D2.2.2 Report: Controlled Language IE Components version 2

Adam Funk, Brian Davis, Valentin Tablan, Kalina Bontcheva, Hamish Cunningham
(University of Sheffield)

Abstract.
EU-IST Integrated Project (IP) IST-2003-506826 SEKT
Deliverable D2.2.2 (WP2.2)

This deliverable outlines an approach to automatic generation of metadata from natural language. The emphasis is on easy to use applications and controlled language interfaces. In brief, Controlled Language Information Extraction (CLIE) uses natural language processing technology to enable users to create, modify and use knowledge stored in repositories like KAON and SESAME.

Keyword list: controlled language, information extraction, language processing

Document Id.	SEKT/2005/D2.2.2/v1.0
Project	SEKT EU-IST-2003-506826
Date	January 15, 2007
Distribution	public

SEKT 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-2003-506826.

British Telecommunications plc.
Orion 5/12, Adastral Park
Ipswich IP5 3RE
UK
Tel: +44 1473 609583, Fax: +44 1473 609832
Contact person: John Davies
E-mail: john.nj.davies@bt.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

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: Hamish Cunningham
E-mail: hamish@dcs.shef.ac.uk

Intelligent Software Components S.A.
Pedro de Valdivia, 10
28006 Madrid
Spain
Tel: +34 913 349 797, Fax: +49 34 913 349 799
Contact person: Richard Benjamins
E-mail: rbenjamins@isoco.com

Ontoprise GmbH
Amalienbadstr. 36
76227 Karlsruhe
Germany
Tel: +49 721 50980912, Fax: +49 721 50980911
Contact person: Hans-Peter Schnurr
E-mail: schnurr@ontoprise.de

Vrije Universiteit Amsterdam (VUA)
Department of Computer Sciences
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands
Tel: +31 20 444 7731, Fax: +31 84 221 4294
Contact person: Frank van Harmelen
E-mail: frank.van.harmelen@cs.vu.nl

Empolis GmbH
Europaallee 10
67657 Kaiserslautern
Germany
Tel: +49 631 303 5540, Fax: +49 631 303 5507
Contact person: Ralph Traphöner
E-mail: ralph.traphoener@empolis.com

University of Karlsruhe, Institute AIFB
Englerstr. 28
D-76128 Karlsruhe
Germany
Tel: +49 721 608 6592, Fax: +49 721 608 6580
Contact person: York Sure
E-mail: sure@aifb.uni-karlsruhe.de

University of Innsbruck
Institute of Computer Science
Technikerstraße 13
6020 Innsbruck
Austria
Tel: +43 512 507 6475, Fax: +43 512 507 9872
Contact person: Jos de Bruijn
E-mail: jos.de-bruijn@deri.ie

Kea-pro GmbH
Tal
6464 Springen
Switzerland
Tel: +41 41 879 00, Fax: 41 41 879 00 13
Contact person: Tom Bösser
E-mail: tb@keapro.net

Sirma AI EAD, Ontotext Lab
135 Tsarigradsko Shose
Sofia 1784
Bulgaria
Tel: +359 2 9768 303, Fax: +359 2 9768 311
Contact person: Atanas Kiryakov
E-mail: naso@sirma.bg

Universitat Autonoma de Barcelona
Edifici B, Campus de la UAB
08193 Bellaterra (Cerdanyola del Vallès)
Barcelona
Spain
Tel: +34 93 581 22 35, Fax: +34 93 581 29 88
Contact person: Pompeu Casanovas Romeu
E-mail: pompeu.casanovas@uab.es

Executive Summary

In this task we developed a controlled language and a software tool to allow end users to edit ontologies.

A controlled language is a subset of a natural language which is generally designed to be less ambiguous than the complete language and furthermore to include only certain vocabulary terms and grammar rules which are relevant for a specific task. Controlled languages have a long history of use. Since the 1970s, CLs have been used by large companies such as Caterpillar, Boeing, and Perkins for generation of multilingual documentation.

Although most machine translation systems use some sort of intermediate representation for the information contained in the sentences which are being translated, the use of CLs for knowledge management is a relatively new endeavour whose first applications appeared in the mid 1990s. [Sow02] shows that all languages have the power to express first order logic statements.

Natural language, however, is far more powerful than first order logic and contains lexical, syntactic, semantic and logical ambiguities. Constraining the language to avoid ambiguities yields a great improvement in parsing accuracy and efficiency and can result in a medium suitable for human use to produce formal conceptual data.

The Controlled Language Information Extraction (CLIE) software and the CLOnE (Controlled Language for Ontology Editing) language have been developed in SEKT to enable users to create, administer and use information stored in knowledge repositories like KAON and SESAME.

In 2006 (SEKT year 3) we made significant improvements in CLOnE and CLIE. The language now supports synonyms (implemented as RDF labels), two types of properties, multiple properties sharing the same domain and range (with different property descriptions), and delete- and undo-functionality. The CLIE engine also refers to the existing information in the ontology to interpret input sentences, allowing CLOnE to be more powerful without further complicating the syntax, so CLOnE is still easy to learn by following examples.

We also carried out a comparative evaluation of CLIE and Protégé with human subjects in order to measure usability for ontology editing; we obtained favourable results for CLIE as well as interesting and useful suggestions for improving it.

CLIE development will not end with SEKT; it will be used and further developed in Poleazy, Nepomuk and Líon, and potentially in NeOn, KnowledgeWeb and other projects.

Contents

Chapter 1

Introduction

1.1 Aims of CLIE

People need to organise, compartmentalise, and order their environments. The world of computing is rife with structured information. Object-oriented programs, databases and file-systems are common examples of structured data. Controlled Language Information Extraction (CLIE) is an application which will allow users to design, create, and manage information spaces without knowledge of complicated standards such as XML, RDF and OWL, or ontology engineering tools such as Protégé.[1] The CLIE architecture is divided into two independent parts: the language interface and the application.

The language interface builds on existing research from machine translation (MT) and applications of controlled natural languages for knowledge representation. The application interface is based on common lightweight technologies which are familiar to web users. The growing interest in Semantic Web applications and the need to port information into a machine readable format create many uses for this type of application.

Creating formal data is a high initial barrier for small organisations and individuals wishing to create ontologies and thus benefit from semantic knowledge technologies. Part of the solution comes from ontology authoring tools such as Protégé, but these often require specialist knowledge about ontology engineering. Therefore, in the case of naive users, the definition of a controlled language for formal data description will enable them to author ontologies directly in natural language. Building on controlled language MT work, information extraction (IE) for controlled language analysis may achieve the high levels of accuracy necessary to make the ontology authoring process more widely feasible.

The controlled language IE task has developed a simplified natural language processor that allows the specification of logical data for Semantic KT purposes in normal language, but at the same time attains the high accuracy levels necessary for high reliability in applications. The components are based on GATE's existing cascade of finite state transducers

[1] `http://protege.stanford.edu`

(FSTs) for IE. [CMBT02] CLIE is configured so that it either accepts input as valid (in which case accuracy is generally 100%) or rejects it and will warn the user of needed repairs to his syntax. Because the parsing process is deterministic, the usual IE performance measures (precision and recall) are not relevant.

1.2 Achievements in Year 3 of SEKT

In 2006 we made significant improvements in the CLOnE controlled language and the CLIE engine. (The following points are explained in more detail in Chapter 4.)

- Users can now add (and remove) synonyms for classes and instances, which are stored in the ontology as RDF labels for subsequent re-use. Synonyms can be used in subsequent CLOnE statements and are dereferenced to the appropriate classes and instances.
- CLOnE now supports two types of object properties:
  1. `Books have chapters.`
  2. `Agents are authors of documents.`
     `Agents are publishers of documents.`
     This construction allows multiple properties to have the same domain and range, distinguished by the "middle part" (e.g. *author* or *publisher*) of the property.

  (CLIE previously supported only the first type, and more than one property could not share the same domain and range.)
- All additions to the ontology can be undone or deleted by writing CLOnE sentences beginning with the keyword `Forget`.
- CLIE uses information already in the ontology in several ways as part of the interpretation of input sentences (as explained in detail in Section 4.3).
- Referring to the ontology allows us to "re-use" the same syntax in some cases with different semantics for classes and instances (see Sections 4.2 and 4.3) so that the language is more powerful but no less intuitive, and still easy to learn by examples (as 3.2 will show).

We also carried out a comparative usability evaluation of CLIE as Chapter 5 will describe in detail.

1.3 Structure of this document

Chapter 2 reviews related work in controlled languages and user-friendly ontology-editing interfaces, and compares them with our goals for CLIE. Chapter 3 is our user's guide to running CLIE and writing input in CLOnE, and Chapter 4 (supplemented by Appendix B) explains in detail how the CLIE software works.

To demonstrate the usability of our work, Chapter 5 and Appendix A present our user evaluation of CLIE in comparison with the well-known ontology-editing tool, Protégé. Finally, Chapter 6 sets out our plans for further development and use of CLIE outside the SEKT project.

Chapter 2

Related work

2.1 ACE

Attempto Controlled English[1] (ACE) is a well-known controlled natural language. It is a subset of standard English designed for knowledge representation and technical specifications, and constrained to be unambiguously machine-readable into discourse representation structures, a form of first-order logic. (It can also be translated into other formal languages.) ACE has been adopted as the controlled language for the EU FP6 Network of Excellence REWERSE[2] (Reasoning on the Web with Rules and Semantics). [FKK+06]

The Attempto Parsing Engine (APE) consists principally of a definite clause grammar, augmented with features and inheritance and written in Prolog. [Hoe04] This tool can be tested and demonstrated with a web browser through the *APE Webinterface*[3] and clients for the APE web service are also available.[4]

Figure 2.1 presents APE's output for the sentence "Every book has a title.", as generated through the web interface. The resulting DRS (Discourse Representation Structure) can be mapped cleanly to first-order logic and OWL DL[5], and REWERSE proposes ACE OWL, a sublanguage of ACE, as a means of writing formal, simultaneously human- and machine-readable summaries of scientific papers. [KF06, Kuh06]

There are a few difficulties with ACE OWL. In order to be highly expressive but at the same time strictly unambiguous, ACE itself prohibits certain very natural constructions such as the use of *only* as an adverb; consequently defining *carnivore* as an animal that eats only meat requires either two affirmative sentences ("Every carnivore eats nothing but meat." and "Everything that eats nothing but meat is a carnivore.") or a logically

[1] `http://www.ifi.unizh.ch/attempto/`
[2] `http://rewerse.net/`
[3] `http://www.ifi.unizh.ch/attempto/tools/`
[4] `http://www.ifi.unizh.ch/attempto/documentation/ape_webservice.html`
[5] Web Ontology Language Description Logic

Input:

```
Every book has a title.
```

or

```
Each book has a title.
```

Paraphrase:

```
if there is a book A then the book A has a title B .
```

DRS:

```
[]
   [A]
   object(A, atomic, book, object, cardinality,
          count_unit, eq, 1)-1
   =>
   [B, C]
   object(B, atomic, title, object, cardinality,
          count_unit, eq, 1)-1
   predicate(C, unspecified, have, A, B)-1
```

Figure 2.1: Example of APE input and output

equivalent but less human-readable sentence (such as "No carnivore eats something that is not a meat."). [Kal06a] Since ACE OWL also aims to provide reversibility (translating OWL DL into ACE), OWL's *allValuesFrom* must be translated into a similar construction which can be rather difficult for humans to read. Currently, ACE OWL does not currently support enumerations (OWL's *oneOf*) and has limited support for datatype properties. [Kal06b]

ACE OWL imposes several further restrictions on ACE, such as the elimination of plural nouns. (Although APE generates identical output for "Every book has a title." and "Each book has a title.", it rejects the input "All books have titles." with a general error message indicating that it cannot be analysed.) [Kal06b]

Although ACE itself has a predefined lexicon, unknown words can be used if they are annotated with a POS[6] tag, e.g. "Every carnivore is a n:meat-eater", but this does require the user to be familiar with the lexicon. [Kal06b]

In summary, ACE OWL allows users to express in a sublanguage of English much of the functionality of OWL DL, but it requires training and practice to write correctly. It is, however, easy to read—with the exception of some constructions generated from OWL DL (as mentioned above).

[6]POS: *part of speech*, e.g. noun, verb, adjective, determiner.

2.2 Aqualog

AquaLog[7] is an ontology-driven, portable question-answering (QA) system built for the goal of providing a natural language query interface to semantic mark-up stored in knowledge base. The application scenario for AquaLog and other systems reviewed in this section is very much similar to a natural language interface to relational database, but with semi-structured data encoded in ontologies instead of structured data stored in RDBMS. [LM04]

By using tools from the GATE framework [CMBT02], AquaLog translates controlled natural language queries into a triple representation called Query-Triples. GATE provides a set of linguistic annotations and additional JAPE (Java Annotation Pattern Engine) grammars are created in order to capture relations and question indicators and to perform partial or shallow parsing across the user's questions. As with typical finite-state parsing or shallow parsing, JAPE consists of a cascade of finite state automata, which in combination can provide sufficient parsing information in order to process the questions efficiently. Not all language processing tasks require deep context-free-based parsing. The authors [LM04] comment in relation to [KL03] on the discussion of the costs and benefits of full parsing and furthermore on the time-consuming efforts of engineering the grammar not to mention how often a substantial amount of parse information would be discarded. This is also the case for Aqualog, whereby the system is implemented using a simple triple-based intermediate representation as opposed to DRS (discourse representation structure) or fully formed FOPL (first-order predicate logic). [BKGK05]

Further processing is conducted by the Relation Similarity Service (RSS) module. The role of the RSS module is to map CL questions into ontology compliant queries. Furthermore this module invokes the use of string similarity metrics, lexical resources such as WordNet [Fel98] and domain-dependent lexicons in order to generate query-triples that are compliant with the underlying ontology. [LM04] The RSS module plays a crucial role within the system. It structurally checks the generated query triples against the underlying ontology. If the RSS fails to discover matching relations or concepts within the KB, it requests, as last resort, the user to disambiguate the relation or concept from a given set of candidates.

An initial study of Aqualog against a KB containing the AKT ontology and the KMi knowledge base, with 10 users having no knowledge about Aqualog, when given unrestricted questions, demonstrates that AquaLog can handle 48.68% of the questions. The advantages of AquaLog include: its wide coverage over the types of questions that can be asked, the combination of WordNet and users' feedback to tackle the disambiguation problem while mapping users' queries to concepts and relations in the ontology, and portability that allows minimal configuration change while using different ontology as knowledge source. Though Aqualog can handle unrestricted questions, from a linguistic perspective the syntactical constructs themselves are controlled and thus the input lan-

[7] http://kmi.open.ac.uk/technologies/aqualog/

guage can be seen as a controlled language.

Theoretically, in comparison to the ACE Question Answering System [BKGK05] for instance, the linguistic components are more robust and maintainable, due to the shallow NLP methods employed by the system. However, existing linguistic rules pose difficulties with respect to complex queries, requiring extension of the NLP component. Aqualog's authors plan to remedy this in their future work. An additional key limitation of AquaLog is that only a single ontology can be used at a time. This is to be addressed in PowerAqua [LMU06], which aims to find answers from distributed, ontology-based semantic markup. This allows PowerAqua to find answers that require "composing heterogeneous information derived from multiple information sources that are autonomously created and maintained." [LMU06]. At the time of writing, PowerAqua had not been completely implemented for any evaluation.

2.3 Cypher

Cypher[8] is proprietary (but free of charge) software from Monrai Technologies that translates natural language input into RDF and SeRQL (Sesame RDF Query Language) according to grammars and lexica defined by the user in XML. It is designed particularly to parse noun phrases and descriptive clauses, in order to help users build a natural language interface to the Semantic Web.

As Cypher is recently developed proprietary software and is not the subject of any research papers, it is difficult to examine here. We may note, however, that it is designed to analyse a user-defined subset of natural language (not necessarily a controlled language in the technical sense) and that it requires the development of a grammar and lexicon for each application domain.

2.4 GINO

GINO (Guided Input Natural Language Ontology Editor) aims to use guided data entry to overcome two problems of natural language input (NLI) systems: the *adaptivity barrier*, caused by the varying requirements of different domains; and the *habitability problem*, the disparity between what users expect and what the system is capable of doing. [BK06]

The GINO editor provides a guided, controlled NLI for domain-independent ontology editing for the Semantic Web. As the user types a sentence, GINO incrementally parses the input not only to warn the user as soon as possible about errors but also to offer the user (through the GUI) suggested completions of words and sentences—similarly to the "code assist" feature of Eclipse[9] and other development environments. GINO translates

[8] `http://www.monrai.com/products/cypher/cypher_manual`
[9] `http://www.eclipse.org/`

the completed sentence into triples (for ontology editing) or SPARQL[10] statements (for queries) and passes them to the Jena Semantic Web framework. (The JENA Eyeball[11] model checker verifies the OWL for consistency.) Because GINO has been developed as an extension to the GINSENG [Kau06] NLI querying system it also support queries in a similar way.

Unlike many NLIs (including to some extent ACE), GINO does not rely on a lexicon fixed in advance; it flexibly uses the vocabulary inherent in the grammar (specified in a form similar to Backus-Naur) along with the terms found in the loaded ontologies. GINO also offers the option of graphically editing the ontology beside the NLI.

Although the guided interface facilitates input, the sentences are quite verbose. For example to create a class *lake*, a numeric property *lakeDepth* and an instance, the user must enter the following.

- `there is a class called lake.`
- `there is a new property named lake depth.`
- `there is an instance of class lake named tahoe.`

2.5 Simplified English

Simplified English is a well-established controlled language originally developed by the European Association of Aerospace Manufacturers in the 1980s for writing technical documentation, especially aircraft maintenance manuals. It is now maintained by the AeroSpace and Defence Industries Association of Europe (ASD) Simplified Technical English Maintenance Group (STEMG) and sold as a proprietary specification[12] although an obsolete document is available on the web.[13]

Simplified English is designed to produce unambiguous documents that can be easily read (especially by non-native speakers of English) and translated (even semi-automatically), particularly for safety-critical, human-readable documents.[14] It is based on a standard lexicon of syntactically and semantically unambiguous words: for example, *close* functions as a verb only, and *follow* means only *come after*, not *obey*. The standard lexicon includes a large number of verbs and nouns relevant to the intended domain and allows the addition of "Technical names" such as proper nouns, measurements, dial markings, etc.

[10] `http://www.w3.org/TR/rdf-sparql-query/`
[11] `http://jena.sourceforge.net/Eyeball/full.html`
[12] `http://www.simplifiedenglish-aecma.org/Simplified_English.htm`
[13] `http://www.userlab.com/Downloads/SE.pdf`
[14] `http://www.userlab.com/SE.html`

Although Simplified English is designed to facilitate the use of machine translation, it is not a formal language intended for knowledge representation. It does however represent a historically important and commercially significant step in the history of controlled natural language.

2.6 Discussion

CLIE and CLOnE, the software and input language that we present in this deliverable, are designed to offer the following advantages.

- CLIE requires only one programming language or runtime environment, Java 1.5.
- CLOnE is a sublanguage of English.
- As far as possible, CLOnE is grammatically lax; in particular it does not matter whether the input is singular or plural (or even in grammatical agreement). For example, the lines of input within each group in Figure 2.2 have the same semantics.
- CLOnE is case-insensitive.
- CLOnE can be compact; the user can create any number of classes or instances in one sentence.
- As Section 3.2 will show, CLOnE is easy to learn by following examples and a few guiding rules, without having to study formal expressions of syntax.
- Nonetheless, the current implementation of CLIE uses only 11 syntactic rules (presented in Section 4.2).
- Any valid sentence of CLOnE can be unambiguously parsed.

CLIE and CLOnE have been favourably evaluated by test users (as shown in Chapter 5) as part of the SEKT project, and will be developed and applied beyond the project (as indicated in Chapter 6).

```
Book is a type of document.
Books is a type of document.
Books are types of document.
Books are types of documents.
```

```
Alice is a person.  Bob is a person.
Alice and Bob are person.
Alice and Bob are persons.
```

Figure 2.2: Groups of equivalent sentences in CLOnE

Chapter 3

User's guide

3.1 Installing and running CLIE in GATE

1. Ensure you have Sun Java 1.5 and a recent development snapshot of GATE 4.0. (CLIE requires GATE 4.0, which requires Java 1.5.)
   `http://java.sun.com/javase/downloads/index.jsp`
   `http://gate.ac.uk/download/snapshots/`

2. Refer to the GATE documentation for instructions on how to run GATE on your platform.
   `http://gate.ac.uk/documentation.html`

3. Unpack the CLIE application (`clie.zip` or `clie.tar.gz`). This will create the `sins` and `sins/clone` directories, the application file `sins/clone/clie-demo.xgapp` and the additional files that make up the application.

4. Launch GATE. Click `File` on the menu bar and then `Restore Application from File`. Browse to `clie-demo.xgapp` and select it, as shown in Figure 3.1. This will create a corpus pipeline `CLIE`, a document `Text input` and an ontology `Ontology`, as well as the other components required by the application.

5. At this point the CLIE application is loaded and connected to an empty ontology and an empty input document. You can start with the empty ontology (CLIE will create a top class `Entity` when a top class is first required) or load an existing ontology using either of the following methods.

   - To load a file of ontological data, right-click on the `Ontology` and select `Load OWL-Lite data`, `Load OWL-DL data`, `Load OWL-Full data` or `Load RDF(S) data`. Then select the file. Two sample files of

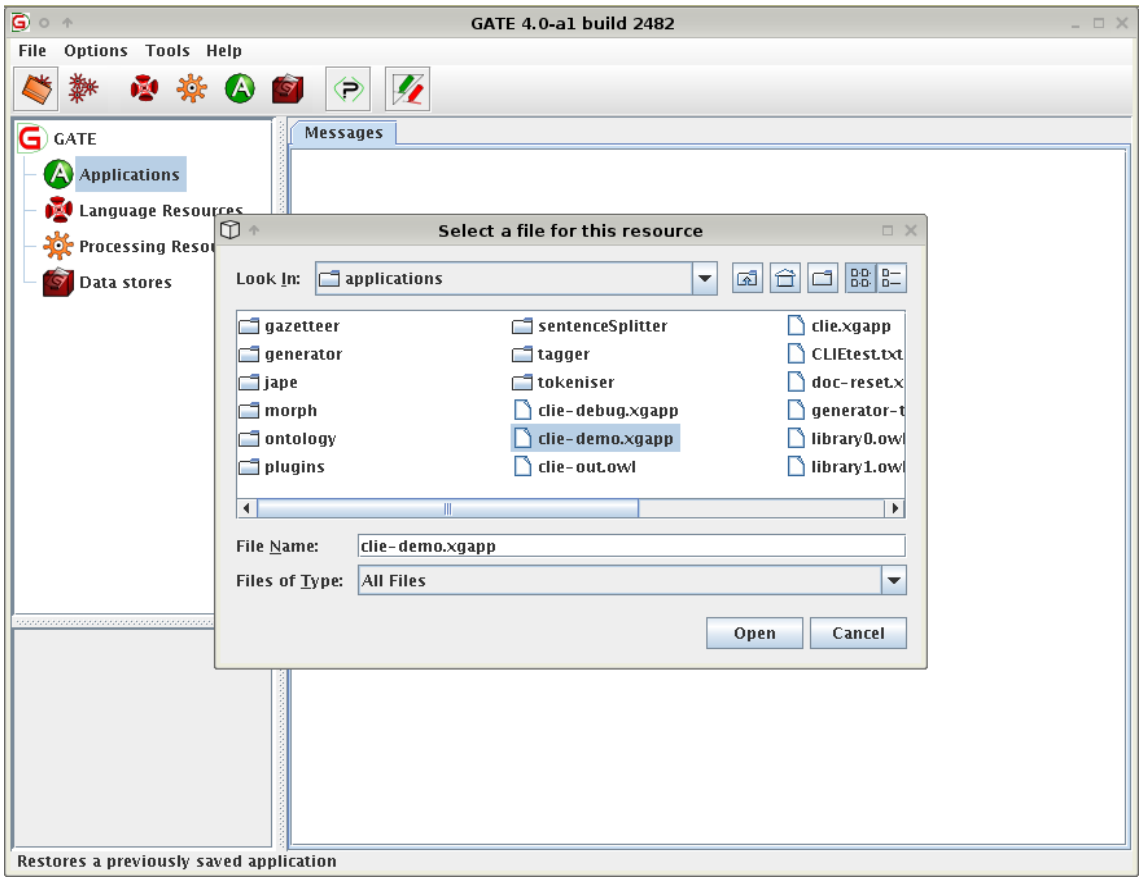

Figure 3.1: Loading the CLIE application

OWL-Lite data are provided with CLIE: `sins/clone/library0.owl` and `library1.owl`.

- To load an ontology from a URL, right-click on `Language Resources` and select either `KAON Ontology` or `Jena Ontology`, then enter a name of your choice and the source URL.[1] You must now configure CLIE to use this ontology instead of the default one. Double-click on `CLIE`, then select `CLIE Jape Transducer` (under `Selected Processing Resources`). Click the pull-down menu for the `ontology` parameter in the lower part of the window and select the new ontology you wish to modify with CLIE.

6. To modify the ontology with CLIE, double-click on the `Text input` GATE document and type sentences in the CLOnE language (described in the next section). To process the input, either right-click on `CLIE` and select `Run`, or click on the `Run` button in the `CLIE` pane. After running CLIE you can inspect the selected ontology by double-clicking on it and view the RDF output and error messages in the `Messages` pane.

   Figures A.2 and A.3 (page 47) illustrate the text input and ontology viewing areas in the GATE GUI.

   Once your input text has been correctly processed, you can delete it using your platform's usual "select all" keybinding and the delete or backspace key. If it has not been completely successful, you can delete the correct sentences, undo others by prefixing them with `Forget that`, and edit others to make them correct and parseable.

   If you leave correctly processed sentences in the input text and run them again, it will generate errors.

7. To save your modified ontology as an OWL-Lite file, right-click on the ontology, select `Save to file`, select a directory and enter a file name.

3.2 Writing CLOnE

CLOnE (Controlled Language for Ontology Editing) is designed to be easy to learn from examples, and consists of keywords (including punctuation) and names (of classes, instances and properties). In the following examples, keywords (all of which are listed in in Figure 3.2) are underlined.

Many CLOnE sentences can be negated by adding `Forget` or `Forget that` at the beginning. These negative sentences generally "undo" the effect of the basic form, but

[1] For example `http://proton.semanticweb.org/2005/04/protonkm` for the Proton Knowledge Management Module.

they do not always delete everything you might expect. Such differences are pointed out below.

In any place in CLOnE where we give a list of names (e.g. `projects and organizations`) you can also use a single name as well as a longer list of more than two names (provided there is a comma or the keyword `and` between each name).

Manipulating classes

1. `There are projects and organizations.`

   Create two new classes, *Project* and *Organization*, directly under the top class *Entity*.

2. `Universities and companies are types of organization.`

   The class *Organization* must already exist. Make classes *University* and *Company* direct subclasses of *Organization* and create the first two classes if they do not already exist. (A class can have more than one direct superclass.)

3. `Forget that universities and companies are types of organization.`

   Unlink the subclass-superclass relationships. This statement does not delete any classes. (See example 6 for deleting classes.)

Manipulating instances

4. `'University of Sheffield' is a university.`

   Create an instance *University_of_Sheffield* of the class *University* (which must already exist). Because the name contains a preposition (`of`) it needs to be enclosed in quotation marks—these tell the CLIE program to treat everything between them as one name. (See item 6 for deleting instances.)

5. `'Smith and Sons' and 'Jones Ltd.' are companies.`

   Create two instances *Smith_and_Sons* and *Jones_Ltd* of the class *Company* (which must already exist). These names are quoted because the first one contains a keyword (`and`) and the second one contains punctuation (`.`). (See item 23 below for a fuller explanation of quoting.)

Deleting classes and instances

6. `Forget 'Smith and Sons', Alice Smith and projects.`

and	have	text as
are	is	texts
are a type of	is a	texts as
are also called	is a type of	textual
are also known as	is also called	textual as
are called	is also known as	that can have
are known as	is an	that has
are types of	is called	that have
can have	is known as	there are
date	number	there is
date as	number as	which are
dates	numbers	which can have
dates as	numbers as	which has
delete all	numeric	which have
delete everything	numeric as	which is
forget	string	with value
forget all	string as	.
forget everything	strings	,
forget that	strings as	
has	text	

a, an, that, the, these and other determiners.
at, in, of and other prepositions.

Figure 3.2: Reserved words and phrases

Delete the instances *Smith_and_Sons* and *Alice_Smith* and the class *Project*. In this statement, the list can contain a mixture of classes and instances.

Manipulating object properties

7. `Persons are authors of deliverables.`

   If the classes *Person* and *Deliverable* exist, define a property *Person_Author_of_Deliverable* between them. You can provide a list of class names for both the domain and range, and CLIE will create the property for all the combinations.

8. `Forget that persons are authors of deliverables.`

   Delete the property defined in the last example.

9. `Alice Smith and Bob Davis are authors of 'D2.3.4'.`

   If *Alice_Smith* and *Bob_Davis* are instances of *Person* and *D2.3.4* is an instance of *Deliverable*, and the property already exists, create two property definitions to indicate that they are authors of it. `D2.3.4` must be quoted because it contains punctuation (`.`)—see item 23 below.

10. `Forget that Bob Davis is author of 'D2.3.4'.`

    Remove the property definition for one of the authors in the previous example. (This leaves the other author defined.)

11. `Journals and conferences have articles.`

    If classes *Journal*, *Conference* and *Article* exist, create properties *Journal_has_Article* and *Conference_has_Article*.

12. `'Journal of Knowledge Management' has 'Crossing the Chasm'.`

    If the named instances are members of classes between which a *has*-property already exists, instantiate the property with these instances. (See item 23 below for an explanation of the quotation marks.)

13. `Forget that conferences have articles.`

    Delete the property *Conference_has_Article* (created in example 11).

14. `Forget that 'Journal of Knowledge Management' has 'Crossing the Chasm'.`

    Delete the property definition instantiated in example 12.

Manipulating datatype properties

15. `Projects have string names.`

    Create a datatype property with domain *Project* (the class must already exist) and the name *Project_has_name*. The type can be specified with one of the following keywords: `text`, `textual`, or `string`; `date`; and `number` or `numeric`.

16. `Forget that projects have string names.`

    Delete the property from the previous example.

17. `SEKT has name with value 'Semantically Enhanced Knowledge Technologies'.`

    Instantiate a datatype property definition.

Adding and removing synonyms

18. `Alice Smith is also called Alice and 'A. Smith'.`

    Add two synonyms (*Alice* and *A._Smith*) to the instance *Alice_Smith*. After doing this, you can use the either form, `Alice` or `'A. Smith'`, to refer to the same instance in later statements.

19. `Persons are also called people.`

    Add the synonym *People* to the class *Person*, so that you can later make a statement such as "`Alice and Bob are people.`" with the same effect as "`Alice and Bob are persons.`".

20. `Forget that Alice Smith is also called 'A. Smith'.`

    Delete a synonym (or list of synonyms). This works for classes and instances. This statement does not affect changes already made to the ontology using the synonym.

Clearing the entire ontology

21. `Forget everything.`

    This deletes everything in the ontology.

Typing the names of classes and instances

22. Names are normalized using initial upper-case letters and the base forms of words with underscores between them, so that

- `Deliverables` and `deliverable` both refer to the class *Deliverable*, and
- `Alice Smith` and `alice smith` both refer to the instance *Alice_Smith*.

23. Names containing reserved words (see Figure 3.2), punctuation, prepositions (`at`, `of`, etc.) and determiners (`the`, `that`, `these`, etc.) must be enclosed in quotation marks (`'...'`).

    For example, `'Journal of Cell Biology'`, `'String Theory'` and `'Smith and Sons'` will not be interpreted correctly without them. Sentences containing the first two expressions without quotes are unparseable, whereas `Smith and Sons` will be interpreted as a list of two names separated by the keyword `and`. It is also important to use quotation marks in correctly positioned pairs.

Lists of classes and instances

24. Many CLOnE statements take a list of names as one or two of their arguments. A list can consist of only one name or of two or more names with a comma (`,`) or the keyword `and` or both (`, and`) separating the names. So the following lists are equivalent.

    ```
    Alice, Bob, and Charles
    Alice and Bob and Charles
    Alice, Bob, Charles
    ```

Sentences that refer to classes or instances

25. Some statements have the same structure (syntax) but take either classes or instances as arguments, for example (25C) and (25D) in the following list.:

    (A) `Books are a type of document.`

    (B) `'Syntactic Structures' is a book.`

    (C) `People are authors of documents.`

    (D) `Chomsky is the author of 'Syntactic Structures'.`

    In these cases, CLIE examines the ontology to see if the arguments are classes (as in 25C) or instances (as in 25D) and processes the statement (or generates an error message) appropriately. As long as you match classes with classes and instances with instances in such sentences (examples 7–12 above), CLIE will resolve them and interpret the sentences correctly.

Furthermore, 25B makes *Syntactic_Structures* an instance of class *Book* but 25C treats *Syntactic_Structures* as an instance of *Document*—in fact *Syntactic_Structures* is an instance of both *Book* and *Document* and is subject to all the properties defined for both classes. CLIE resolves this automatically for you.

Chapter 4

Implementation

The syntax of the controlled language is based principally on *chunks*, which are used to name classes, instances, properties and values, and *keyphrases*; POS (part-of-speech) tags and morphological analysis (stemming) also play a role.

Section 4.1 will present the procedural description of CLIE, then Section 4.2 will discuss the syntax and semantics of CLOnE. Finally Section 4.3 will highlight the ways in which CLIE consults the ontology (which in effect acts as input as well as output) in order to interpret the semantics of the CLOnE input.

4.1 Processing

Procedurally, CLIE's analysis consists of a GATE pipeline of Processing Resources (PRs) which are executed in the following sequence, as illustrated in Figure 4.1.

PR 1. The text is annotated into tokens[1] with the ANNIE English tokenizer.

PR 2. The text is split into sentences (again using the ANNIE tool for this purpose).

PR 3. The tokens are POS-tagged[2] with the Hepple tagger.

PR 4. The GATE morphological analyser adds stores each token's lemma[3] as a `root` feature.

[1] Linguistic *tokens* are the units of lexical meaning in a sentence, such as words, numbers and punctuation marks. In English (but not in all languages) they are usually separated by spaces, although the string *doesn't*, for example, would be divided into two tokens: *does+n't*.

[2] *POS-tagging* means marking each token with a code for its part of speech (noun, verb, adjective, etc.).

[3] The *lemma* is the canonical form of a word, typically its dictionary headword. For example, *go*, *goes*, *going*, *gone* and *went* all have the same lemma: *go*.

Chunk content in the input text	Features	
'Wiley and Sons'	canonical	Wiley_and_Sons
	root	wiley and son
	string	Wiley and Sons
'multiword expressions'	canonical	Multiword_Expressions
	root	multiword expression
	string	multiword expressions
Multiword expressions	canonical	Multiword_Expression
	root	multiword expression
	string	Multiword expressions

Table 4.1: Examples of `Chunk` annotations from quoted and unquoted chunks

Lookup.majorType == CLIE-Subclass
is a type of
are a type of
are types of

Table 4.2: Keyphrases indicating subclasses

PR 5. The CLIE QuoteFinder (a JAPE[4] transducer) identifies passages of the text in pairs of quotation marks (either single ' or double ") and annotates them as *chunks*. Each such Chunk annotation's span includes the quotation marks but its features are derived from the string and root features of the Token and SpaceToken annotations within the quotation marks, as shown in Table 4.1. (Chunks are the basis of names of classes, instances and properties and their role will become clearer in subsequent sections.)

PR 6. The CLIE gazetteer marks all the keywords and phrases with `Lookup` annotations indicating their signifance; for example, the phrases in Table 4.2 receive `Lookup` annotations with the feature (attribute-value pair) `majorType == CLIE-Subclass`. Appendix B lists the complete gazetteer with the feature or features for each phrase's `Lookup` annotation.

PR 7. The CLIE Chunker is a multi-phase JAPE transducer that marks *list separators* (commas and the word `and`) and then labels as chunks all continuous sequences of tokens excluding list separators, `Lookup` annotations, stops (`.`), tokens POS-tagged as prepositions and determiners, and chunks already marked by the QuoteFinder. Each chunk is marked with a `Chunk` annotation with features

[4] *JAPE* (Java Annotation Pattern Engine) is a language used in GATE for writing regular expressions over annotations and rules for adding more annotations and executing Java code when the patterns are matched. Section 4.2 will illustrate its use.

derived from from the string features of the Token and SpaceToken annotations within the chunk's span, as shown in Table 4.1.

PR 8. Finally, the CLIE JAPE transducer processes each sentence in the input text and manipulates the ontology appropriately—as Section 4.2 will explain in detail. This PR refers to the contents of the ontology in order to analyse the input sentences and check for errors; some syntactically identical sentences may have different results if they refer to existing classes, existing instances, or non-existent names, for example.

As Table 4.1 suggests, the *canonical* feature—from which the name of a new class or instance is derived—is generated differently for quoted and unquoted chunks. For quoted chunks, it is the catenation of the string values of the tokens and underscores for the space-tokens (which can represent literal spaces, tabs or newlines). For unquoted chunks, the lemmata of the tokens are used. This is reasonable because quoted chunks are intended principally for proper names such as *Wiley and Sons* and *Journal of Irreproducible Results*.

The Java code that tests chunks in the input text against existing classes and instances in the ontology returns a match if any of the three features of the chunk (canonical, root or string) is case-insensitively equal to any of those features of an existing class or instance; for example, the chunks `'multiword expressions'` and `Multiword expressions` match each other, although the class name in the ontology varies according to which one is first used to create the class.

4.2 Syntax and semantics

An input document in CLOnE consists of a series of sentences, each of which consists of *chunks*, *list separators*, *prepositions* and *keyphrases* and ends with a full stop. A parseable sentence matches the pattern (similar to a regular expression) left-hand side (LHS) of one of the rules in the CLIE JAPE transducer and the sentence's semantics are determined by the rule's right-hand side (RHS), which contains Java code to manipulate the ontology.

We now present the full current list of CLIE syntactic rules, for each of which is given the following details:

- the JAPE pattern (regular expression) of GATE annotations (please refer to Tables 4.3, 4.4 and 4.5 for explanations of the notation used);
- an informal statement of the pattern, using variables such as CLASS and INSTANCE (and CLASSES, for example, to indicate a list of one or more);
- one or more examples of CLOnE input; and

Notation	Explanation
SlantedText	a GATE annotation of the named type over one or more tokens of the input text
*	Kleene star
*Chunk**	zero or more consecutive occurrences of the *Chunk* annotation, for example
?	optional item
CLIE-Negate?	zero or one occurrence of the *CLIE-Negate* annotation, for example
(...)	grouping
*(A B)**	an empty list, *A B*, *A B A B*, etc.

Table 4.3: Notation used for JAPE syntax in this document

Annotation type	Significance
Chunk	name of a class, instance or property; value of a datatype property
Lookup	keyword or phrase from the gazetteer
Prep	preposition
Separator	comma (`,`) or `and`
Split	full stop (`.`)

Table 4.4: Annotation types used in CLIE JAPE patterns

- an explanation of the rule's semantics, i.e. what CLIE does when an input sentence matches the rule.

Many of the sentences also have negative forms indicated by the *CLIE-Negate?* optional element at the beginning of the pattern. The negative forms (`Forget...`) can be used to correct input errors (a form of "undo"-function) as well as to delete old (but not necessarily previously erroneous) information while editing an existing ontology.

Rule 1. *CLIE-Negate? CLIE-NewClass ChunkList Split*

There is/are CLASSES.
Forget that there is/are CLASSES.

`There are agents and documents.`

Create a new class immediately under the top class for each chunk in *ChunkList*. If negated, delete each class named in *ChunkList*.

Rule 2. *CLIE-Negate? ChunkList0 CLIE-InstanceOf Chunk1 Split*

INSTANCES is a/are CLASS.

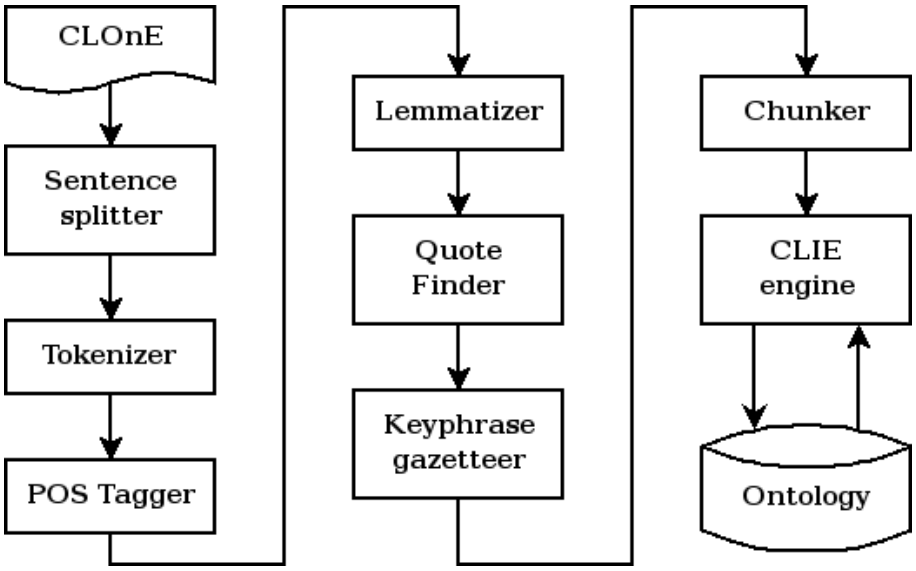

Figure 4.1: The CLIE pipeline

Abbreviation	Significance
DT	determiner (*Token.category==DT*)
Prep	preposition (*Token.category==IN*)
ChunkList	one or more chunks, separated by commas or `and` pattern: *(Chunk Separator)* Chunk*
CLIE-Have	*Lookup.majorType==CLIE-Have*
CLIE-NewClass	*Lookup.majorType==CLIE-NewClass*
...	...

Table 4.5: Abbreviations used in the CLIE JAPE patterns

Forget that INSTANCES is a/are CLASS.

```
'University of Sheffield' is a university.
Alice Jones and Bob Smith are persons.
```

For each chunk in *ChunkList0* create an instance of class *Chunk1*. If negated, delete each instance. If *Chunk1* does not name an existing class, generate an error.

Rule 3. *CLIE-Negate? ChunkList0 CLIE-Subclass Chunk1 Split*

CLASSES is/are a type/types of CLASS.
Forget that CLASSES is/are a type/types of CLASS.

```
Universities and persons are types of agent.  Dogs
are a type of mammal.  Forget that dogs are a type of
cat.
```

For each chunk in ChunkList0, if it already exists as a class, make it a subclass of the class named by Chunk1; if it does not exist, create a new class as a subclass of Chunk1.[5]

If the sentence is negated, unlink the subclass-superclass relationship (but do not delete the subclass).

If Chunk1 does not name an existing class, generate an error.

Rule 4. *CLIE-Negate? ChunkList0 CLIE-Have ChunkList1 Split*

CLASSES/INSTANCES have CLASSES/INSTANCES.
Forget that CLASSES/INSTANCES have CLASSES/INSTANCES.

```
Journals have articles.  'Journal of Knowledge
Management' has 'Crossing the Chasm'.
```

Iterate through the cross-product of chunks in ChunkList0 and chunks in ChunkList1. For each pair, if both are classes, create a property of the form *Domain_has_Range*. If both are instances, find a suitable property and instantiate it with those instances; if there is a class-instance mismatch or a suitable property does not exist, generate an error.

Rule 5. *CLIE-Negate? ChunkList0 CLIE-Have CLIE-Datatype Chunk1 Split*

CLASSES have DATATYPE DESCRIPTION.
Forget that CLASSES have DATATYPE DESCRIPTION.

```
Projects have string names.  Deliverables and
conferences have dates as deadlines.
```

[5]CLIE and the GATE Ontology API support multiple inheritance.

For each class named in ChunkList0, create a datatype property of the form *Domain_has_Chunk1*.[6]

Rule 6. *CLIE-Negate? ChunkList0 CLIE-Have Chunk1 CLIE-PropertyValue Chunk2 Split*

INSTANCE has DESCRIPTION with value VALUE.
Forget that INSTANCE has DESCRIPTION with value VALUE.

```
SEKT has name with value 'Semantically-Enabled
Knowledge Technology'.  D2.2.2' has deadline with
value 'M36'.
```

For each instance named in ChunkList0, find a suitable datatype property with the description Chunk1 and instantiate it with the data value Chunk2.

Rule 7. *CLIE-Negate? Chunk0 CLIE-Synonymous ChunkList1 Split*

CLASS/INSTANCE is/are also called/known as SYNONYMS.
Forget that CLASS/INSTANCE is/are also called/known as SYNONYMS.

```
Dogs are also called canines.  Bob Smith is also
called Bob.
```

Add all the chunks listed in ChunkList1 as synonyms of the class or instance named in Chunk0. The synonyms are indexed and can be used in subsequent statements of CLIE, although they do not affect the classes' and instances' primary names in the ontology. If negated, delete the synonym in ChunkList1 from the class or instance in Chunk0.

Synonyms are implemented as RDF-labels so they are saved in the OWL-Lite file that CLIE exports and can be used again when the same file is re-loaded.

Rule 8. *CLIE-NewClass ChunkList0 Separator? CLIE-Have ChunkList1 Split*

There are CLASSES, which have CLASSES.

```
There are projects, which have workpackages and
deliverables.
```

Create a class under the top class for each chunk in ChunkList0, and create properties of the form *Domain_has_Range* for the cross-product of new classes in ChunkList0 and existing classes in ChunkList1.

Rule 9. *CLIE-Negate? ChunkList0 CLIE-Copula DT? Chunk1 Prep ChunkList2 Split*

CLASSES/INSTANCES are DESCRIPTION PREPOSITION CLASSES/INSTANCES.
Forget that CLASSES/INSTANCES are DESCRIPTION PREPOSITION CLASSES/INSTANCES.

[6]The gazetteer distinguishes string, numeric and date types, but they are all currently implemented as strings.

`Persons are authors of documents. Carl Pollard and Ivan Sag are the authors of 'Head-Driven Phrase Structure Grammar'.`

Iterate through the cross-product of chunks in ChunkList0 and in ChunkList2. For each pair, if both name classes, create a property of the form *Domain_Chunk1_Prep_Range*.

If both name instances and a suitable property can be found, instantiate the property between the given instances. If there is a class-instance mismatch or one of the names cannot be dereferenced, an error message is produced.

This rule is a particularly good example of CLIE's use of information from the ontology to interpret the input sentences.

Rule 10. *CLIE-ClearAll Split*

`Forget everything.`

Clear the whole ontology (and start over).

Rule 11. *CLIE-Negate ChunkList Split*

Forget CLASSES/INSTANCES.

`Forget projects, journals and 'Department of Computer Science'.`

For each chunk in ChunkList, delete the named class or instance. (The GATE ontology API will automatically delete subclasses and instances of named classes, and properties and property definitions referring to named classes and instances.)

4.3 Using the ontology to interpret CLOnE input

The last stage of analysis, the CLIE JAPE transducer (PR 8 in Section 4.1), refers to the existing ontology in several ways in order to interpret the input sentences.

- Rules 4 and 9 can take classes or instances as their arguments. The Java code that interprets these rules refers to the ontology and behaves differently according to each argument's status in the ontology (class, instance, or non-existent).
  - If the domain and range arguments both refer to classes, the code creates a new property between those classes.
  - If the domain and range arguments both refer to instances and a suitable property definition exists, the code defines a new property value between the instances.

  - In other cases (either argument does not exist, or one is a class and the other an instance), the code generates an error message.

  Each syntactic rule is easier for the user to learn and the two functions (semantic interpretations) of each rule are intuitively distinguished by the user.

- When rules 4 and 9 are used to define property values between instances, the relevant Java code looks for a suitable property. It searches the ontology for a list of all the classes of which the domain instance is a direct or indirect member and a similar list for the range instance; then it scans up the two lists (from the direct classes to the top class in the ontology) until it finds a property whose "middle part" (e.g. *has* or *Author_of*) matches the corresponding string used in the sentence currently being processed.

  So in example 25 in Section 3.2 (page 19), *Chomsky* is a direct instance of *Person*, and *Syntactic_Structures* is a direct instance of *Book* and an indirect instance of *Document*. Sentence 25D is therefore interpreted as a definition of the property *Person_Author_of_Document* which was created by sentence 25C.

- Rule 11 for deleting a list of classes and instances checks each name on the list and calls the GATE Ontology API's `removeClass(OClass)` or `removeInstance(OInstance)` method as appropriate; it returns an error message for each name it cannot find in the ontology.

- Rule 2 checks that the class exists before attempting to create instances.

- In all the rules, the user can refer to a class or instance by its canonical name (derived from the name used when it was created) or any of the synonyms that have been added (and not subsequently deleted) by rule 7. The canonical names and synonyms are stored in an index and as RDF labels in the ontology. Common Java code used by all the rules looks up names in the index (with case-insensitive matching) and dereferences them to the correct classes and instances.

- Rule 3 verifies that *Chunk1* refers to an existing class in the ontology. For each name in *ChunkList0* that refers to an existing class, *Chunk1* is added as a direct superclass; for each new class name in *ChunkList0*, CLIE creates a new class as a direct subclass of *Chunk1*.

Chapter 5

Evaluation

We carried out a user evaluation of CLIE in comparison with Protégé in order to obtain quantitative measures and quantified subjective evaluations of usability as well as comments and suggestions.

5.1 Methodology

We prepared the following documents (given in full in Appendix A) for the users.

- The pre-test questionnaire (Figure A.15) asks for background information: how much each subject already knows about ontologies, the Semantic Web, Protégé and controlled languages. We scored this questionnaire by assigning each answer a value from 0 to 2 (from left to right) and dividing the total by 12 to obtain a score of 0–100.
- The short manual (Section A.1) introduces ontologies and provides "quick start" instructions for both pieces of software. Although much simpler, our manual was partly inspired by Protégé's *Ontology 101: Creating your First Ontology* documentation. [NM01]
- The post-test questionnaire for each tool (Figure A.16) is the *System Usability Scale*, a *de facto* standard for evaluating software usability, which also produces a score of 0–100. [Bro96]
- We devised the comparative questionnaire (Figure A.17) to measure each user's preference for one of the two tools. This form is scored similarly to SUS so that 0 would indicate a total preference for Protégé, 100 would indicate a total preference for CLIE, and 50 would result from marking all the questions *neutral*. On the reverse side (Figure A.18) and in discussion with the facilitator, we offered each user the opportunity to provide comments and suggestions.

- We prepared two lists of ontology-editing tasks (Figures A.11 and A.13) divided into three sublists covering the following task types:
  - creating subclasses,
  - creating instances, and
  - creating and defining properties.

We recruited 15 volunteers with varying experience levels and asked each subject to complete the pre-test questionnaire, to read the manual, and to carry out each of the two task lists with one of the two tools. Approximately half the users (8 of 15) carried out Task List A (Figure A.11) with CLIE and then Task List B (Figure A.13) with Protégé; the others (7 of 15) carried out A with Protégé and then B with CLIE.

We measured each user's time for each task list and in most cases (12 of 15) for each sublist. After each task list we asked the user to complete the SUS questionnaire for the specific tool used, and finally we asked him to complete the comparative questionnaire.

Section 5.2 explains the theoretical justification for our methodology; the remainder of this chapter presents the statistical results as well as a summary of the users' comments and suggestions for improving CLIE.

5.2 Background

Our methodology constitutes a *repeated-measures, task-based* evaluation: each subject carries out a similar list of tasks on both tools being compared.

We chose the SUS questionnaire as our principal measure of software usability because it is a *de facto* standard in this field. Although it superficially seems subjective and its creator called it "quick and dirty", it was developed according to the proper techniques for a Likert scale. [Bro96]

Furthermore, researchers at Fidelity Investments carried out a comparative study of SUS, three other published usability questionnaires and an internal questionnaire used at Fidelity, over a population of 123 subjects, to determine the sample sizes required to obtain consistent, accurate results. They found that SUS produced the most reliable results across all sample sizes; they noted a jump in accuracy to 75% at a sample size of 8, but recommended a sample of at least 12–14 subjects. [TS04]

As a reference for interpreting the results, average SUS scores are usually between 65 and 70. [Bai06] We will consider this to be the baseline for comparison in the next section.

Measure	min	mean	median	max
Pre-test scores	25	55	58	83
CLIE SUS rating	65	78	78	93
Protégé SUS rating	20	47	48	78
CLIE/Protégé preference	43	72	70	93

Table 5.1: Summary of the questionnaire scores

5.3 Statistical analysis

5.3.1 Statistical measures

Before presenting and interpreting the findings from our experimental data, we briefly explain the statistical measures used in the following sections.

A *95% confidence interval* calculated from a data sample is a range which is 95% likely to contain the mean score of the whole population which the sample represents. [JLP96]

A *correlation coefficient* over a set of pairs of numeric data is analogous to the appearance of a scatter graph or X-Y plot. $+1$ signifies a perfect correlation and corresponds to a graph in which all points lie on a straight line with a positive slope; -1 signifies a perfect inverse correlation (the points lie on a straight line with a negative slope); 0 indicate a complete lack of correlation (random distribution of the points). Values $> +0.7$ and < -0.7 are generally considered to indicate strong correlations.

The formula for Pearson's coefficients assumes that the two variables are linearly meaningful; physical measurements such as length and temperature are good examples of such variables. The formula for Spearman's coefficients, on the other hand, stipulates only ordinal significance (ranking) and is often considered more appropriate for subjective measurements (such as many in the social sciences). [CS71, HLR77, JLP96, Cal96, Sim06]

5.3.2 Quantitative findings

As the descriptive statistics in Table 5.1 show, the SUS scores for CLIE are generally above the baseline and distributed generally higher than those for Protégé, and scores on the comparative questionnaire are generally favourable to CLIE. We can also break the scores down according to the tool used and the task list (A or B) carried out and calculate confidence intervals as shown in Table 5.2; these indicate that for each task list and for the combined results, the larger population which our sample represents will also produce mean SUS scores for CLIE that are both higher than those for Protégé and above the SUS baseline.

Task List	Tool	Confidence interval
A	Protégé	33–65
A	CLIE	75–93
B	Protégé	30–58
B	CLIE	67–79
A&B	Protégé	37–56
A&B	CLIE	73–84

Table 5.2: Confidence intervals (95%) for the SUS scores

Measure	Measure	Pearson's	Spearman's
Pre-test	CLIE time	-0.06	-0.15
Pre-test	Protégé time	-0.13	-0.27
CLIE time	Protégé time	0.78	0.51
CLIE SUS	Protégé SUS	-0.31	-0.20
CLIE SUS	C/P Preference	0.68	0.63
Protégé SUS	C/P Preference	-0.62	-0.63
Pre-test	CLIE SUS	-0.17	-0.17
Pre-test	Protégé SUS	-0.16	-0.15
CLIE time	CLIE SUS	0.26	0.15
Protégé time	Protégé SUS	-0.17	-0.24
CLIE time	Protégé SUS	0.19	-0.01
Protégé time	CLIE SUS	0.42	0.44

Table 5.3: Correlation coefficients

We also generated Pearson's and Spearman's correlations coefficients for a wide range of data from the experiments; Table 5.3 shows the highlights of these calculations. In particular, we note the following points.

- The pre-test score has no correlation with task times or SUS results.
- The task times for both tools are moderately correlated with each other but there are no significant correlations between task times and SUS scores, so both tools are technically suitable for carrying out both task lists.
- As expected, the C/P preference score has a moderately strong correlation with the CLIE SUS score and a moderately strong negative correlation with the Protégé SUS score. (The SUS scores for the two tools also show a weak negative correlation with each other.) These figures confirm the coherence of the questionnaires as a whole.

Source		Tool order PC	CP	Total
G	GATE team	4	5	9
NG	others	4	2	6
Total		15	8	7

Table 5.4: Groups of subjects by source and tool order

5.3.3 Sample quality

Although our sample size ($n = 15$) satisifies the requirements for reliable SUS evaluations (as discussed in Section 5.2), it is also worthwhile to establish the consistency of two partitions of our sample, as enumerated in Table 5.4:

by tool order or task-tool assignment: subjects who carried out task list A on CLIE and then B on Protégé, in comparison with those who carried out A on Protégé then B on CLIE; and

by sample source: subjects drawn from the GATE development team, in comparison with others.

Tool order was divided almost evenly among our sample. Although the SUS scores differed slightly according to tool order (as indicated in Table 5.2), the task times given in Table 5.5 suggest that task lists A and B required similar effort.[1] We note that the SUS scores for each tool tend to be slightly lower for task list B, as shown in Table 5.6, and we suspect this may have resulted from the subjects' waning interest as the evaluation progressed.[2] But because Table 5.2 in particular shows consistent results between CLIE and Protégé for each task list, we conclude that our study was fair.

We must also consider the possibility of biased subjects drawn from colleagues of the developers and facilitator. As Table 5.4 shows, members of the GATE team constituted 60% of the user sample. The measures summarized in Table 5.7, however, show the following.

- Members of group G generally rated their own expertise higher (in the pre-test questionnaire) than those in group NG.
- Groups G and NG produced very similar ranges of SUS scores for each tool and of C/P preferences scores.

[1] The shortest time for task list A with Protégé, 1.5 minutes, is an outlier; the second shortest time for this task list and tool was 4.5 minutes.

[2] To eliminate the possibility of this effect with the same reliability, we would need twice as many subjects, each carrying out one task list with one tool (a *between-subject* experiment, in contrast to our *within-subject* experiment).

Task List	Tool	min	mean	median	max
A	CLIE	5.0	9.9	9.3	17.2
A	Protégé	*1.5	9.6	10.0	18.9
A	both	1.5	9.7	9.3	18.9
B	CLIE	4.9	9.1	8.1	18.3
B	Protégé	5.5	9.9	10.0	18.0
B	both	4.9	9.5	8.5	18.3
A&B	CLIE	4.9	9.5	8.5	18.3
A&B	Protégé	1.5	9.7	10.0	18.9

(**outlier: see note 1*)

Table 5.5: Times (in minutes) by task list and tool

Task List	Tool	min	mean	median	max
A	CLIE	65	84	88	93
A	Protégé	20	49	49	78
A	both	20	66	70	93
B	CLIE	65	73	73	88
B	Protégé	25	44	45	68
B	both	25	60	68	88
A&B	CLIE	65	78	78	93
A&B	Protégé	20	47	48	78

Table 5.6: SUS scores by task list and tool

Measure	Group	min	mean	median	max
Pre-test	G	25	62	67	83
	NG	33	44	42	58
CLIE SUS	G	65	78	78	90
	NG	65	78	78	93
Protégé SUS	G	25	46	48	70
	NG	20	47	46	78
C/P Preference	G	50	71	68	90
	NG	43	74	79	93
CLIE time	G	4.9	10.0	9.3	18.3
	NG	6.7	8.7	8.5	12.1
Protégé time	G	4.5	10.3	7.3	18.9
	NG	*1.5	8.8	10.3	10.6

(**outlier: see note 3*)

Table 5.7: Comparison of the two sources of subjects

- Groups G and NG produced largely overlapping ranges of task times, although some members of group G took significantly longer than any members of NG.[3]

These measures allay concerns about biased subjects: we conclude that groups G and NG were equally reliable so the sample as a whole is satisfactory for the SUS evaluation.

5.4 Subjects' suggestions and comments

The test users made several suggestions about the controlled language and the user interface.

- Several subjects complained that they needed to spell and type correctly the exact names of the classes and instances, such as "Journal of Knowledge Management" and that CLOnE is intolerant of typos and spelling mistakes. They suggested spell-checking and hinting (as provided in the Eclipse[4] UI) to alleviate this cognitive load.
- A related suggestion is to highlight the input text with different colours for classes, instances and properties, perhaps after the user clicks a *Check* button, which would be provided in addition to the *Run* button. The *Check* button could also suggest corrections and give error messages without affecting the ontology.
- Users complained that it was difficult to tell why some input sentences failed to have any effect, because CLIE does not explicitly indicate unparsable sentences.
- Some suggested that CLIE should automatically clear the input text box after each run, but they also realized this would make it more difficult to correct errors, because it is currently easy to prefix the keyword `forget` to incorrect sentences from the previous input without having to retype them.
- Some suggested making the *Run* button easier to find and putting the ontology viewer and the input box on the screen at the same time instead of in alternative panes.
- A few users said it would be useful to have an *Undo* button that would simply reverse the last input text, instead of having to `forget` all the sentences individually.

[3] The shortest time for Protégé in group NG, 1.5 minutes, is an outlier (in fact, the same one as mentioned in note 1 on page 34); the second shortest time for this tool and group was 9.5 minutes.

[4] http://www.eclipse.org/

5.5 Discussion

Our user evaluation consistently indicated that our subjects found CLIE signficantly more usable and preferable than Protégé for the straightforward tasks that we assigned. (Of course we make no claims about the more complicated knowledge engineering work for which Protégé but not CLIE is designed and intended.)

Our subjects made several interesting and useful suggestions for improvements to CLIE, many of which we already envisage developing in future work on this software beyond SEKT, as Chapter 6 will indicate. In particular, we will embed CLIE in wikis and perhaps other user interfaces which will eliminate some of the constraints imposed by running it in the GATE GUI, which is really intended for developing language engineering applications (such as CLIE itself) rather than for interactively editing language resources (such as documents and ontologies). The use of CLIE in NEPOMUK (Section 6.2) will especially address these issues.

Chapter 6

Ongoing and future work

6.1 Poleazy

We are assisting the EPSRC-funded Poleazy project to use CLIE to provide a controlled natural language interface for editing IT authorization policies (access to network resources such as directories and printers) stored as ontologies. This project will probably involve specialized extensions to the controlled language as well as a *virtuous circle* or round-trip information flow, to be completed by generation of CLOnE (as discussed in Section 6.4). [CS06]

6.2 Nepomuk

Work is being carried out in collaboration with DERI Galway[1] on the integration of CLIE as a web service for Round Trip Ontology Authoring (see also Section 6.4) for the Integrated Knowledge Articulation and Visualization Workbench as a part the Nepomuk Solution (The Social Semantic Desktop). We are also investigating the provision of CLOnE as a local service within one of the promised semantic wikis in addition to the possible use of CLIE for Semi-Semantic Annotation post editing.

NEPOMUK[2] aims to realize and deploy a comprehensive solution—methods, data structures, and a set of tools—for extending the personal computer into a collaborative environment, which improves the state of art in online collaboration and personal data management and augments the intellect of people by providing and organizing infor-

[1] Located at the National University of Ireland, Galway.
`http://www.deri.ie/`
`http://www.nuigalway.ie/`

[2] Project FP6-027705
`http://nepomuk.semanticdesktop.org/xwiki/bin/view/Main1/Project+Objectives`

mation created by single or group efforts. This solution is called the *Social Semantic Desktop*. This enhanced personal workspace (Desktop) will give information a well defined meaning, making it processable by the computer (Semantic), and will support the interconnection and exchange with other desktops and their users (Social).

6.3 Líon

In cooperation with DERI Galway, National University of Ireland, Galway, we are currently evaluating possible use cases for the inclusion of CLOnE within the Líon Project, specifically the WikiAnno subproject.

The Semantic Web aims to leverage and expand Web technology to cover new ground. In recent years, new standard proposals have been established, laying the foundation for further expansion and consolidation. Although the standards have been established to create a world-wide shared information space, the critical mass to overcome the initial chicken-and-egg problem for creating this space has not yet been established. One of the reasons is that development so far has only focused on standards and core technology development, but not on creating applications. The mission of the Líon project in DERI is to develop technologies and applications to create a social semantic information space. From the very beginning, the Web was a medium that helped to create communities and to connect individuals and communities - which may be communities of interest or practice and which also include businesses. The developed Semantic Web standards allow for a greater automation of information dissemination and connection of these communities. The Líon project is the research element of the DERI CSET (Centre for Science, Engineering and Technology), and is funded by Science Foundation Ireland[3] under grant number SFI/02/CE1/I131.

6.4 Generation of CLOnE from ontologies

In previous chapters, we discussed using CLIE to generate ontologies from input text. The reverse of the process involves the generation of CLOnE from an existing on ontology by Natural Language Generation (NLG), specifically *shallow NLG*. A prototype component for generating CLOnE has already been implemented as a GATE resource. The textual output of the generator is configured using a an XML file, which contains text templates that are instantiated and filled in with ontological values. The NL generator and the authoring process both combine to form a round-trip ontology authoring environment: one can start with an existing or empty ontology, produce CLOnE using the NL generator, modify or edit the text as requirement and subsequently parse the text back into the ontology using the CLOnE environment. The process can be repeated as necessary until

[3] `http://www.sfi.ie/`

the required result is obtained. Current developments in relation to the NL generator involve extending modifying the XML templates with respect to the generator's linguistic output in order to ensure compliance with the grammar and subsequent extensions such as alternatives for expressing the same message. This is essential in order to ensure that the generation component does not interfere with ontological data created or modified by CLIE. We refer the reader to [TPCB06] for specific implementation details.

6.5 Other uses

CLIE is being considered for use in the NeOn[4] FP6 project and the KnowledgeWeb[5] FP6 Network of Excellence.

The GATE development team at the University of Sheffield is considering embedding CLIE in an internally used web wiki to enhance it semantically.

[4] http://www.neon-project.org/web-content/
[5] http://knowledgeweb.semanticweb.org/

Bibliography

[Bai06] Bob Bailey. Getting the complete picture with usability testing. Usability updates newsletter, U.S. Department of Health and Human Services, March 2006.

[BK06] Abraham Bernstein and Esther Kaufmann. GINO—a guided input natural language ontology editor. In *5th International Semantic Web Conference (ISWC2006)*, 2006.

[BKGK05] A. Bernstein, E. Kaufmann, A. Göring, and C. Kiefer. Querying Ontologies: A Controlled English Interface for End-users. In *4th International Semantic Web Conference (ISWC)*, pages 112–126, November 2005.

[Bro96] J. Brooke. SUS: a "quick and dirty" usability scale. In P.W. Jordan, B. Thomas, B.A. Weerdmeester, and A.L. McClelland, editors, *Usability Evaluation in Industry*. Taylor and Francis, London, 1996.

[BS06] François Bry and Uta Schwertel, editors. *REWERSE Annual Meeting 2006*, March 2006.

[Cal96] Judith Calder. Statistical techniques. In Roger Sapsford and Victor Jupp, editors, *Data Collection and Analysis*, chapter 9. Open University, 1996.

[CMBT02] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*, 2002.

[CS71] T. G. Connolly and W. Sluckin. *An Introduction to Statistics for the Social Sciences*. Macmillan, third edition, 1971.

[CS06] David Chadwick and Angela Sasse. The virtuous circle of expressing authorization policies. In *Semantic Web Policy Workshop*, Athens, Georgia, 2006.

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

[FKK+06] Norbert E. Fuchs, Kaarel Kaljurand, Tobias Kuhn, Gerold Schneider, Loic Royer, and Michael Schröder. Attempto Controlled English and the semantic web. Deliverable I2D7, REWERSE Project, April 2006.

[HLR77] David K. Hildebrand, James D. Laing, and Howard Rosenthal. *Analysis of Ordinal Data*. Quantititate Applications in the Social Sciences. Sage, 1977.

[Hoe04] Stefan Hoefler. The syntax of Attempto Controlled English: An abstract grammar for ACE 4.0. Technical Report ifi-2004.03, Department of Informatics, University of Zurich, 2004.

[JLP96] Jr. John L. Phillips. *How to Think about Statistics*. W. H. Freeman and Company, New York, 1996.

[Kal06a] Kaarel Kaljurand. From ACE to OWL and from OWL to ACE. In Bry and Schwertel [BS06].

[Kal06b] Kaarel Kaljurand. Writing owl ontologies in ace. Technical report, University of Zurich, August 2006.

[Kau06] E. Kaufmann. Talking to the semantic web—query interfaces to ontologies for the casual user. In *5th International Semantic Web Conference (ISWC)*, 2006.

[KF06] Kaarel Kaljurand and Norbert E. Fuchs. Bidirectional mapping between OWL DL and Attempto Controlled English. In *Fourth Workshop on Principles and Practice of Semantic Web Reasoning*, Budva, Montenegro, June 2006.

[KL03] B. Katz and J. Lin. Selectively using relations to improve precision in question answering. In *Proceedings of the EACL-2003 Workshop on Natural Language Processing for Question Answering, April 2003*, 2003.

[Kuh06] Tobias Kuhn. Attempto Controlled English as ontology language. In Bry and Schwertel [BS06].

[LM04] Vanessa Lopez and Enrico Motta. Ontology driven question answering in AquaLog. In *NLDB 2004 (9th International Conference on Applications of Natural Language to Information Systems)*, Manchester, 2004.

[LMU06] V. Lopez, E. Motta, and V. S. Uren. Poweraqua: Fishing the semantic web. In *ESWC*, pages 393–410, 2006.

[NM01] Natalya F. Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report KSL-01-05, Stanford Knowledge Systems Laboratory, March 2001.

[Sim06] Steve Simon. Stats: Steve's attempt to teach statistics. Technical report, Children's Mercy Hospitals & Clinics, Kansas City, Missouri, November 2006.

[Sow02] J. Sowa. Architectures for intelligent systems. *IBM Systems Journal*, 41(3), 2002.

[TPCB06] V. Tablan, T. Polajnar, H. Cunningham, and K. Bontcheva. User-friendly ontology authoring using a controlled language. In *5th Language Resources and Evaluation Conference*, 2006.

[TS04] Thomas S. Tullis and Jacqueline N. Stetson. A comparison of questionnaires for assessing website usability. In *Usability Professionals' Association Conference*, Minneapolis, Minnesota, June 2004.

Appendix A

Evaluation documents

A.1 Training manual

An *ontology* is a formal representation of knowledge about a domain: it contains information about the objects and types of objects in that domain and the relationships between them. *Formal* here means basically "machine-readable"—the information is stored in a well-defined way so that computer programs can read and analyse it and reason with it. In recent years ontologies have become very important to scientific research because they help with the digital classification and retrieval of human-readable information (such as research papers and other documents).

An ontology consists of classes, instances and properties. The classes and instances are often drawn in a tree as shown in Figure A.1.

A *class* is a description of a set or the name of a type of thing, such as `Person` or `Document`. Classes are arranged in a hierarchy, so that the `Document` class might have several subclasses (subtypes), `Book`, `Journal` and `Article`. In this example `Document` is the "direct superclass" of `Book`, `Journal` and `Article`.

An *instance* or *individual* is one member of a class; for example, `Syntactic Structures` is an instance of the class `Book`, and `Noam Chomsky` is an instance of the class `Person`. Because of the superclass-subclass relationship, `Syntactic Structures` is also an instance of the class `Document`.

A *property* is a relation between two classes which can be instantiated between instances. We can for example create a property to express the idea that "persons are authors of documents", and then define an instance of this property to express the idea that "Noam Chomsky is the author of *Syntactic Structures*". Note that a property has an inherent "direction"; in other words, the two arguments of an property cannot (usually) be interchanged: it is not the case, for example, that documents can be authors of persons.

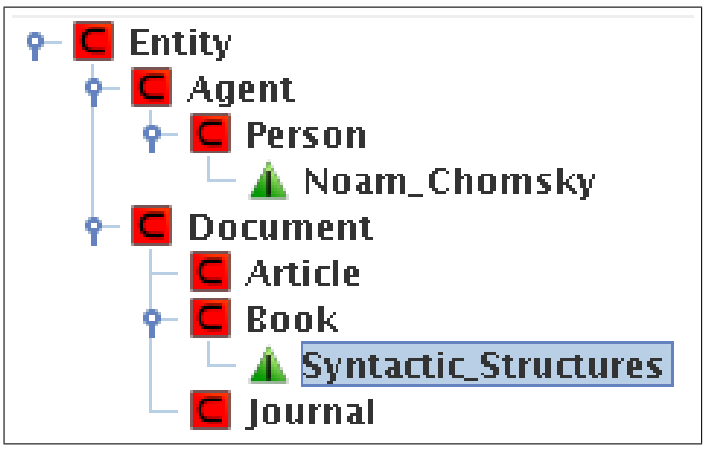

Figure A.1: Graphical depiction of classes and instances

A.1.1 CLIE How-To

The facilitator will start CLIE and load the initial data, so that you will have a window like the one shown in Figure A.2 to work with. You can click on the buttons or tabs across the top to bring up the following panes.

Messages This pane explains in detail what CLIE has just done and includes error messages. You can distinguish the error messages by the word *WARNING*, and probably ignore the *INFO* messages.

Text input In this pane (shown in Figure A.2) you will type statements in the controlled language explained below. To clear the input, select all the text with the mouse and press the backspace key. You can also edit individual parts of the text normally using the arrow and backspace keys and the mouse.

Ontology This pane (shown in Figure A.3) shows you the state of the ontology, represented by a class and instance diagram (top left), a general list of properties (below), and information about the selected class or instance (right). Click on classes or instances to change the information in the right-hand section. Before you begin the tasks, you might wish to look at the initial ontology to become familiar with it.

To run CLIE on your current input text, right click on the word *CLIE* near the top of the left-hand pane and click *Run* in the menu that appears. As you are probably aware, human language does not lend itself to precise computer processing; it contains stylistic variations, ambiguities and other features that make it difficult for computers to interpret. One way to let people write instructions and data so that computers can handle the input correctly is to use an artificial language, such as a computer programming language. Another approach is to use a *controlled language*, which is a restricted subset of a natural language such as English. All the sentences in the controlled language are human-readable sentences in English and have a specific meaning for the computer program, but not all English sentences are valid in the controlled language. In one of the programs you will test today, you will type sentences in a controlled language and run a program that interprets them in order to add more classes, instances and properties to a simple ontology representing a digital library (a computerized record of information about documents).

The controlled language used in CLIE is designed to be easy to learn from examples. (To avoid giving you the answers to your tasks here, we provide examples about research projects.) The controlled language consists of keywords (including punctuation) and names (of classes, instances and properties). In the following examples, keywords are underlined.

- Manipulating classes

  1. `Universities and companies are types of partner.`

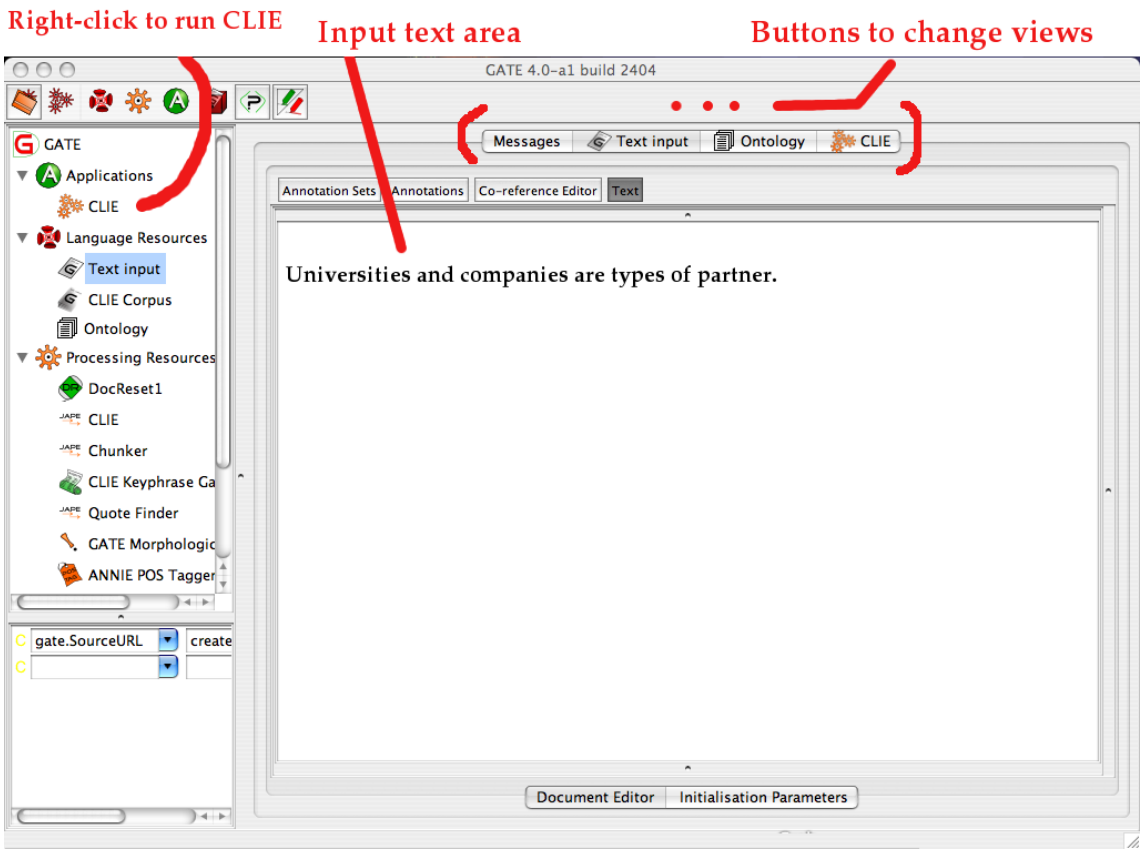

Figure A.2: CLIE Text input

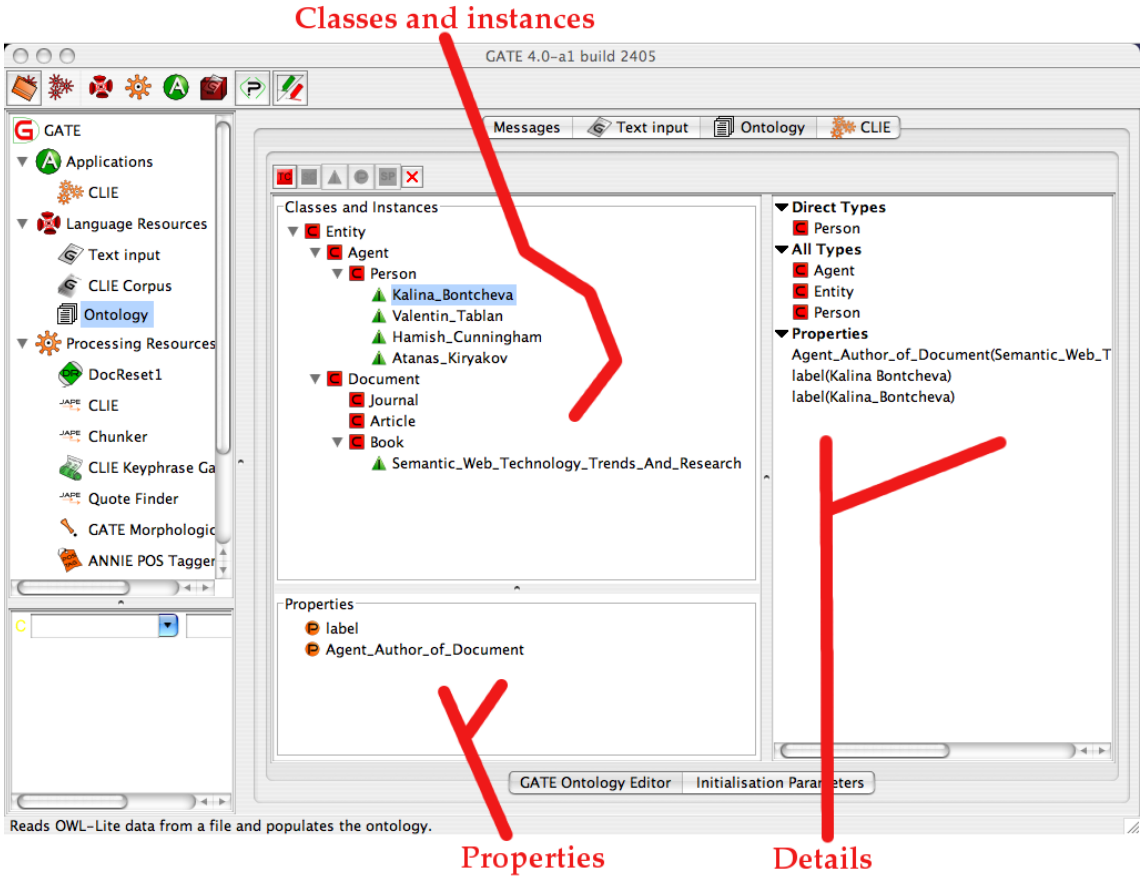

Figure A.3: CLIE Ontology viewer

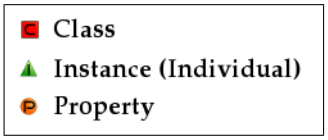

Figure A.4: Symbols used in CLIE

The class `Partner` must already exist. Make classes `University` and `Company` direct subclasses of `Partner` and create the first two classes if they do not already exist. (A class can have more than one direct superclass.)

2. `Forget that universities are types of partner.`

   Unlink the subclass-superclass relationship. This statement does not delete any classes.

- Manipulating instances

3. `'University of Sheffield' is a university.`

   Create an instance `University of Sheffield` of the class `University` (which must already exist). Because the name contains a preposition (`of`) it needs to be enclosed in quotation marks—these tell the CLIE program to treat everything between them as one name.

4. `'Smith and Sons' and 'Jones Ltd.' are companies.`

   Create two instances `Smith and Sons` and `Jones Ltd` of the class `Company` (which must already exist). The names are quoted because the first one contains a keyword (`and`) and the second one contains punctuation (`.`). (Figure A.5 lists all the keywords.)

- Deleting classes and instances

5. `Forget 'Smith and Sons', Alice Smith and projects.`

   Delete the instance `Smith and Sons` and `Alice Smith` and the class `Project`. In this statement, the list can contain a mixture of classes and instances.

- Manipulating properties

6. `Persons are authors of deliverables.`

   If the classes `Person` and `Deliverable` exist, define a property `Person Author of Deliverable` between them.

7. `Forget that persons are authors of deliverables.`

   Delete the property defined in the last example.

8. `Alice Smith and Bob Davis are authors of 'D2.3.4'.`

   If `Alice Smith` and `Bob Davis` are instances of `Person` and `D2.3.4` is an instance of `Deliverable`, and the property already exists, create two property definitions to indicate that they are authors of it. (`D2.3.4` must be quoted because it contains punctuation (`.`).

9. `Forget that Bob Davis is author of 'D2.3.4'.`

   Remove the property definition for one of the authors in the previous example. (This leaves the other author defined.)

- Typing the names of classes and instances

10. Names are normalized using initial upper-case letters and the base forms of words with underscores between them, so that `Deliverables` and `deliverable` both refer to the class `Deliverable`, and `Alice Smith` refers to the instance `Alice_Smith`.
11. Names containing reserved words (see Figure A.5), punctuation, prepositions (such as *of*) and determiners (`the`, `that`, `these`, etc.) must be enclosed in quotation marks (`'...'`). For example, `'Journal of Cell Biology'`, `'Smith and Sons'` and `'String Theory'` will not be interpreted correctly without them.

In order to carry out the tasks we ask you to do, you can alternate between the ontology viewer and the input text box. When you are satisfied that your input text is probably right, click "Run" in the CLIE pane and check the results in the ontology viewer. You can make corrections by undoing mistakes with "`Forget...`" statements and trying again with new statements.

When you click "Run", CLIE processes your input text from the top down, so it is important (for example) to define a new class before using it to define instances, subclasses or properties—although you can do all these steps in the same input text.

```
and                 have                text as
are                 is                  texts
are a type of       is a                texts as
are also called     is a type of        textual
are also known as   is also called      textual as
are called          is also known as    that can have
are known as        is an               that has
are types of        is called           that have
can have            is known as         there are
date                number              there is
date as             number as           which are
dates               numbers             which can have
dates as            numbers as          which has
delete all          numeric             which have
delete everything   numeric as          which is
forget              string              with value
forget all          string as           .
forget everything   strings             ,
forget that         strings as
has                 text
```

Figure A.5: Reserved words and phrases

A.1.2 Protégé How-To

The facilitator will start Protégé and load the initial data, so that you will have a window like the one shown in Figure A.6 to work with. The named buttons across the top have the following roles.

OWLClasses As shown in Figure A.6, this button brings up the *Subclass Explorer*, which shows a hierarchical diagram of the classes in the ontology and which you can use to select a specific class to work with, and the *Class Editor*, used for editing an individual class.

You can right-click on a class in the Subclass Explorer to get a menu which will allow you to create subclasses and individuals (instances) of existing classes, and you can drag and drop classes to move them in the class hierarchy.

Properties This produces the *Property Browser* as shown in Figure A.7, a list of the properties in the ontology.

You can create new properties by selecting the correct kind of property ("Object" or "Datatype") and clicking the first button after the list heading ("Object properties" or "Datatype properties"). In the *Property Editor* you can select the domain and range of each property.

Individuals This produces the display shown in Figure A.8, including the *Class Browser* (similar to the Subclass Explorer), the *Instance Browser* (which shows a list of the instances of the class currently selected in the Class Browser),and the *Individual Editor*, which lets you edit and fill in the property definitions of the instance selected in the Instance Browser.

The second button with a purple symbol in the Instance Browser provides another means of adding instances to the selected class.

You can ignore the **Metadata** and **Forms** buttons and panes.

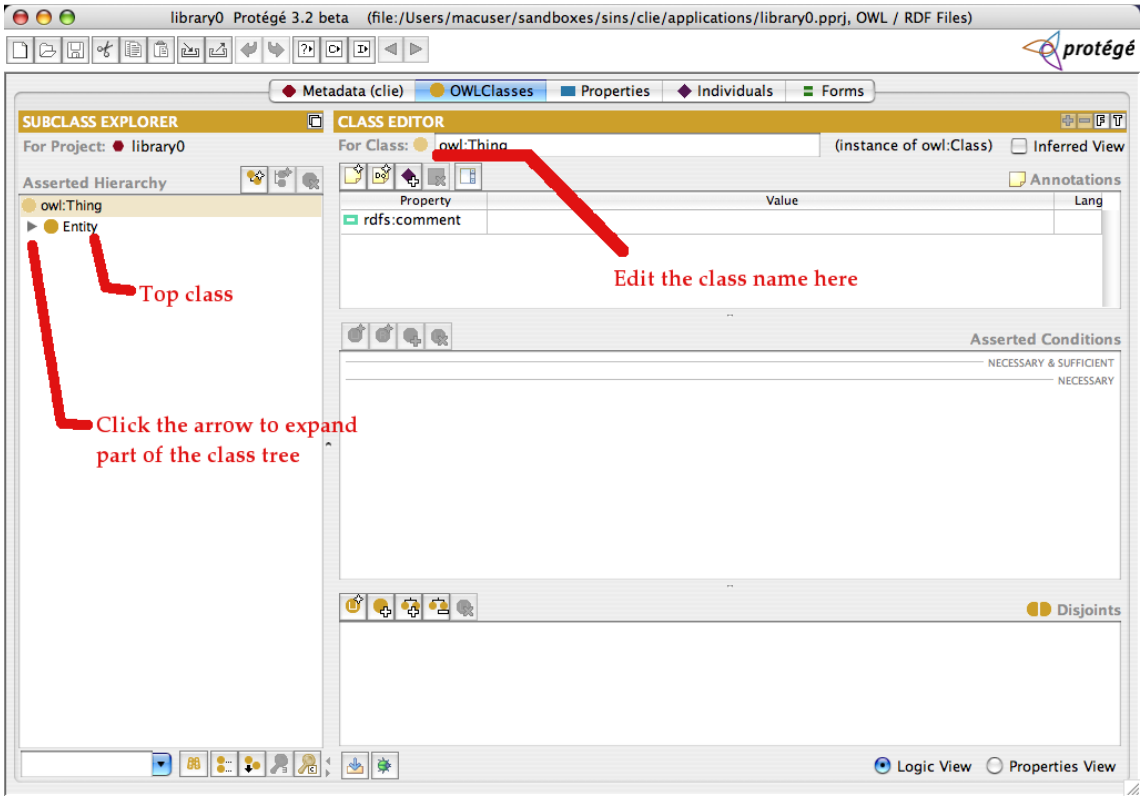

Figure A.6: Protégé's Subclass Explorer and Class Editor

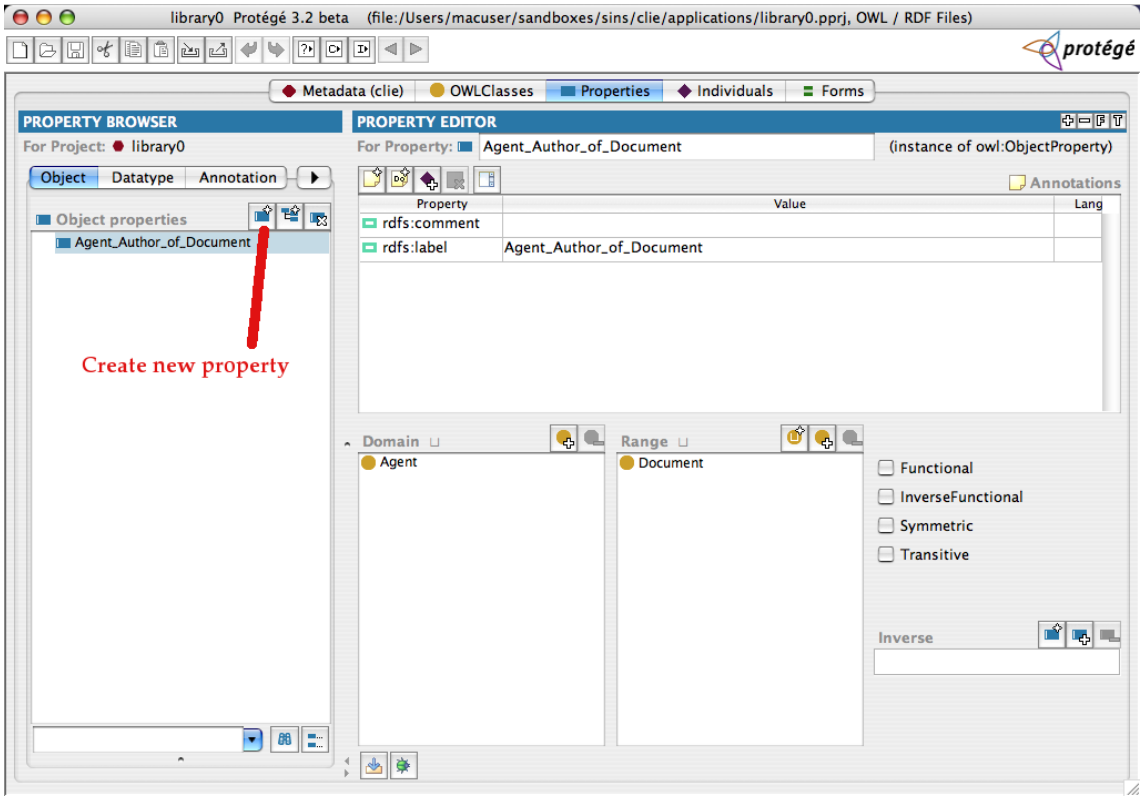

Figure A.7: Protégé's Property Browser and Property Editor

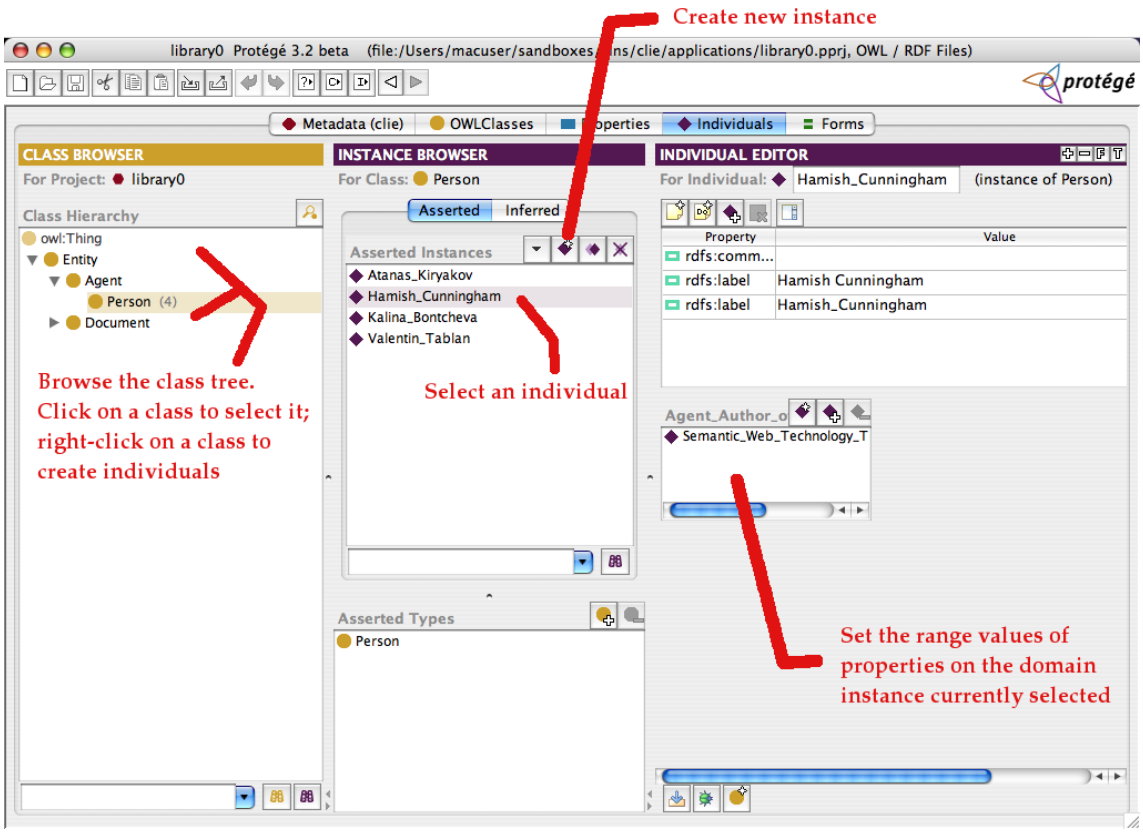

Figure A.8: Protégé's Instance Browser and Individual Editor

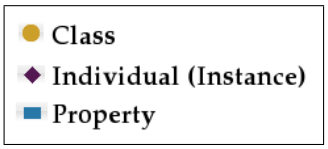

Figure A.9: Symbols used in Protégé

A.2 Test procedure, tasks and questionnaires

First we asked each subject to complete the pre-test questionnaire shown in Figure A.15.

We then gave him either CLIE or Protégé loaded with the initial ontology shown in Figure A.10 and asked him to carry out the groups of related tasks (Task List A) shown in Figure A.11, while we recorded the time taken to complete each group.

We then asked the subject to complete the questionnaire shown in Figure A.16. We then gave the subject the other tool (Protégé or CLIE, respectively) loaded with the ontology in the state that would result from correctly carrying out the tasks listed above, as shown in Figure A.12, and asked him to carry out the additional groups of tasks (Task List B) in Figure A.13 with the second tool.

We then saved the ontology to examine later for correctness; the correct result is illustrated in Figure A.14. We asked the subject to complete the questionnaire shown in Figure A.16 and then separately the one in Figures A.17 and A.18.

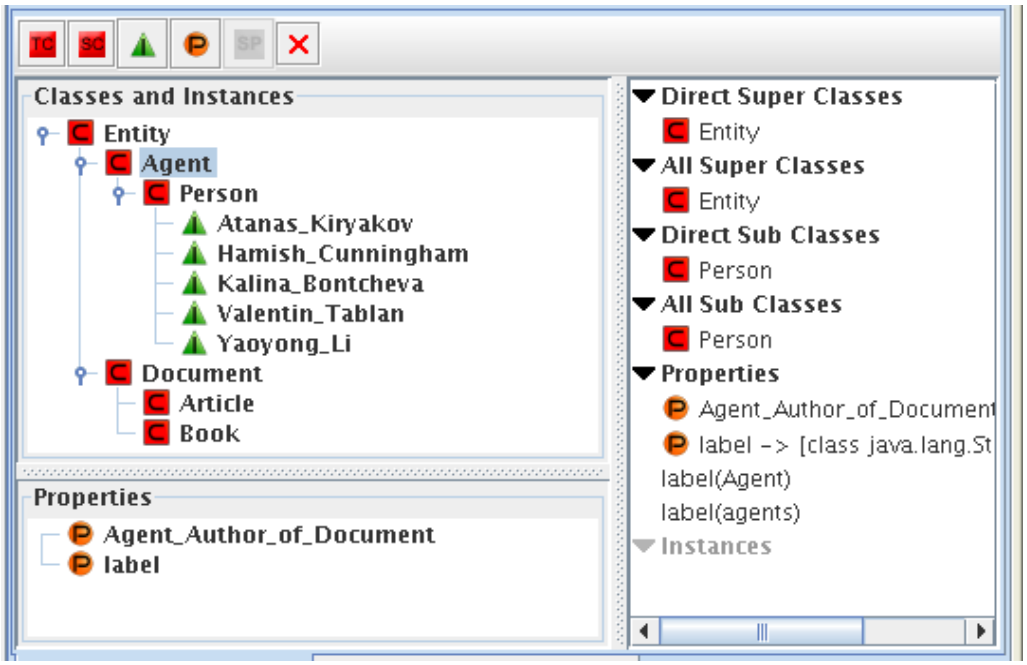

Figure A.10: Initial ontology (viewed in GATE)

- Class tasks
  - Create a subclass *Periodical* of *Document*.
  - Create a subclass *Journal* of *Periodical*.
- Instance tasks
  - Create an instance *Crossing the Chasm* of class *Article*.
  - Create an instance *Journal of Knowledge Management* of class *Journal*.
- Property tasks
  - Create a property that agents are publishers of documents.
  - Define a property that Hamish Cunningham, Kalina Bontcheva and Yaoyong Li are authors of *Crossing the Chasm*.

Figure A.11: Task list A

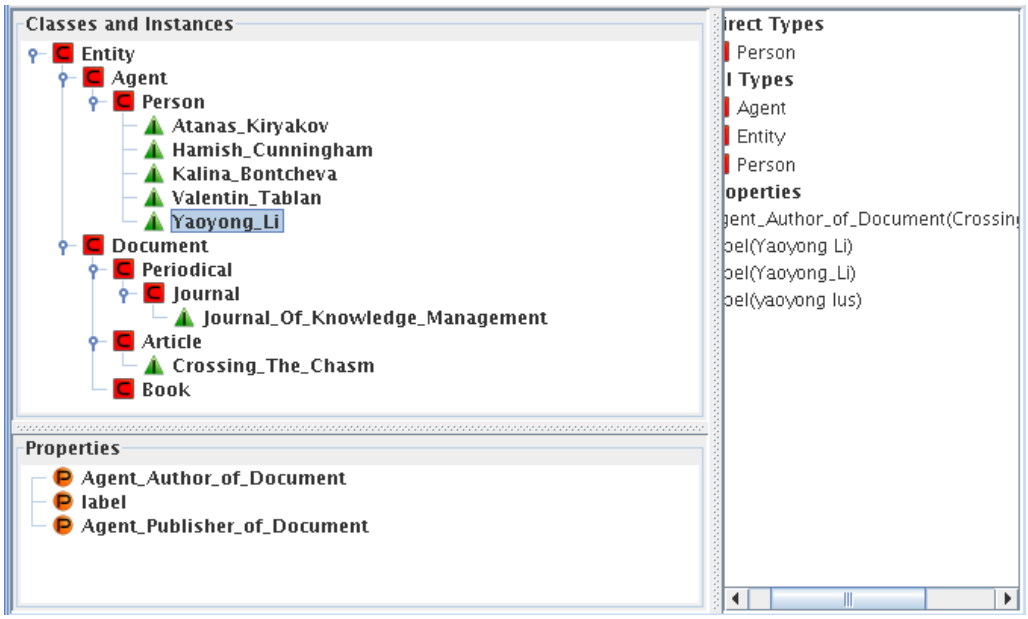

Figure A.12: Intermediate ontology (viewed in GATE)

- Class tasks
  - Create a subclass *Institution* of *Agent*.
  - Create a subclass *Company* of *Institution*.
- Instance tasks
  - Create an instance *Wiley and Sons* of class *Company*.
  - Create an instance *Trends and Research* of class *Book*.
- Property tasks
  - Define a property that *Wiley and Sons* is publisher of *Trends and Research*.
  - Define a property that *Wiley and Sons* is publisher of *Crossing the Chasm*.

Figure A.13: Task list B

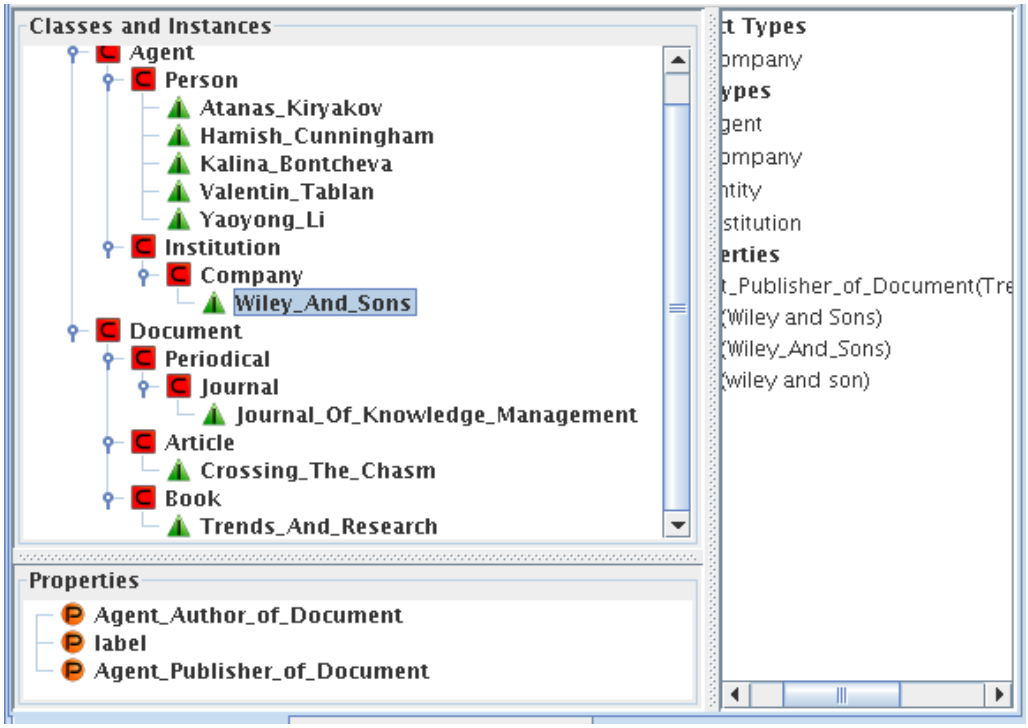

Figure A.14: Final ontology (viewed in GATE)

1 I understand the term “Semantic Web”. No ☐ A little ☐ Yes ☐

2 I am familiar with ontologies. No ☐ A little ☐ Yes ☐

3 I have worked with ontologies. Never ☐ Sometimes ☐ Often ☐

4 I have edited or designed an ontology. Never ☐ Sometimes ☐ Often ☐

5 I understand the term “controlled language”. No ☐ A little ☐ Yes ☐

6 I have used a controlled language. Never ☐ Sometimes ☐ Often ☐

Figure A.15: Pre-test questionnaire

		Strongly disagree	Disagree	Neutral	Agree	Strongly agree
1	I think that I would like to use this system frequently.	☐	☐	☐	☐	☐
2	I found the system unnecessarily complex	☐	☐	☐	☐	☐
3	I thought the system was easy to use	☐	☐	☐	☐	☐
4	I think that I would need the support of a technical person to be able to use this system	☐	☐	☐	☐	☐
5	I found the various functions in this system were well integrated	☐	☐	☐	☐	☐
6	I thought there was too much inconsistency in this system	☐	☐	☐	☐	☐
7	I would imagine that most people would learn to use this system very quickly	☐	☐	☐	☐	☐
8	I found the system very cumbersome to use	☐	☐	☐	☐	☐
9	I felt very confident using the system	☐	☐	☐	☐	☐
10	I needed to learn a lot of things before I could get going with this system	☐	☐	☐	☐	☐

Figure A.16: Post-test questionnaire for each tool

		CLIE		**CLIE**			**Protégé**		**Protégé**	
1	I found one system's documentation easier to understand.	much easier	☐	easier	☐	neutral ☐	easier	☐	much easier	☐
2	I particularly disliked using one system.	disliked strongly	☐	disliked	☐	neutral ☐	disliked	☐	disliked strongly	☐
3	I found one system easier to use.	much easier	☐	easier	☐	neutral ☐	easier	☐	much easier	☐
4	One system was harder to learn.	much harder	☐	harder	☐	neutral ☐	harder	☐	much harder	☐
5	I would prefer to use one system again.	strongly prefer	☐	prefer	☐	neutral ☐	prefer	☐	strongly prefer	☐
6	I found one system more complicated.	much more complex	☐	more complex	☐	neutral ☐	more complex	☐	much more complex	☐
7	I found it easier to control classes and subclasses in one system.	much easier	☐	easier	☐	neutral ☐	easier	☐	much easier	☐
8	New properties were easier to create with one system.	much easier	☐	easier	☐	neutral ☐	easier	☐	much easier	☐
9	It was difficult to create instances in one system.	much harder	☐	harder	☐	neutral ☐	harder	☐	much harder	☐
10	It was awkward to fill (or define) properties in one system.	very awkward	☐	awkward	☐	neutral ☐	awkward	☐	very awkward	☐

Figure A.17: Post-test questionnaire comparing the tools

Do you have any comments on either or both systems?
Do you have any specific problems to report?
Do you have any suggestions for improving either system?

Figure A.18: Post-test questionnaire comparing the tools

Appendix B

CLIE gazetteer (keywords and phrases)

This table lists all the phrases annotated by the CLIE gazetteer, as described by PR 6 in Section 4.1 (page 22). The gazetteer marks each listed phrase with a `Lookup` annotation with the specified `majorType` feature; some phrases' annotations also have a `minorType` feature.

majorType	minorType	gazetteer entry
CLIE-ClearAll		delete all
CLIE-ClearAll		delete everything
CLIE-ClearAll		forget all
CLIE-ClearAll		forget everything
CLIE-Copula		are
CLIE-Copula		is
CLIE-Datatype	date	date as
CLIE-Datatype	date	date
CLIE-Datatype	date	dates as
CLIE-Datatype	date	dates
CLIE-Datatype	numeric	number as
CLIE-Datatype	numeric	number
CLIE-Datatype	numeric	numbers as
CLIE-Datatype	numeric	numbers
CLIE-Datatype	numeric	numeric as
CLIE-Datatype	numeric	numeric
CLIE-Datatype	string	string as
CLIE-Datatype	string	string
CLIE-Datatype	string	strings as
CLIE-Datatype	string	strings
CLIE-Datatype	string	text as
CLIE-Datatype	string	text
CLIE-Datatype	string	texts as

majorType	minorType	gazetteer entry
CLIE-Datatype	string	texts
CLIE-Datatype	string	textual as
CLIE-Datatype	string	textual
CLIE-Have		can have
CLIE-Have		has
CLIE-Have		have
CLIE-Have		that can have
CLIE-Have		that has
CLIE-Have		that have
CLIE-Have		which can have
CLIE-Have		which has
CLIE-Have		which have
CLIE-InstanceOf		are
CLIE-InstanceOf		is a
CLIE-InstanceOf		is an
CLIE-Negate		forget that
CLIE-Negate		forget
CLIE-NewClass		there are
CLIE-NewClass		there is
CLIE-PropertyValue		which are
CLIE-PropertyValue		which is
CLIE-PropertyValue		with value
CLIE-Subclass		are a type of
CLIE-Subclass		are types of
CLIE-Subclass		is a type of
CLIE-Synonymous		are also called
CLIE-Synonymous		are also known as
CLIE-Synonymous		are called
CLIE-Synonymous		are known as
CLIE-Synonymous		is also called
CLIE-Synonymous		is also known as
CLIE-Synonymous		is called
CLIE-Synonymous		is known as