GATE: an Architecture for Development of Robust HLT Applications

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

Abstract

In this paper we present GATE, a framework and graphical development environment which enables users to develop and deploy language engineering components and resources in a robust fashion. The GATE architecture has enabled us not only to develop a number of successful applications for various language processing tasks (such as Information Extraction), but also to build and annotate corpora and carry out evaluations on the applications generated. The framework can be used to develop applications and resources in multiple languages, based on its thorough Unicode support.

1 Introduction

Producing robust components to process human language as part of applications software requires attention to the engineering aspects of their construction. This paper reports work on GATE[1], an infrastructure for language processing software development that contributes on several fronts to this type of predictability:

- 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.
- Automating measurement of performance of language processing components.
- Reducing integration overheads by providing standard mechanisms for components to communicate data about language, and using open standards such as Java and XML as the underlying platform.
- Providing a baseline set of language processing components that can be extended and/or replaced by users as required.

The rest of the paper is structured as follows. We first describe the GATE architecture in Section 2, and then give details of some of the applications we have built using GATE in Section 3. Section 4 describes the processing resources available within GATE, while Section 5 describes the language resources. In Section 6 we discuss the mechanisms for evaluation. Finally, Section 7 puts this work in the context of some previous work and Section 8 discusses future directions.

2 A framework for robust tools and applications

GATE (Cunningham, 2002) is an architecture, a framework and a development environment for LE (Language Engineering)[2]. As an *architecture*, it defines the organisation of an LE system and the assignment of responsibilities to dif-

[1]This work has been supported by the Engineering and Physical Sciences Research Council (EPSRC) under grants GR/K25267 and GR/M31699, and by several smaller grants.

[2]GATE is freely available for download from http://gate.ac.uk.

Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, July 2002, pp. 168-175.

ferent components, and ensures that the component interactions satisfy the system requirements. As a *framework*, it provides a reusable design for an LE software system and a set of prefabricated software building blocks that language engineers can use, extend and customise for their specific needs. As a *development environment*, it helps its users to minimise the time they spend building new LE systems or modifying existing ones, by aiding overall development and providing a debugging mechanism for new modules. Because GATE has a component-based model, this allows for easy coupling and decoupling of the processors, thereby facilitating comparison of alternative configurations of the system or different implementations of the same module (e.g., different parsers). The availability of tools for easy visualisation of data at each point during the development process aids immediate interpretation of the results.

The GATE framework comprises a core library (analogous to a bus backplane) and a set of reusable LE modules. The framework implements the architecture and provides (amongst other things) facilities for processing and visualising resources, including representation, import and export of data.

The reusable modules provided with the backplane are able to perform basic language processing tasks such as POS tagging and semantic tagging. This eliminates the need for users to keep recreating the same kind of resources, and provides a good starting point for new applications. The modules are described in more detail in Section 4.

Applications developed within GATE can be deployed outside its Graphical User Interface (GUI), using programmatic access via the GATE API (see `http://gate.ac.uk`). In addition, the reusable modules, the document and annotation model, and the visualisation components can all be used independently of the development environment.

GATE components may be implemented by a variety of programming languages and databases, but in each case they are represented to the system as a Java class. This class may simply call the underlying program or provide an access layer to a database; alternatively it may implement the whole component.

In the rest of this section, we show how the GATE infrastructure takes care of the resource discovery, loading, and execution, and briefly discuss data storage and visualisation.

2.1 Algorithms + data + GUI = applications

The title expresses succinctly the distinction made in GATE between data, algorithms, and ways of visualising them. In other words, GATE components are one of three types:

- **LanguageResources** (LRs) represent entities such as lexicons, corpora or ontologies;
- **ProcessingResources** (PRs) represent entities that are primarily algorithmic, such as parsers, generators or ngram modellers;
- **VisualResources** (VRs) represent visualisation and editing components that participate in GUIs.

These resources can be local to the user's machine or remote (available via HTTP), and all can be extended by users without modification to GATE itself.

One of the main advantages of separating the algorithms from the data they require is that the two can be developed independently by language engineers with different types of expertise, e.g. programmers and linguists. Similarly, separating data from its visualisation allows users to develop alternative visual resources, while still using a language resource provided by GATE.

Collectively, all resources are known as CREOLE (a Collection of REusable Objects for Language Engineering), and are declared in a repository XML file, which describes their name, implementing class, parameters, icons, etc. This repository is used by the framework to discover and load available resources.

A parameters tag describes the parameters which each resource needs when created or executed. Parameters can be optional, e.g. if a document list is provided when the corpus is

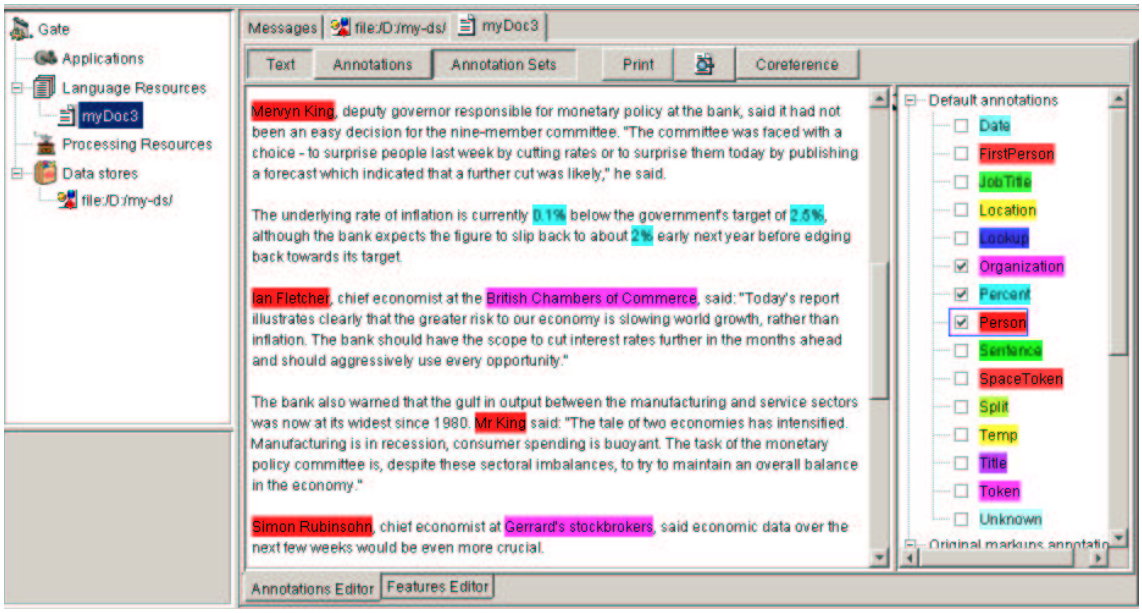

Figure 1: GATE's document viewer/editor

constructed, it will be populated automatically with these documents.

When an application is developed within GATE's graphical environment, the user chooses which processing resources go into it (e.g. tokeniser, POS tagger), in what order they will be executed, and on which data (e.g. document or corpus).

The execution parameters of each resource are also set there, e.g. a loaded document is given as a parameter to each PR. When the application is run, the modules will be executed in the specified order on the given document. The results can be viewed in the document viewer/editor (see Figure 1).

2.2 Data representation and handling

GATE supports a variety of formats including XML, RTF, HTML, SGML, email and plain text. In all cases, when a document is created/opened in GATE, the format is analysed and converted into a single unified model of *annotation*. The annotation format is a modified form of the TIPSTER format (Grishman, 1997) which has been made largely compatible with the Atlas format (Bird et al., 2000), and uses the now standard mechanism of 'stand-off markup' (Thompson and McKelvie, 1997). The annotations associated with each document are a structure central to GATE, because they encode the language data read and produced by each processing module.

The GATE framework also provides persistent storage of language resources. It currently offers three storage mechanisms: one uses relational databases (e.g. Oracle) and the other two are file- based, using Java serialisation or an XML-based internal format. GATE documents can also be exported back to their original format (e.g. SGML/XML for the British National Corpus (BNC)) and the user can choose whether some additional annotations (e.g. named entity information) are added to it or not.

To summarise, the existence of a unified data structure ensures a smooth communication between components, while the provision of import and export capabilities makes communication with the outside world simple.

2.3 Multilingual processing

In recent years, the emphasis on multilinguality has grown, and important advances have been

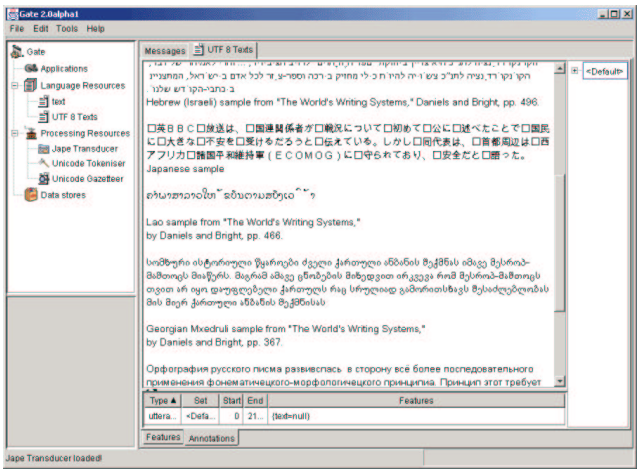

Figure 2: Unicode text in Gate2

witnessed on the software scene with the emergence of Unicode as a universal standard for representing textual data.

GATE supports multilingual data processing using Unicode as its default text encoding. It also provides a means of entering text in various languages, using virtual keyboards where the language is not supported by the underlying operating platform. (Note that although Java represents characters as Unicode, it doesn't support input in many of the languages covered by Unicode.) Currently 28 languages are supported, and more are planned for future releases. Because GATE is an open architecture, new virtual keyboards can be defined by the users and added to the system as needed. For displaying the text, GATE relies on the rendering facilities offered by the Java implementation for the platform it runs on. Figure 2 gives an example of text in various languages displayed by GATE.

The ability to handle Unicode data, along with the separation between data and implementation, allows LE systems based on GATE to be ported to new languages with no additional overhead apart from the development of the resources needed for the specific language. These facilities have been developed as part of the EMILLE project (McEnery et al., 2000), which focuses on the construction a 63 million word electronic corpus of South Asian languages.

3 Applications

One of GATE's strengths is that it is flexible and robust enough to enable the development of a wide range of applications within its framework. In this section, we describe briefly some of the NLP applications we have developed using the GATE architecture.

3.1 MUSE

The MUSE system (Maynard et al., 2001) is a multi-purpose Named Entity recognition system which is capable of processing texts from widely different domains and genres, thereby aiming to reduce the need for costly and time-consuming adaptation of existing resources to new applications and domains. The system aims to identify the parameters relevant to the creation of a name recognition system across different types of variability such as changes in domain, genre and media. For example, less formal texts may not follow standard capitalisation, punctuation and spelling formats, which can be a problem for many generic NE systems. Current evaluations with this system average around 93% precision and 95% recall across a variety of text types.

3.2 ACE

The MUSE system has also been adapted to take part in the current ACE (Automatic Content Extraction) program run by NIST. This requires systems to perform recognition and tracking tasks of named, nominal and pronominal entities and their mentions across three types of clean news text (newswire, broadcast news and newspaper) and two types of degraded news text (OCR output and ASR output).

3.3 MUMIS

The MUMIS (MUltiMedia Indexing and Searching environment) system uses Information Extraction components developed within GATE to produce formal annotations about essential events in football video programme material. This IE system comprises versions of the tokenisation, sentence detection, POS-tagging, and semantic tagging modules developed as part of GATE's standard resources, but also includes

morphological analysis, full syntactic parsing and discourse interpretation modules, thereby enabling the production of annotations over text encoding structural, lexical, syntactic and semantic information. The semantic tagging module currently achieves around 91% precision and 76% recall, a significant improvement on a baseline named entity recognition system evaluated against it.

4 Processing Resources

Provided with GATE is a set of reusable processing resources for common NLP tasks. (None of them are definitive, and the user can replace and/or extend them as necessary.) These are packaged together to form ANNIE, A Nearly-New IE system, but can also be used individually or coupled together with new modules in order to create new applications. For example, many other NLP tasks might require a sentence splitter and POS tagger, but would not necessarily require resources more specific to IE tasks such as a named entity transducer. The system is in use for a variety of IE and other tasks, sometimes in combination with other sets of application-specific modules.

ANNIE consists of the following main processing resources: tokeniser, sentence splitter, POS tagger, gazetteer, finite state transducer (based on GATE's built-in regular expressions over annotations language (Cunningham et al., 2002)), orthomatcher and coreference resolver. The resources communicate via GATE's annotation API, which is a directed graph of arcs bearing arbitrary feature/value data, and nodes rooting this data into document content (in this case text).

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 aim is to limit the work of the tokeniser to maximise efficiency, and enable greater flexibility by placing the burden of analysis on the grammars. This means that the tokeniser does not need to be modified for different applications or text types.

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.

The **tagger** is a modified version of the Brill tagger, which produces a part-of-speech tag as an annotation on each word or symbol. Neither the splitter nor the tagger are a mandatory part of the NE system, but the annotations they produce can be used by the grammar (described below), in order to increase its power and coverage.

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 can match text tokens.

The **semantic tagger** consists of handcrafted rules written in the JAPE (Java Annotations Pattern Engine) language (Cunningham et al., 2002), which describe patterns to match and annotations to be created as a result. JAPE is a version of CPSL (Common Pattern Specification Language) (Appelt, 1996), which provides finite state transduction over annotations based on regular expressions. A JAPE grammar consists of a set of phases, each of which consists of a set of pattern/action rules, and which run sequentially. Patterns can be specified by describing a specific text string, or annotations previously created by modules such as the tokeniser, gazetteer, or document format analysis. Rule prioritisation (if activated) prevents multiple assignment of annotations to the same text string.

The **orthomatcher** is another optional module for the IE system. Its primary objective is to perform co-reference, or entity tracking, by recognising relations between entities. It also has a secondary role in improving named entity recognition by assigning annotations to previously unclassified names, based on relations with existing entities.

The **coreferencer** finds identity relations between entities in the text. For more details see (Dimitrov, 2002).

4.1 Implementation

The implementation of the processing resources is centred on robustness, usability and the clear distinction between declarative data representations and finite state algorithms. The behaviour of all the processors is completely controlled by external resources such as grammars or rule sets, which makes them easily modifiable by users who do not need to be familiar with programming languages.

The fact that all processing resources use finite-state transducer technology makes them quite performant in terms of execution times. Our initial experiments show that the full named entity recognition system is capable of processing around 2.5KB/s on a PIII 450 with 256 MB RAM (independently of the size of the input file; the processing requirement is linear in relation to the text size). Scalability was tested by running the ANNIE modules over a randomly chosen part of the British National Corpus (10% of all documents), which contained documents of up to 17MB in size.

5 Language Resource Creation

Since many NLP algorithms require annotated corpora for training, GATE's development environment provides easy-to-use and extendable facilities for text annotation. In order to test their usability in practice, we used these facilities to build corpora of named entity annotated texts for the MUSE, ACE, and MUMIS applications.

The annotation can be done manually by the user or semi-automatically by running some processing resources over the corpus and then correcting/adding new annotations manually. Depending on the information that needs to be annotated, some ANNIE modules can be used or adapted to bootstrap the corpus annotation task. For example, users from the humanities created a gazetteer list with 18th century place names in London, which when supplied to the ANNIE gazetteer, allows the automatic annotation of place information in a large collection of 18th century court reports from the Old Bailey in London.

Since manual annotation is a difficult and error-prone task, GATE tries to make it simple to use and yet keep it flexible. To add a new annotation, one selects the text with the mouse (e.g., "Mr. Clever") and then clicks on the desired annotation type (e.g., Person), which is shown in the list of types on the right-hand-side of the document viewer (see Figure 1). If however the desired annotation type does not already appear there or the user wants to associate more detailed information with the annotation (not just its type), then an annotation editing dialogue can be used.

6 Evaluation

A vital part of any language engineering application is the evaluation of its performance, and a development environment for this purpose would not be complete without some mechanisms for its measurement in a large number of test cases. GATE contains two such mechanisms: an evaluation tool (AnnotationDiff) which enables automated performance measurement and visualisation of the results, and a benchmarking tool, which enables the tracking of a system's progress and regression testing.

6.1 The AnnotationDiff Tool

Gate's AnnotationDiff tool enables two sets of annotations on a document to be compared, in order to either compare a system-annotated text with a reference (hand-annotated) text, or to compare the output of two different versions of the system (or two different systems). For each annotation type, figures are generated for precision, recall, F-measure and false positives.

The AnnotationDiff viewer displays the two sets of annotations, marked with different colours (similar to 'visual diff' implementations such as in the MKS Toolkit or TkDiff). Annotations in the key set have two possible colours depending on their state: white for annotations which have a compatible (or partially compatible) annotation in the response set, and orange for annotations which are missing in the response set. Annotations in the response set have three possible colours: green if they are compatible with the key annotation, blue if they

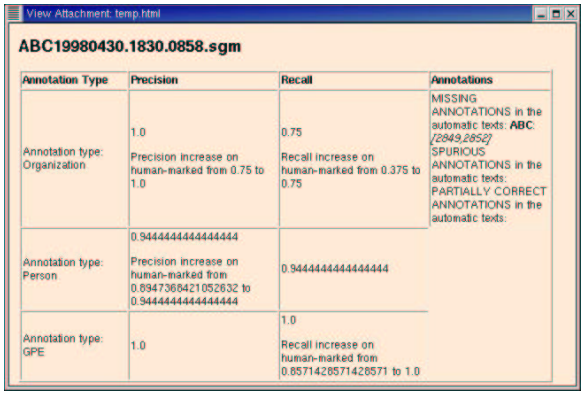

Figure 3: Fragment of results from benchmark tool

are partially compatible, and red if they are spurious. In the viewer, two annotations will be positioned on the same row if they are co-extensive, and on different rows if not.

6.2 Benchmarking tool

GATE's benchmarking tool differs from the AnnotationDiff in that it enables evaluation to be carried out over a whole corpus rather than a single document. It also enables tracking of the system's performance over time.

The tool requires a clean version of a corpus (with no annotations) and an annotated reference corpus. First of all, the tool is run in generation mode to produce a set of texts annotated by the system. These texts are stored for future use. The tool can then be run in three ways:

1. Comparing the annotated set with the reference set;
2. Comparing the annotated set with the set produced by a more recent version of the system resources (the latest set);
3. Comparing the latest set with the reference set.

In each case, performance statistics will be provided for each text in the set, and overall statistics for the entire set, in comparison with the reference set. In case 2, information is also provided about whether the figures have increased or decreased in comparison with the annotated set. The annotated set can be updated at any time by rerunning the tool in generation mode with the latest version of the system resources. Furthermore, the system can be run in verbose mode, where for each figure below a certain threshold (set by the user), the non-coextensive annotations (and their corresponding text) will be displayed. The output of the tool is written to an HTML file in tabular form, as shown in Figure 3.

Current evaluations for the MUSE NE system are producing average figures of 90-95% Precision and Recall on a selection of different text types (spoken transcriptions, emails etc.). The default ANNIE system produces figures of between 80-90% Precision and Recall on news texts. This figure is lower than for the MUSE system, because the resources have not been tuned to a specific text type or application, but are intended to be adapted as necessary. Work on resolution of anaphora is currently averaging 63% Precision and 45% Recall, although this work is still very much in progress, and we expect these figures to improve in the near future.

7 Related Work

GATE draws from a large pool of previous work on infrastructures, architectures and development environments for representing and processing language resources, corpora, and annotations. Due to space limitations here we will discuss only a small subset. For a detailed review and its use for deriving the desiderata for this architecture see (Cunningham, 2000).

Work on standard ways to deal with XML data is relevant here, such as the LT XML work at Edinburgh (Thompson and McKelvie, 1997), as is work on managing collections of documents and their formats, e.g. (Brugman et al., 1998; Grishman, 1997; Zajac, 1998). We have also drawn from work on representing information about text and speech, e.g. (Brugman et al., 1998; Mikheev and Finch, 1997; Zajac, 1998; Young et al., 1999), as well as annotation standards, such as the ATLAS project (an architecture for linguistic annotation) at LDC (Bird et

al., 2000). Our approach is also related to work on user interfaces to architectural facilities such as development environments, e.g. (Brugman et al., 1998) and to work on comparing different versions of information, e.g. (Sparck-Jones and Galliers, 1996; Paggio, 1998).

This work is particularly novel in that it addresses the complete range of issues in NLP application development in a flexible and extensible way, rather than focusing just on some particular aspect of the development process. In addition, it promotes robustness, re-usability, and scalability as important principles that help with the construction of practical NLP systems.

8 Conclusions

In this paper we have described an infrastructure for language engineering software which aims to assist the develeopment of robust tools and resources for NLP.

One future direction is the integration of processing resources which learn in the background while the user is annotating corpora in GATE's visual environment. Currently, statistical models can be integrated but need to be trained separately. We are also extending the system to handle language generation modules, in order to enable the construction of applications which require language production in addition to analysis, e.g. intelligent report generation from IE data.

References

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

S. Bird, D. Day, J. Garofolo, J. Henderson, C. Laprun, and M. Liberman. 2000. ATLAS: A flexible and extensible architecture for linguistic annotation. In *Proceedings of the Second International Conference on Language Resources and Evaluation*, Athens.

H. Brugman, H.G. Russel, and P. Wittenburg. 1998. An infrastructure for collaboratively building and using multimedia corpora in the humaniora. In *Proceedings of the ED-MEDIA/ED-TELECOM Conference*, Freiburg.

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

H. Cunningham. 2000. *Software Architecture for Language Engineering*. Ph.D. thesis, University of Sheffield. `http://gate.ac.uk/sale/thesis/`.

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

M. Dimitrov. 2002. *A Light-weight Approach to Coreference Resolution for Named Entities in Text*. MSc Thesis, University of Sofia, Bulgaria. `http://www.ontotext.com/ie/thesis-m.pdf`.

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

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.

A.M. McEnery, P. Baker, R. Gaizauskas, and H. Cunningham. 2000. EMILLE: Building a Corpus of South Asian Languages. *Vivek, A Quarterly in Artificial Intelligence*, 13(3):23–32.

A. Mikheev and S. Finch. 1997. A Workbench for Finding Structure in Text. In *Fifth Conference on Applied NLP (ANLP-97)*, Washington, DC.

P. Paggio. 1998. Validating the TEMAA LE evalutation methodology: a case study on Danish spelling checkers. *Journal of Natural Language Engineering*, 4(3):211–228.

K. Sparck-Jones and J. Galliers. 1996. *Evaluating Natural Language Processing Systems*. Springer, Berlin.

H. Thompson and D. McKelvie. 1997. Hyperlink semantics for standoff markup of read-only documents. In *Proceedings of SGML Europe'97*, Barcelona.

S. Young, D. Kershaw, J. Odell, D. Ollason, V. Valtchev, and P. Woodland. 1999. *The HTK Book (Version 2.2)*. Entropic Ltd., Cambridge. `ftp://ftp.entropic.com/pub/htk/`.

R. Zajac. 1998. Reuse and Integration of NLP Components in the Calypso Architecture. In *Workshop on Distributing and Accessing Linguistic Resources*, pages 34–40, Granada, Spain. `http://www.dcs.shef.ac.uk/~hamish/dalr/`.