Using GATE as an Environment for Teaching NLP

Kalina Bontcheva, Hamish Cunningham, Valentin Tablan, Diana Maynard, Oana Hamza
Department of Computer Science
University of Sheffield
Sheffield, S1 4DP, UK
{kalina,hamish,valyt,diana,oana}@dcs.shef.ac.uk

Abstract

In this paper we argue that the GATE architecture and visual development environment can be used as an effective tool for teaching language engineering and computational linguistics. Since GATE comes with a customisable and extendable set of components, it allows students to get hands-on experience with building NLP applications. GATE also has tools for corpus annotation and performance evaluation, so students can go through the entire application development process within its graphical development environment. Finally, it offers comprehensive Unicode-compliant multilingual support, thus allowing students to create components for languages other than English. Unlike other NLP teaching tools which were designed specifically and only for this purpose, GATE is a system developed for and used actively in language engineering research. This unique duality allows students to contribute to research projects and gain skills in embedding HLT in practical applications.

1 Introduction

When students learn programming, they have the benefit of integrated development environments, which support them throughout the entire application development process: from writing the code, through testing, to documentation. In addition, these environments offer support and automation of common tasks, e.g., user interfaces can be designed easily by assembling them visually from components like menus and windows. Similarly, NLP and CL students can benefit from the existence of a graphical development environment, which allows them to get hands-on experience in every aspect of developing and evaluating language processing modules. In addition, such a tool would enable students to see clearly the practical relevance and need for language processing, by allowing them to experiment easily with building NLP-powered (Web) applications.

This paper shows how an existing infrastructure for language engineering research – GATE (Cunningham et al., 2002a; Cunningham, 2002) – has been used successfully as an NLP teaching environment, in addition to being a successful vehicle for building NLP applications and reusable components (Maynard et al., 2002; Maynard et al., 2001). The key features of GATE which make it particularly suitable for teaching are:

- The system is designed to separate cleanly low-level tasks such as data storage, data visualisation, location and loading of components and execution of processes from the data structures and algorithms that actually process human language. In this way, the students can concentrate on studying and/or modifying the NLP data and algorithms, while leaving the mundane tasks to GATE.

- Automating measurement of performance of language processing components and facilities for the creation of the annotated corpora needed for that.

- Providing a baseline set of language processing components that can be extended and/or replaced by students as required. These modules typically separate clearly the linguistic data from the algorithms that use it, thus allowing teachers to present them separately and the students to adapt the modules to new domains/languages by just modifying the linguistic data.

- It comes with exhaustive documentation, tutorials, and online movie demonstrations, available on its Web site (http://gate.ac.uk).

GATE and its language processing modules were developed to promote robustness and scalability of NLP approaches and applications, with an emphasis on language engineering research. Therefore, NLP/LE courses based on GATE offer students the opportunity to learn from non-toy applications, running on big, realistic datasets (e.g., British National corpus or news collected by a Web crawler). This unique research/teaching duality also allows students to contribute to research projects and gain skills in embedding HLT in practical applications.

2 GATE from a Teaching Perspective

GATE (Cunningham et al., 2002a) is an architecture, a framework and a development environment for human language technology modules and applications. It comes with a set of reusable modules, which are able to perform basic language processing tasks such as POS tagging and semantic tagging. These eliminate the need for students to re-implement useful algorithms and modules, which are pre-requisites for completing their assignments. For example, Marin Dimitrov from Sofia University successfully completed his masters' degree by implementing a lightweight approach to pronominal coreference resolution for named entities[1], which uses GATE's reusable modules for the earlier processing and builds upon their results (see Section 4).

For courses where the emphasis is more on linguistic annotation and corpus work, GATE can be used as a corpus annotation environment (see http://gate.ac.uk/talks/tutorial3/). The annotation can be done completely manually or it can be bootstrapped by running some of GATE's processing resources over the corpus and then correcting/adding new annotations manually. These facilities can also be used in courses and assignments where the students need to learn how to create data for quantitative evaluation of NLP systems.

If evaluated against the requirements for teaching environments discussed in (Loper and Bird, 2002), GATE covers them all quite well. The graphical development environment and the JAPE language facilitate otherwise difficult tasks. Inter-module consistency is achieved by using the annotations model to hold language data, while extensibility and modularity are the very reason why GATE has been successfully used in many research projects (Maynard et al., 2000). In addition, GATE also offers robustness and scalability, which allow students to experiment with big corpora, such as the British National Corpus (approx. 4GB). In the following subsections we will provide further detail about these aspects of GATE.

2.1 GATE's Graphical Development Environment

GATE comes with a graphical development environment (or GATE GUI) that facilitates students in inspecting the language processing results and debugging the modules. The environment has facilities to view documents, corpora, ontologies (including the popular Protégé editor (Noy et al., 2001)), and linguistic data (expressed as annotations, see below), e.g., Figure 1 shows the document viewer with some annotations highlighted. It also shows the resource panel on the left with all loaded appli-

[1]The thesis is available at `http://www.ontotext.com/ie/thesis-m.pdf`

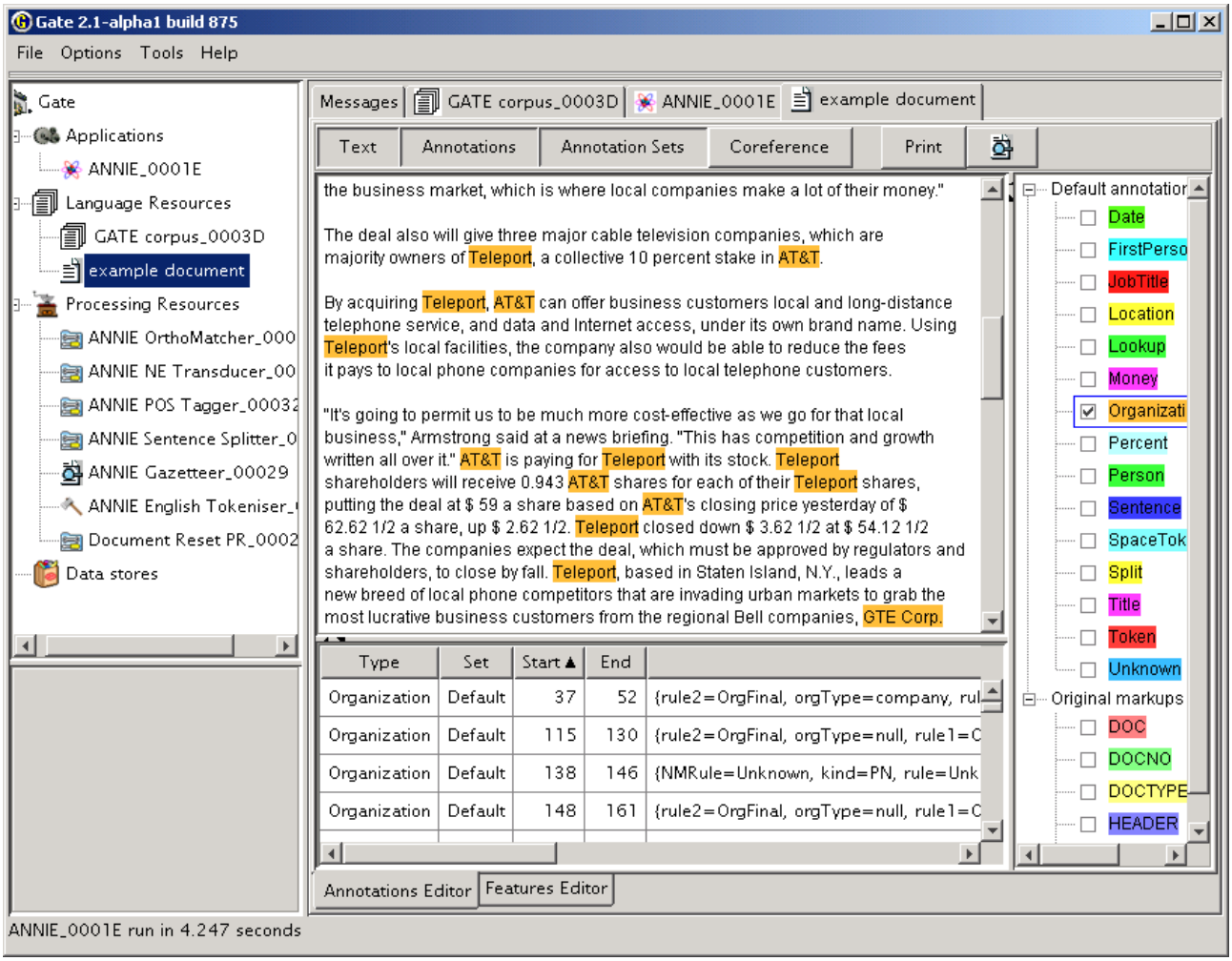

Figure 1: GATE's visual development environment

cations, language resources, and processing resources (i.e., modules). There are also viewers/editors for complex linguistic data like *coreference chains* (Figure 2) and *syntax trees* (Figure 3). New graphical components can be integrated easily, thus allowing lecturers to customise the environment as necessary. The GATE team is also developing new visualisation modules, especially a visual JAPE rule development tool.

2.2 GATE API and Data Model

The central concept that needs to be learned by the students before they start using GATE is the annotation data model, which encodes all linguistic data and is used as input and output for all modules. GATE uses a single unified model of *annotation* - a modified form of the TIPSTER format (Grishman, 1997) which has been made largely compatible with the Atlas format (Bird and Liberman, 1999). Annotations are characterised by a *type* and a set of *features* represented as attribute-value pairs. The annotations are stored in structures called *annotation sets* which constitute independent layers of annotation over the text content. The annotations format is independent of any particular linguistic formalism, in order to enable the use of modules based on different linguistic theories. This generality enables the representation of a wide-variety of linguistic information, ranging from very simple (e.g., tokeniser results) to very com-

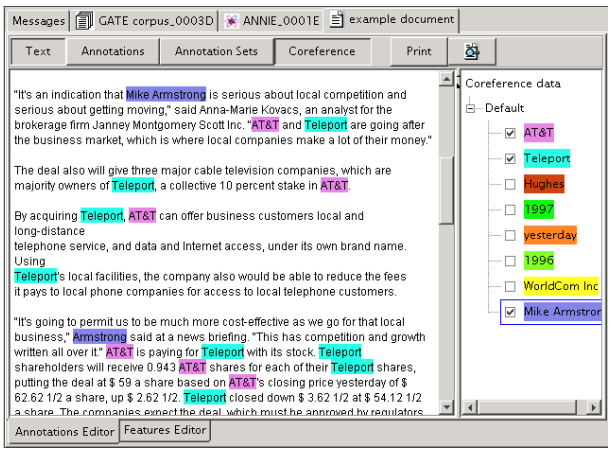

Figure 2: The coreference chains viewer

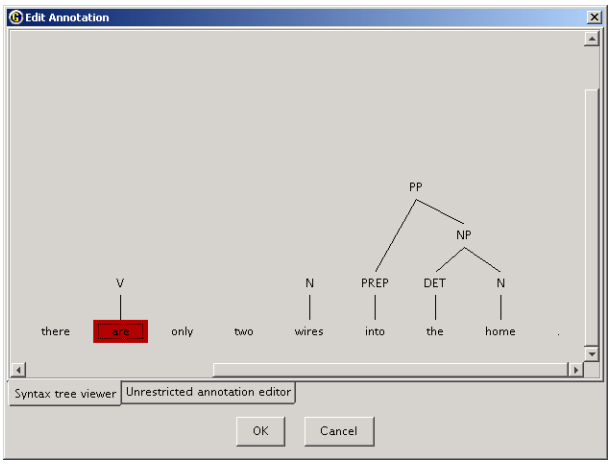

Figure 3: The syntax tree viewer, showing a partial syntax tree for a sentence from a telecom news text

plex (e.g., parse trees and discourse representation: examples in (Saggion et al., 2002)). In addition, the annotation format allows the representation of incomplete linguistic structures, e.g., partial-parsing results. GATE's tree viewing component has been written especially to be able to display such disconnected and incomplete trees.

GATE is implemented in Java, which makes it easier for students to use it, because typically they are already familiar with this language from their programming courses. The GATE API (Application Programming Interface) is fully documented in Javadoc and also examples are given in the comprehensive User Guide (Cunningham et al., 2002b). However, students generally do not need to familiarise themselves with Java and the API at all, because the majority of the modules are based on GATE's JAPE language, so customisation of existing and development of new modules only requires knowledge of JAPE and the annotation model described above.

JAPE is a version of CPSL (Common Pattern Specification Language) (Appelt, 1996) and is used to describe patterns to match and annotations to be created as a result (for further details see (Cunningham et al., 2002b)). Once familiar with GATE's data model, students would not find it difficult to write the JAPE pattern-based rules, because they are effectively regular expressions, which is a concept familiar to most CS students.

An example rule from an existing named entity recognition grammar is:

```
Rule: Company1
Priority: 25
  (
    ({Token.orthography == upperInitial})+
    {Lookup.kind == companyDesignator}
  ):companyMatch
-->
  :companyMatch.NamedEntity =
    {kind = "company", rule = "Company1"}
```

The rule matches a pattern consisting of any kind of word, which starts with an upper-cased letter (recognised by the tokeniser), followed by one of the entries in the gazetteer list for company designators (words which typically indicate companies, such as 'Ltd.' and 'GmBH'). It then annotates this pattern with the entity type "NamedEntity", and gives it a feature "kind" with value company and another feature "rule" with value "Company1". The rule feature is simply used for debugging purposes, so it is clear which particular rule has fired to create the annotation.

The grammars (which are sets of rules) do not need to be compiled by the students, because they are automatically analysed and executed by the JAPE Transducer module, which is a finite-

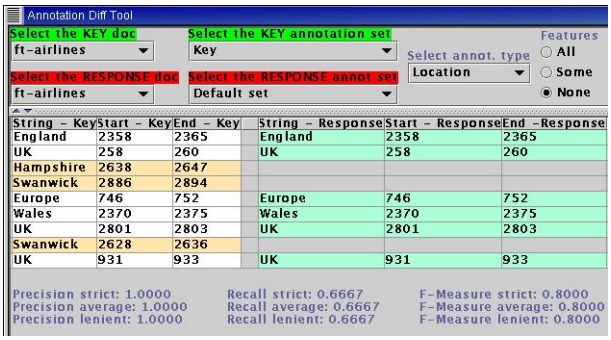

Figure 4: The visual evaluation tool

state transducer over the annotations in the document. Since the grammars are stored in files in a plain text format, they can be edited in any text editor such as Notepad or Vi. The rule development process is performed by the students using GATE's visual environment (see Figure 1) to execute the grammars and visualise the results. The process is actually a cycle, where the students write one or more rules, re-initialise the transducer in the GATE GUI by right-clicking on it, then run it on the test data, check the results, and go back to improving the rules. The evaluation part of this cycle is performed using GATE's visual evaluation tools which also produce precision, recall, and f-measure automatically (see Figure 4).

The advantage of using JAPE for the student assignments is that once learned by the students, it enables them to experiment with a variety of NLP tasks from tokenisation and sentence splitter, to chunking, to template-based information extraction. Because it does not need to be compiled and supports incremental development, JAPE is ideal for rapid prototyping, so students can experiment with alternative ideas.

Students who are doing bigger projects, e.g., a final year project, might want to develop GATE modules which are not based on the finite-state machinery and JAPE. Or the assignment might require the development of more complex grammars in JAPE, in which case they might have to use Java code on the right-hand side of the rule. Since such GATE modules typically only access and manipulate annotations, even then the students would need to learn only that part of GATE's API (i.e., no more than 5 classes). Our experience with two MSc students – Partha Lal and Marin Dimitrov – has shown that they do not have significant problems with using that either.

2.3 Some useful modules

The **tokeniser** splits text into simple tokens, such as numbers, punctuation, symbols, and words of different types (e.g. with an initial capital, all upper case, etc.). The tokeniser does not generally need to be modified for different applications or text types. It currently recognises many types of words, whitespace patterns, numbers, symbols and punctuation and should handle any language from the Indo-European group without modifications. Since it is available as open source, one student assignment could be to modify its rules to cope with other languages or specific problems in a given language. The tokeniser is based on finite-state technology, so the rules are independent from the algorithm that executes them.

The **sentence splitter** is a cascade of finite-state transducers which segments the text into sentences. This module is required for the tagger. Both the splitter and tagger are domain- and application-independent. Again, the splitter grammars can be modified as part of a student project, e.g., to deal with specifically formatted texts.

The **tagger** is a modified version of the Brill tagger, which assigns a part-of-speech tag to each word or symbol. To modify the tagger's behaviour, students will have to re-train it on relevant annotated texts.

The **gazetteer** consists of lists such as cities, organisations, days of the week, etc. It not only consists of entities, but also of names of useful *indicators*, such as typical company designators (e.g. 'Ltd.'), titles, etc. The gazetteer lists are compiled into finite state machines, which annotate the occurrence of the list items in the given document. Students can easily extend the existing lists and add new ones by double-clicking on the Gazetteer processing resource, which brings up the gazetteer editor if it has been installed, or using GATE's Unicode editor.

The **JAPE transducer** is the module that runs JAPE grammars, which could be doing tasks like chunking, named entity recognition, etc. By default, GATE is supplied with an NE transducer which performs named entity recognition for English and a VP Chunker which shows how chunking can be done using JAPE. An even simpler (in terms of grammar rules complexity) and somewhat incomplete NP chunker can be obtained by request from the first author.

The **orthomatcher** is a module, whose primary objective is to perform co-reference, or entity tracking, by recognising relations between entities, based on orthographically matching their names. It also has a secondary role in improving named entity recognition by assigning annotations to previously unclassified names, based on relations with existing entities.

2.4 Support for languages other than English

GATE uses Unicode (Unicode Consortium, 1996) throughout, and has been tested on a variety of Slavic, Germanic, Romance, and Indic languages. The ability to handle Unicode data, along with the separation between data and algorithms, allows students to perform easily even small-scale experiments with porting NLP components to new languages. The graphical development environment supports fully the creation, editing, and visualisation of linguistic data, documents, and corpora in Unicode-supported languages (see (Tablan et al., 2002)). In order to make it easier for foreign students to use the GUI, we are planning to localise its menus, error messages, and buttons which currently are only in English.

2.5 Installation and Programming Languages Support

Since GATE is 100% Java, it can run on any platform that has a Java support. To make it easier to install and maintain, GATE comes with installation wizards for all major platforms. It also allows the creation and use of a site-wide GATE configuration file, so settings need only be specified once and all copies run by the students will have the same configuration and modules available. In addition, GATE allows students to have their own configuration settings, e.g., specify modules which are available only to them. The personal settings override those from GATE's default and site-wide configurations. Students can also easily install GATE on their home computers using the installation program. GATE also allows applications to be saved and moved between computers and platforms, so students can easily work both at home and in the lab and transfer their data and applications between the two.

GATE's graphical environment comes configured by default to save its own state on exit, so students will automatically get their applications, modules, and data restored automatically the next time they load GATE.

Although GATE is Java-based, modules written in other languages can also be integrated and used. For example, Prolog modules are easily executable using the Jasper Java-Prolog linking library. Other programming languages can be used if they support Java Native Interface (JNI).

3 Existing Uses of GATE for Teaching

Postgraduates in locations as diverse as Bulgaria, Copenhagen and Surrey are using the system in order to avoid having to write simple things like sentence splitters from scratch, and to enable visualisation and management of data. For example, Partha Lal at Imperial College is developing a summarisation system based on GATE and ANNIE as a final-year project for an MEng Degree in Computing (http://www.doc.ic.ac.uk/~ pl98/). His site includes the URL of his components and once given this URL, GATE loads his software over the network. Another student project will be discussed in more detail in Section 4.

Our colleagues in the Universities of Edinburgh, UMIST in Manchester, and Sussex (amongst others) have reported using previous versions of the system for teaching, and the University of Stuttgart produced a tutorial in Ger-

man for the same purposes. Educational users of early versions of GATE 2 include Exeter University, Imperial College, Stuttgart University, the University of Edinburgh and others. In order to facilitate the use of GATE as a teaching tool, we have provided a number of tutorials, online demonstrations, and exhaustive documentation on GATE's Web site (http://gate.ac.uk).

4 An Example MSc Project

The goal of this work was to develop a coreference resolution module to be integrated within the named entity recognition system provided with GATE. This required a number of tasks to be performed by the student: *(i)* corpus analysis; *(ii)* implementation and integration; *(iii)* testing and quantitative evaluation.

The student developed a lightweight approach to resolving pronominal coreference for named entities, which was implemented as a GATE module and run after the existing NE modules provided with the framework. This enabled him also to use an existing annotated corpus from an Information Extraction evaluation competition and the GATE evaluation tools to establish how his module compared with results reported in the literature. Finally, the testing process was made simple, thanks to GATE's visualisation facilities, which are already capable of displaying coreference chains in documents.

GATE not only allowed the student to achieve verifiable results quickly, but it also did not incur substantial integration overheads, because it comes with a bootstrap tool which automates the creation of GATE-compliant NLP modules. The steps that need to be followed are:[2]

- use the bootstrap tool to create an empty Java module, then add the implementation to it. A JAVA development environment like JBuilder and VisualCafe can be used for this and the next stages, if the students are familiar with them;
- compile the class, and any others that it uses, into a Java Archive (JAR) file (GATE generates automatically a Makefile too, to facilitate this process);
- write some XML configuration data for the new resource;
- tell GATE the URL of the new JAR and XML files.

[2]For further details and an example see (Cunningham et al., 2002b).

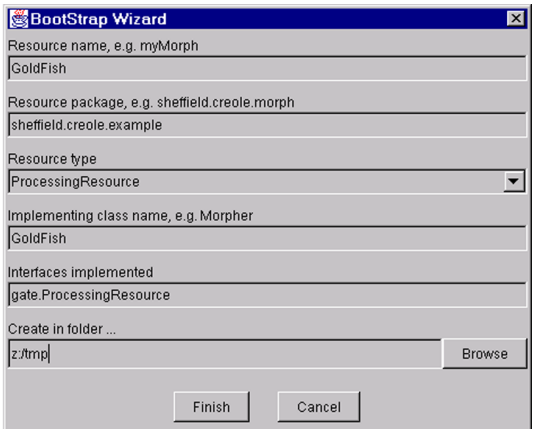

Figure 5: BootStrap Wizard Dialogue

5 Example Topics

Since GATE has been used for a wide range of tasks, it can be used for the teaching of a number of topics. Topics that can be covered in (part of) a course, based on GATE are:

- Language Processing, Language Engineering, and Computational Linguistics: differences, methodologies, problems.
- Architectures, portability, robustness, corpora, and the Web.
- Corpora, annotation, and evaluation: tools and methodologies.
- Basic modules: tokenisation, sentence splitting, gazetteer lookup.
- Part-of-speech tagging.
- Information Extraction: issues, tasks, representing linguistic data in the TIPSTER annotation format, MUC, results achieved.
  - Named Entity Recognition.

  - Coreference Resolution
  - Template Elements and Relations
  - Scenario Templates
- Parsing and chunking
- Document summarisation
- Ontologies and discourse interpretation
- Language generation

While language generation, parsing, summarisation, and discourse interpretation modules are not currently distributed with GATE, they can be obtained by contacting the authors. Modules for text classification and learning algorithms in general are to be developed in the near future. A lecturer willing to contribute any such modules to GATE will be very welcome to do so and will be offered integration support.

6 Example Assignments

The availability of example modules for a variety of NLP tasks allows students to use them as a basis for the development of an entire NLP application, consisting of separate modules built during their course. For example, let us consider two problems: recognising chemical formulae in texts and making an IE system that extracts information from dialogues. Both tasks require students to make changes in a number of existing components and also write some new grammars.

Some example assignments for the chemical formulae recognition follow:

- *tokeniser*: while it will probably work well for the dialogues, the first assignment would be to make modifications to its regular expression grammar to tokenise formulae like H4ClO2 and Al-Li-Ti in a more suitable way.
- *gazetteer*: create new lists containing new useful clues and types of data, e.g., all chemical elements and their abbreviations.
- *named entity recognition*: write a new grammar to be executed by a new JAPE transducer module for the recognition of the chemical formulae.

Some assignments for the dialogue application are:

- *sentence splitter*: modify it so that it splits correctly dialogue texts, by taking into account the speaker information (because dialogues often do not have punctuation). For example:

```
A: Thank you, can I have your full name?
C: Err John Smith
A: Can you also confirm your postcode and
   telephone number for security?
C: Erm it's 111 111 11 11
A: Postcode?
C: AB11 1CD
```

- *corpus annotation and evaluation*: use the default named entity recogniser to bootstrap the manual annotation of the test data for the dialogue application; evaluate the performance of the default NE grammars on the dialogue texts; suggest possible improvements on the basis of the information about missed and incorrect annotations provided by the corpus benchmark tool.
- *named entity recognition*: implement the improvements proposed at the previous step, by changing the default NE grammar rules and/or by introducing rules specific to your dialogue domain.

Finally, some assignments which are not connected to any particular domain or application:

- *chunking*: implement an NP chunker using JAPE. Look at the VP chunker grammars for examples.
- *template-based IE*: experiment with extracting information from the dialogues using templates and JAPE (an example implementation will be provided soon).
- (for a group of students) *building NLP-enabled Web applications*: embed one of the IE applications developed so far into a Web application, which takes a Web page and returns it annotated with the entities. Use http://gate.ac.uk/annie/index.jsp as an example.

In the near future it will be also possible to have assignments on summarisation and generation, but these modules are still under development. It will be possible to demonstrate parsing and discourse interpretation, but because these modules are implemented in Prolog and somewhat difficult to modify, assignments based on them are not recommended. However, other such modules, e.g., those from NLTK (Loper and Bird, 2002), can be used for such assignments.

7 Conclusions

In this paper we have outlined the GATE system and its key features that make it an effective tool for teaching NLP and CL. The main advantage is that GATE is a framework and a graphical development environment which is suitable both for research and teaching, thus making it easier to connect the two, e.g., allow a student to carry out a final-year project which contributes to novel research, carried out by their lecturers. The development environment comes with a comprehensive set of tools, which cover the entire application development cycle. It can be used to provide students with hands-on experience in a wide variety of tasks. Universities willing to use GATE as a teaching tool will benefit from the comprehensive documentation, several tutorials, and online demonstrations.

References

D.E. Appelt. 1996. The Common Pattern Specification Language. Technical report, SRI International, Artificial Intelligence Center.

S. Bird and M. Liberman. 1999. A Formal Framework for Linguistic Annotation. Technical Report MS-CIS-99-01, Department of Computer and Information Science, University of Pennsylvania. `http://xxx.lanl.gov/abs/cs.CL/9903003`.

H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. 2002a. 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*.

H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, and C. Ursu. 2002b. *The GATE User Guide*. `http://gate.ac.uk/`.

H. Cunningham. 2002. GATE, a General Architecture for Text Engineering. *Computers and the Humanities*, 36:223–254.

R. Grishman. 1997. TIPSTER Architecture Design Document Version 2.3. Technical report, DARPA. `http://www.itl.nist.gov/-div894/894.02/related_projects/tipster/`.

E. Loper and S. Bird. 2002. NLTK: The Natural Language Toolkit. In *ACL Workshop on Effective Tools and Methodologies in Teaching NLP*.

D. Maynard, H. Cunningham, K. Bontcheva, R. Catizone, George Demetriou, Robert Gaizauskas, Oana Hamza, Mark Hepple, Patrick Herring, Brian Mitchell, Michael Oakes, Wim Peters, Andrea Setzer, Mark Stevenson, Valentin Tablan, Christian Ursu, and Yorick Wilks. 2000. A Survey of Uses of GATE. Technical Report CS–00–06, Department of Computer Science, University of Sheffield.

D. Maynard, V. Tablan, C. Ursu, H. Cunningham, and Y. Wilks. 2001. Named Entity Recognition from Diverse Text Types. In *Recent Advances in Natural Language Processing 2001 Conference*, Tzigov Chark, Bulgaria.

D. Maynard, V. Tablan, H. Cunningham, C. Ursu, H. Saggion, K. Bontcheva, and Y. Wilks. 2002. Architectural elements of language engineering robustness. *Journal of Natural Language Engineering – Special Issue on Robust Methods in Analysis of Natural Language Data*. forthcoming.

N.F. Noy, M. Sintek, S. Decker, M. Crubzy, R.W. Fergerson, and M.A. Musen. 2001. Creating Semantic Web Contents with Protégé-2000. *IEEE Intelligent Systems*, 16(2):60–71.

H. Saggion, H. Cunningham, K. Bontcheva, D. Maynard, C. Ursu, O. Hamza, and Y. Wilks. 2002. Access to Multimedia Information through Multisource and Multilanguage Information Extraction. In *7th Workshop on Applications of Natural Language to Information Systems (NLDB 2002)*, Stockholm, Sweden.

V. Tablan, C. Ursu, K. Bontcheva, H. Cunningham, D. Maynard, O. Hamza, Tony McEnery, Paul Baker, and Mark Leisher. 2002. A unicode-based environment for creation and use of language resources. In *Proceedings of 3rd Language Resources and Evaluation Conference*. forthcoming.

Unicode Consortium. 1996. *The Unicode Standard, Version 2.0*. Addison-Wesley, Reading, MA.