiris Pro 8

```
interface Relation
interface Seman ticR eta Hon
interface LexicaiRefation
interface VerhFrame
interface WordSense
S
interface Adjective
LanguageResource
interface WordNet
Gate Exception WordNetException
```

ABBYY FineReader 8.0

Figure 6.7: ReadIRIS and FineReader results on the WordNet UML diagram

6.2 Content Augmentation of the OCR Results

The screenshots from the software documentation, once processed by OCR, are annotated semantically in order to obtain a list of domain concepts for which each screen shot is relevant. For instance, the ANNIE diagram mentions concepts such as ANNIE, sentence splitter, etc. and therefore we would like to retrieve it as a search result if the user is interested in any of these domain terms.

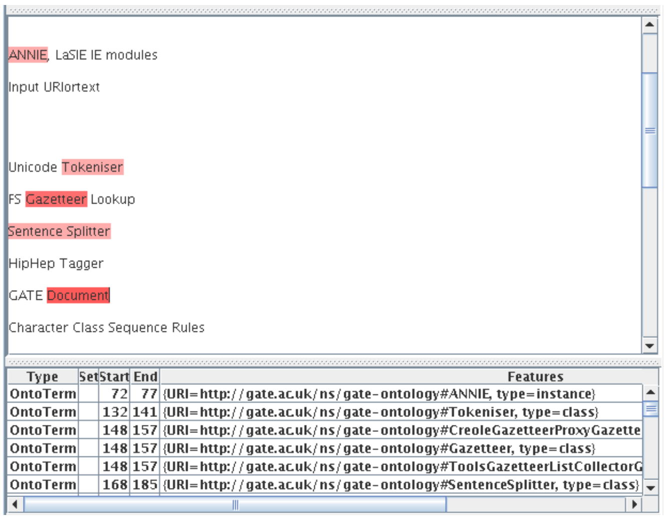
The OCR results are processed first with the KCIT tool, which identifies mentions of classes, instances, and properties using the GATE domain ontology[6], created in WP6.

As discussed in Section 4.1.3, the KCIT tool does not tackle ambiguities in the results. For instance the text ANNIE POS Tagger will be annotated as a mention of several ontology resources: the instance for this tagger, the POS tagger class, and, if applicable, any properties where the pos tagger instance appears as a range.

Resolving these ambiguities is done during the information consolidation phase, which keeps only the longest matching terms and also excludes annotations which refer to property values in the ontology, as we plan to have a separate relation annotation phase, to be developed in the subsequent deliverable D3.2.

The final result is a list of mentions of terms and classes from the ontology, which are attached as metadata to the screenshot image file and this metadata can then be indexed for semantic search in the heterogeneous knowledge store.

[6]http://gate.ac.uk/ns/gate-ontology



ANNIE, LaSIE IE modules

Input URIortext

Unicode Tokeniser

FS Gazetteer Lookup

Sentence Splitter

HipHep Tagger

GATE Document

Character Class Sequence Rules

Type	Set	Start	End	Features
OntoTerm		72	77	{URI=http://gate.ac.uk/ns/gate-ontology#ANNIE, type=instance}
OntoTerm		132	141	{URI=http://gate.ac.uk/ns/gate-ontology#Tokeniser, type=class}
OntoTerm		148	157	{URI=http://gate.ac.uk/ns/gate-ontology#CreoleGazetteerProxyGazette
OntoTerm		148	157	{URI=http://gate.ac.uk/ns/gate-ontology#Gazetteer, type=class}
OntoTerm		148	157	{URI=http://gate.ac.uk/ns/gate-ontology#ToolsGazetteerListCollectorG
OntoTerm		168	185	{URI=http://gate.ac.uk/ns/gate-ontology#SentenceSplitter, type=class}

Figure 6.8: Highlighted results of semantic annotation against the GATE domain ontology

Figure 6.8 shows the semantic annotation results on the OCR transcript of the ANNIE workflow diagram. The list of relevant ontology resources is shown underneath in a table, but it can also be exported as stand-off XML metadata.

6.3 Discussion and Future Work

This chapter discussed experiments on annotating semantically software screen shots, after first pre-processing them with OCR tools. Overall, the results can be of good enough quality to allow us to enrich automatically the images with metadata on the relevant domain concepts and instances from the ontology. The main caveat is in the necessity of using commercial OCR tools, although promising open-source alternatives are in the process of being developed.

As part of this work we have collected a corpus of GATE screen shots, which we plan to annotate manually with relevant ontology resources, in order to enable us to carry out quantitative evaluation. The results will be reported in the evaluation deliverable D3.3.

Another type of data frequently accompanying software artefacts are tutorial movies and lectures. In order to analyse those automatically, we planned originally to process them with a speech analyser, in order to obtain a text transcription. As part of this work, we evaluated the suitability of two freely available ASR tools - HTK[7] and Sphinx-3[8]. Neither toolkit is an off-the-shelf solution. They both require the creation of a domain-dependent language model. The acoustic models that are part of each package are based

[7]http://htk.eng.cam.ac.uk/

[8]http://cmusphinx.sourceforge.net/html/cmusphinx.php

on the HUB-4 broadcast news data [Ste97], collected under relatively similar acoustic conditions than our lecture data. We are hopeful that the acoustic models will be adequate as they are, although they could be augmented with training data specifically from lecture data (e.g., the TED corpus [LSFM94]). We anticipate the need to develop a general purpose language model for our domain, based on orthographic transcriptions from Switchboard data[9] and augmented with domain-specific data (see below). In addtion, we will have to segment and transcribe development and test sets from our data to evaluate the technology in a realistic way.

Research in Automatic Speech Recognition (ASR) has examined the problem of transcribing unrestricted audio from broadcast news sources for many years[10] Recently researchers have begun looking at the use of ASR technology for transcribing lecture data [GHHW04, PHG05]. With both types of speech corpora, transcription is not the end result, but rather a means to process the data further for the purposes of indexing and retrieval [RAKR00, MKL$^+$00] and topic segmentation [MPBG07].

The main caveat in performing all the above ASR customisation tasks is that they require significant expertise and experience with speech recognition, which is not likely to be affordable easily in industrial settings. Consequently, we feel that exploiting ASR for content augmentation in unrestricted, practical applications might be still a medium- to long-term research goal.

On a positive note, we feel that GATE technology will give us a significant advantage in performing the required tasks. To aid in this effort, we propose using technical documentation, academic papers, and text data contained on lecture slides to create a recognition vocabulary and language model tailored to our domain. Our audio data has been made available to ASR researchers (e.g., Thomas Hain at Sheffield) and we have initiated a discussion with members of the HTK group at the University of Cambridge on processing and using our data.

[9]See http://www.ldc.upenn.edu/ for a description of the various Switchboard corpora

[10]See http://www.nist.gov/speech/tests/bnr/bnews_99/bnews_99.htm for an overview of the technology developed for this task.

Chapter 7

Conclusion

This deliverable presented a number of content augmentation tools developed during the first year of activities within this workpackage. The work focused on semantic annotation of legacy software artefacts, with respect to a given domain ontology. In the case of non-textual content, e.g., screen shots and design diagrams, we have applied OCR software, prior to Information Extraction. The results have been made available as a web service, which is in the process of being refined and integrated within the TAO Suite.

The work planned for the remaining eighteen months will complement the automatic approach developed in this deliverable. More specifically, the forthcoming deliverable D3.2 will focus on tools for post-editing and manual correction of augmented content, but it will also include automatic tools for merging information from different content.

The final goal is semantic indexing and search, which will be addressed in another forthcoming deliverable (D3.4), where a set of user tools will be developed to enable user-friendly semantic search and browse of the augmented content. The tools will show the ontology and the user will be able to construct queries in an intuitive manner. These tools will be the front-end to the heterogeneous knowledge stores, where the semantically augmented content and semantics will be stored.

In order to measure the performance of our automatic tools, we will carry out a number of task-based evaluations, where we will compare the automatic results against human-annotated gold standards. Further improvements in the automatic tools and their algorithms is expected as a result. These activities will be reported in deliverable D3.3.

Bibliography

[AKM+03] H. Alani, S. Kim, D.E. Millard, M.J. Weal, W. Hall, P.H. Lewis, and N. Shadbolt. Web-based Knowledge Extraction and Consolidation for Automatic Ontology Instantiation. In *Proceedings of the Knowledge Markup and Semantic Annotation Workshop (SEMANNOT'03)*, Sanibel, Florida, 2003.

[Ama06] F. Amardeilh. OntoPop or how to annotate documents and populate ontologies from texts. In *Proceedings of the Workshop on Mastering the Gap: From Information Extraction to Semantic Representation (ESWC'06)*, Budva, Montenegro, 2006.

[ASHK06] A. Ankolekar, K. Sycara, J. Herbsleb, and R. Kraut. Supporting Online Problem Solving Communities with the Semantic Web. In *Proc. of WWW*, 2006.

[Cha01] Pierre-Antoine Champin. Rdf tutorial. http://www710.univ-lyon1.fr/ champin/rdf-tutorial, April 2001.

[CMB+05] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, C. Ursu, M. Dimitrov, M. Dowman, N. Aswani, and I. Roberts. *Developing Language Processing Components with GATE Version 3 (a User Guide).* `http://gate.ac.uk/`, 2005.

[CMB+06] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, C. Ursu, M. Dimitrov, M. Dowman, N. Aswani, and I. Roberts. *Developing Language Processing Components with GATE Version 3.1 (a User Guide).* `http://gate.ac.uk/`, 2006.

[CP82] K. Church and R. Patil. Coping with syntactic ambiguity or how to put the block in the box. *American Journal of Computational Linguistics*, 8(3-4), 1982.

[CV05] P. Cimiano and J. Voelker. Text2Onto - A Framework for Ontology Learning and Data-driven Change Discovery. In *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB)*, Alicante, Spain, 2005.

[DBCM05] M. Dimitrov, K. Bontcheva, H. Cunningham, and D. Maynard. A Lightweight Approach to Coreference Resolution for Named Entities in Text. In A. Branco, T. McEnery, and R. Mitkov, editors, *Anaphora Processing: Linguistic, Cognitive and Computational Modelling*. John Benjamins, 2005.

[DDM04] J. Domingue, M. Dzbor, and E. Motta. Magpie: Supporting Browsing and Navigation on the Semantic Web. In N. Nunes and C. Rich, editors, *Proceedings ACM Conference on Intelligent User Interfaces (IUI)*, pages 191–197, 2004.

[DTCP05] M. Dowman, V. Tablan, H. Cunningham, and B. Popov. Web-assisted annotation, semantic indexing and search of television and radio news. In *Proceedings of the 14th International World Wide Web Conference*, Chiba, Japan, 2005. `http://gate.ac.uk/sale/www05/web-assisted-annotation.pdf`.

[Fel98] Christiane Fellbaum, editor. *WordNet - An Electronic Lexical Database*. MIT Press, 1998.

[GHHW04] J. Glass, T. Hazen, L. Hetherington, and C. Wang. Analysis and processing of lecture audio data: Preliminary investigations. In *Proc., HLT-NAACL 2004 Workshop on Interdisciplinary Approaches to Speech Indexing and Retrieval*, Boston, 2004.

[GtKAvH07] Risto Gligorov, Warner ten Kate, Zharko Aleksovski, and Frank van Harmelen. Using google distance to weight approximate ontology matches. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 767–776, New York, NY, USA, 2007. ACM Press.

[HSC02] S. Handschuh, S. Staab, and F. Ciravegna. S-CREAM — Semi-automatic CREAtion of Metadata. In *13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02)*, pages 358–372, Siguenza, Spain, 2002.

[KPO$^+$04] A. Kiryakov, B. Popov, D. Ognyanoff, D. Manov, A. Kirilov, and M. Goranov. Semantic annotation, indexing and retrieval. *Journal of Web Semantics, ISWC 2003 Special Issue*, 1(2):671–680, 2004.

[KPS$^+$05] A. Kalyanpur, B. Parsia, E. Sirin, B. Cuenca-Grau, and J. Hendler. Swoop: A 'Web' Ontology Editing Browser. *Journal of Web Semantics*, 4(2), 2005.

[LBC07] Y. Li, K. Bontcheva, and H. Cunningham. Hierarchical, Perceptron-like Learning for Ontology Based Information Extraction. In *16th International World Wide Web Conference (WWW2007)*, pages 777–786, 2007.

[LSFM94] L. Lamel, F. Schiel, A. Fourcin, and J. Mariani. The translingual english database (ted)'. In *Proc., ICSLP*, Yokohama, 1994.

[Mit98] R. Mitkov. Robust Anaphora Resolution with Limited Knowledge. In *Proceedings of COLING'98/ACL'98*, 1998.

[MKL$^+$00] J. Makhoul, F. Kubala, T. Leek, D. Liu, and L. Nguyen. Speech and language technologies for audio indexing and retrieval. *Proc., IEEE*, 88(8), 2000.

[MPBG07] I. Malioutov, A. Park, R. Barzilay, and J. Glass. Making sense of sound: Unsupervised topic segmentation over acoustic input. In *Proceedings, ACL*, 2007.

[PHG05] A. Park, T. Hazen, and J. Glass. Automatic processing of audio lectures for information retrieval: Vocabulary selection and language modeling. In *Proc., ICASSP 05*, Philadelphia, 2005.

[RAKR00] S. Renals, D. Abberley, D. Kirby, and T. Robinson. Indexing and retrieval of broadcast news. *Speech Communication*, 32(1-2), 2000.

[Ste97] R. Stern. Specification of the 1996 Hub 4 Broadcast News Evaluation. In *Proc. 1997 DARPA Speech Recognition Workshop*, Chantilly, Virginia, 1997.

[VHA06] Antti Vehvilinen, Eero Hyvnen, and Olli Alm. A semi-automatic semantic annotation and authoring tool for a library help desk service. In *Proceedings of the first Semantic Authoring and Annotation Workshop*, November 2006.

[WBGH07] Christian Wartena, Rogier Brussee, Luit Gazendam, and Willem-Olaf Huijsen. Apolda: A practical tool for semantic annotation. In *In Proceedings of 18th International Workshop on Database and Expert Systems Applications*, pages 288–292, The Netherlands, September 2007.

[WZR07] Ren Witte, Yonggang Zhang, and Juergen Rilling. Empowering software maintainers with semantic web technologies. In *Proceedings of 4th European Semantic Web Conference*, June 2007.

Appendix A

Accessing and Modifying Ontologies for Content Augmentation

A.1 OWLIM Ontology Access

Ontology-access is based on OWL In Memory (OWLIM)[1], a high performance semantic repository developed at Ontotext. OWLIM is packaged as a Storage and Inference Layer (SAIL) for the Sesame RDF database. OWLIM uses the TRREE engine to perform RDFS, OWL DLP, and OWL Horst reasoning. The most expressive language supported is a combination of limited OWL Lite and unconstrained RDFS. OWLIM offers configurable reasoning support and performance. In the "standard" version of OWLIM (referred to as SwiftOWLIM) reasoning and query evaluation are performed in-memory, while a reliable persistence strategy assures data preservation, consistency and integrity.

OWLIM asks users to provide an XML configuration for the ontology they wish to load into the Sesame RDF database. In order to understand OWL statements, an ontology describing relations between the OWL constructs and the rdfs schema is imported. For example, owl:class is a subclass of the rdfs:class. This allows users to load OWL data into the sesame RDF database.

To load an ontology in an OWLIM repository, the user has to provide certain configuration parameters. These include the name of the repository, the URL of the ontology, the default name space, the format of the ontology (RDF/XML, N3, NTriples and Turtle), the URLs or absolute locations of the other ontologies to be imported, their respective name spaces and so on. Ontology files, based on their format, are parsed and persisted in the NTriples format.

In order to utilize the power of OWLIM, we have provided a fine-grained, service-based access to ontologies stored in OWLIM. Its basic purpose is to hide all the complexities of OWLIM and Sesame and provide an easy to use API for ontology access, as

[1]http://www.ontotext.com/owlim/

required for content augmentation.

The WSDL definition of this service is included at the end of this chapter.

A.2 Ontology Editor for Content Augmentation

As already discussed earlier, many typical content augmentation scenarios require the user to access visually the content of the ontology, add new instances and properties, and even classes, and to do all of this as part of the semantic annotation process. In other words, what is required is a seamless switch between documents and ontology editing.

In order to support this requirement, we developed also a simple ontology browsing and editing component, which we plan to integrate in the user validation interface in D3.2 and the TAO Suite.

The visual component can be used to navigate an ontology and quickly inspect the information relating to any of the objects defined in it — classes and restrictions, instances and their properties. Also, resources can be deleted and new resources can be added.

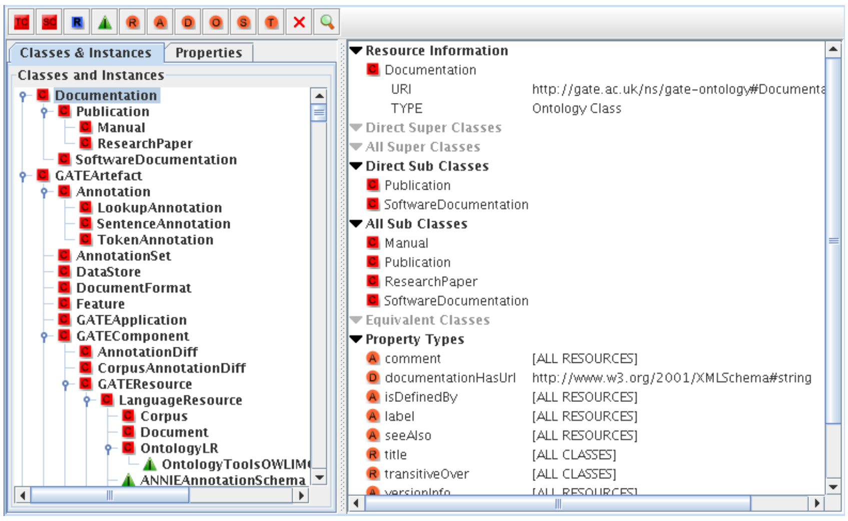
The component is developed in Java, as a Swing JPanel, and consequently, can be embedded easily in Java user interfaces. The next steps, in subsequent deliverables, are integration with other content augmentation tools and the TAO Suite.

The rationale behind developing a specialised light-weight visual component, rather than reusing an existing ontology editor such as Protege is as follows:

- Ontology editors such as Protege are developed to support all aspects of ontology editing, which typically involves many tabs, views, and windows. In contrast, what we need here is a basic ontology visualisation and editing component, which is then integrated within the content augmentation environment.
- Apart from needing simpler user interface, the content augmentation scenario also requires flexibility in the ontology component. For instance, some scenarios require users to be able to add new concepts and define new properties, whereas other scenarios only require addition of new instances, or no ontology editing at all. Therefore, a custom-made component can be made configurable so that different functions and elements of the user interface can be hidden/disabled easily.

As can be seen in Figure A.1, the ontology interface is divided into two areas. One on the left shows separate tabs for hierarchy of classes, instances, and properties. The view on right hand side shows the details of the object currently selected in the other two.

The first tab in the left view displays a tree which shows all the classes and restrictions defined in the ontology. The tree can have several root nodes—one for each top class in the ontology. The same tree also shows each class's instances. Instances that belong to several classes are shown as children of all the classes they belong to.



```
Classes and Instances
Documentation
  Publication
    Manual
    ResearchPaper
  SoftwareDocumentation
GATEArtefact
  Annotation
    LookupAnnotation
    SentenceAnnotation
    TokenAnnotation
  AnnotationSet
  DataStore
  DocumentFormat
  Feature
  GATEApplication
  GATEComponent
    AnnotationDiff
    CorpusAnnotationDiff
    GATEResource
      LanguageResource
        Corpus
        Document
        OntologyLR
          OntologyToolsOWLIM
        ANNIEAnnotationSchema
```

```
Resource Information
  Documentation
    URI                  http://gate.ac.uk/ns/gate-ontology#Documenta
    TYPE                 Ontology Class
Direct Super Classes
All Super Classes
Direct Sub Classes
  Publication
  SoftwareDocumentation
All Sub Classes
  Manual
  Publication
  ResearchPaper
  SoftwareDocumentation
Equivalent Classes
Property Types
  comment              [ALL RESOURCES]
  documentationHasUrl  http://www.w3.org/2001/XMLSchema#string
  isDefinedBy          [ALL RESOURCES]
  label                [ALL RESOURCES]
  seeAlso              [ALL RESOURCES]
  title                [ALL CLASSES]
  transitiveOver       [ALL CLASSES]
  versionInfo          [ALL RESOURCES]
```

Figure A.1: The ontology visualisation component

The second tab in the left view displays a tree of all the properties defined in the ontology. This tree can also have several root nodes—one for each top property in the ontology. The different kinds of properties are distinguished with different icons.

Whenever an item is selected in the tree view, the right-hand view is populated with the details that are appropriate for the selected object. For an ontology class, the details include the brief information about the resource such as the URI of the selected class, type of the selected class etc., set of direct superclasses, the set of all superclasses using the transitive closure, the set of direct subclasses, the set of all the subclasses, the set of equivalent classes, the set of applicable property types, the set of property values set on the selected class and the set of instances that belong to the selected class. For a restriction, in addition to the above information, it displays on which property the restriction is applicable to and the what type of the restriction it is.

For an instance, the details displayed include the brief information about the instance, set of direct types (the list of classes this instance is known to belong to), the set of all types this instance belongs to (through the transitive closure of the set of direct types), the set of same instances, the set of different instances and the values for all the properties that are set.

The information listed in the details pane is organised in sub-lists according to the type of the items. Each sub-list can be collapsed or expanded by clicking on the little triangular button next to the title. The ontology interface is dynamic and will update the information displayed whenever the underlying ontology is changed in OWLIM.

A toolbar at the top contains the following buttons to add and delete ontology resources:

- Add new top class (TC)
- Add new subclass (SC)
- Add new instance (I)
- Add new restriction (R)
- Add new RDF property (R)
- Add new Annotation property (A)
- Add new Datatype property (D)
- Add new Object property (O)
- Add new Symmetric property (S)
- Add new Transitive property (T)
- Remove the selected resource(s) (X)

The tree components allow the user to select more than one node, but the details table on the right-hand side of the GUI only shows the details of the first selected node. The buttons in the toolbar are enabled and disabled based on users' selection of nodes in the tree and also on whether this functionality is enabled in the current configuration of the interface. For instance, it is possible to only allow addition of new instances of existing classes, but not make changes to the schema itself, i.e., add new classes or define new properties.

1. **Creating a new top class:**

   A window appears which asks the user to provide details for its namespace (default name space if specified), and class name. If there is already a class with same name in ontology, the GUI shows an appropriate message.

2. **Creating a new subclass:**

   A class can have multiple super classes. Therefore, selecting multiple classes in the ontology tree and then clicking on the "SC" button, automatically considers the selected classes as the super classes. The user is then asked for details for its namespace and class name.

3. **Creating a new instance:**

   An instance can belong to more than one class. Therefore, selecting multiple classes in the ontology tree and then clicking on the "I" button, automatically considers the selected classes as the type of new instance. The user is then prompted to provide details such as namespace and instance name.

4. **Creating a new restriction:**

   As described above, restriction is a type of an anonymous class and is specified on a property with a constraint set on either the number of values it can take or the type of value allowed for instances to have for that property. User can click on the blue "R" square button which shows a window for creating a new restriction. User can select a type of restriction, property and a value constraint for the same. Please note that restrictions are considered as anonymous classes and therefore user does not have to specify any URI for the same but restrictions are named automatically by the system.

5. **Creating a new property:**

   An RDF property can have any ontology resource as its domain and range, so selecting multiple resources and then clicking on the new RDF Property icons shows a window where the selected resources in the tree are already taken as domain for the property. The user is then asked to provide information such as the namespace and the property name. Two buttons are provided to select resources for domain and range—clicking on them brings up a window with drop down box containing a list of resources that can be selected for domain or range and a list of resources selected by the user.

   - Since an annotation property cannot have any domain or range constraints, clicking on the new annotation property button brings up a dialog that asks the user for information such as the namespace and the annotation property name.
   - A datatype property can have one or more ontology classes as its domain and one of the pre-defined datatypes as its range, so selecting one or more classes and then clicking on the new Datatype property icon, brings up a window where the selected classes in the tree are already taken as the domain. The user is then asked to provide information such as the namespace and the property name. A drop down box allows users to select one of the data types from the list.
   - Object, symmetric and transitive properties can have one or more classes as their domain and range. For a symmetric property the domain and range are the same. Clicking on any of these options brings up a window where user is asked to provide information such as the namespace and the property name. The user is also given two buttons to select one or more classes as values for domain and range.

6. **Removing the selected resources:**

   All the selected nodes are removed when user clicks on the "X" button. Please note that since ontology resources are related in various ways, deleting a resource can affect other resources in the ontology; for example, deleting a resource can cause other resources in the same ontology to be deleted too.

7. **Setting properties on instances/classes:**

   Right-clicking on an instance brings up a menu that provides a list of properties that are inherited and applicable to its classes. Selecting a specific property from the menu allows the user to provide a value for that property. For example, if the property is an Object property, a new window appears which allows the user to select one or more instances which are compatible to the range of the selected property. The selected instances are then set as property values. For classes, all the properties (e.g. annotation and RDF properties) are listed on the menu.

8. **Setting relations among resources:**

   Two or more classes, or two or more properties, can be set as equivalent; similarly two or more instances can be markes as the same. Right-clicking on a resource brings up a menu with an appropriate option (Equivalent Class for ontology classes, Same As Instance for instances and Equivalent Property for properties) which when clicked then brings up a window with a drop down box containing a list of resources that the user can select to specify them as equivalent or the same.

A.3 Operations of the Low-Level Ontology Access Service

The web service has a large number of methods, therefore here we have listed only the operation names, not the WSDL in XML, for the sake of space. In a nutshell, there are methods for accessing a repository, obtaining classes, instances, and properties, and also modifying them (add/delete).

For integration purposes with HKS and the TAO Suite WP4 are now developing a more, high-level service, which works at the ontology level, not at such low-level, operational level.

```
<wsdl:operation name="getDefaultNameSpace">

<wsdl:operation name="addOntologyData">

<wsdl:operation name="isImplicitResource">

<wsdl:operation name="isSuperClassOf" >

<wsdl:operation name="isSubClassOf" >

<wsdl:operation name="getPropertyFromOntology" >

<wsdl:operation name="isEquivalentClassAs" >

<wsdl:operation name="addAnnotationProperty" >
```

```
<wsdl:operation name="getAnnotationProperties" >

<wsdl:operation name="getAnnotationProperties" >

<wsdl:operation name="getRDFProperties" >

<wsdl:operation name="getRDFProperties" >

<wsdl:operation name="getDatatypeProperties" >

<wsdl:operation name="getDatatypeProperties" >

<wsdl:operation name="getObjectProperties" >

<wsdl:operation name="getObjectProperties" >

<wsdl:operation name="getTransitiveProperties" >

<wsdl:operation name="getTransitiveProperties" >

<wsdl:operation name="getSymmetricProperties" >

<wsdl:operation name="getSymmetricProperties" >

<wsdl:operation name="isAnnotationProperty" >

<wsdl:operation name="addAnnotationPropertyValue" >

<wsdl:operation name="getAnnotationPropertyValues" >

<wsdl:operation name="getAnnotationPropertyValue" >

<wsdl:operation name="removeAnnotationPropertyValue" >

<wsdl:operation name="removeAnnotationPropertyValues" >

<wsdl:operation name="addRDFProperty" >

<wsdl:operation name="isRDFProperty" >

<wsdl:operation name="addDataTypeProperty" >

<wsdl:operation name="getDatatype" >

<wsdl:operation name="addSymmetricProperty" >

<wsdl:operation name="isEquivalentPropertyAs" >

<wsdl:operation name="getSuperProperties" >

<wsdl:operation name="getSuperProperties" >
```

```
<wsdl:operation name="getSubProperties" >

<wsdl:operation name="getSubProperties" >

<wsdl:operation name="isSuperPropertyOf" >

<wsdl:operation name="isSubPropertyOf" >

<wsdl:operation name="hasIndividual" >

<wsdl:operation name="hasIndividual" >

<wsdl:operation name="isDifferentIndividualFrom" >

<wsdl:operation name="isSameIndividualAs" >

<wsdl:operation name="addRDFPropertyValue" >

<wsdl:operation name="removeRDFPropertyValue" >

<wsdl:operation name="getRDFPropertyValues" >

<wsdl:operation name="removeRDFPropertyValues" >

<wsdl:operation name="addDatatypePropertyValue" >

<wsdl:operation name="removeDatatypePropertyValue" >

<wsdl:operation name="getDatatypePropertyValues" >

<wsdl:operation name="removeDatatypePropertyValues" >

<wsdl:operation name="addObjectPropertyValue" >

<wsdl:operation name="removeObjectPropertyValue" >

<wsdl:operation name="getObjectPropertyValues" >

<wsdl:operation name="removeObjectPropertyValues" >

<wsdl:operation name="login" >

<wsdl:operation name="logout">

<wsdl:operation name="getRepositoryList">

<wsdl:operation name="setCurrentRepositoryID">

<wsdl:operation name="getCurrentRepositoryID">

<wsdl:operation name="createRepository" >
```

```
<wsdl:operation name="createRepositoryFromUrl" >

<wsdl:operation name="removeRepository" >

<wsdl:operation name="cleanOntology">

<wsdl:operation name="getOntologyData" >

<wsdl:operation name="removeClass">

<wsdl:operation name="hasClass" >

<wsdl:operation name="isTopClass" >

<wsdl:operation name="addSubClass" >

<wsdl:operation name="addSuperClass" >

<wsdl:operation name="removeSubClass" >

<wsdl:operation name="removeSuperClass" >

<wsdl:operation name="getSubClasses" >

<wsdl:operation name="getSuperClasses" >

<wsdl:operation name="setDisjointClassWith" >

<wsdl:operation name="setEquivalentClassAs" >

<wsdl:operation name="getDisjointClasses" >

<wsdl:operation name="getEquivalentClasses" >

<wsdl:operation name="removePropertyFromOntology">

<wsdl:operation name="addObjectProperty" >

<wsdl:operation name="addTransitiveProperty" >

<wsdl:operation name="getRange" >

<wsdl:operation name="isFunctional" >

<wsdl:operation name="setFunctional">

<wsdl:operation name="isInverseFunctional" >

<wsdl:operation name="setInverseFunctional" >

<wsdl:operation name="isTransitiveProperty" >
```

```
<wsdl:operation name="isDatatypeProperty" >

<wsdl:operation name="isObjectProperty" >

<wsdl:operation name="setEquivalentPropertyAs" >

<wsdl:operation name="getEquivalentPropertyAs" >

<wsdl:operation name="addSuperProperty" >

<wsdl:operation name="removeSuperProperty" >

<wsdl:operation name="addSubProperty" >

<wsdl:operation name="removeSubProperty" >

<wsdl:operation name="getInverseProperties" >

<wsdl:operation name="setInverseOf" >

<wsdl:operation name="addIndividual" >

<wsdl:operation name="removeIndividual" >

<wsdl:operation name="getIndividuals" >

<wsdl:operation name="getIndividuals">

<wsdl:operation name="getClassesOfIndividual" >

<wsdl:operation name="setDifferentIndividualFrom" >

<wsdl:operation name="getDifferentIndividualFrom" >

<wsdl:operation name="setSameIndividualAs" >

<wsdl:operation name="getSameIndividualAs" >

<wsdl:operation name="getOnPropertyValue" >

<wsdl:operation name="setOnPropertyValue" >

<wsdl:operation name="getPropertyValue" >

<wsdl:operation name="setPropertyValue" >

<wsdl:operation name="getRestrictionValue" >

<wsdl:operation name="setRestrictionValue" >

<wsdl:operation name="getClassType" >
```

```
<wsdl:operation name="addStatement" >

<wsdl:operation name="removeStatement" >

<wsdl:operation name="addClass" >

<wsdl:operation name="getClasses" >

<wsdl:operation name="getVersion" >

<wsdl:operation name="setVersion" >

<wsdl:operation name="getDomain" >
```