Natural Language Engineering **1** (1): 000–000. Printed in the United Kingdom
© 1998 Cambridge University Press

Evolving GATE to Meet New Challenges in Language Engineering†

KALINA BONTCHEVA, VALENTIN TABLAN,
DIANA MAYNARD, HAMISH CUNNINGHAM
Department of Computer Science
University of Sheffield
Sheffield, S1 4DP, UK
{kalina,valyt,diana,hamish}@dcs.shef.ac.uk

(*Received 30 July 2003; revised 12 February 2004*)

Abstract

In this paper we present recent work on GATE, a widely-used framework and graphical development environment for creating and deploying Language Engineering components and resources in a robust fashion. The GATE architecture has facilitated the development of a number of successful applications for various language processing tasks (such as Information Extraction, dialogue and summarisation), the building and annotation of corpora and the quantitative evaluations of LE applications. The focus of this paper is on recent developments in response to new challenges in Language Engineering: Semantic Web, integration with Information Retrieval and data mining, and the need for machine learning support.

1 Introduction

Natural Language Engineering (LE) is a rapidly changing field, driven by the push of internal technological advances on the one hand, and by the pull of new application contexts on the other. These drivers include:

- the project of the Semantic Web, adding a machine-tractable semantic layer to the existing World Wide Web (Fensel *et al.* 02; Davies *et al.* 02);
- the growing need for easily portable multilingual LE applications in face of increasing integration in an expanding Europe;[1]

† Work on GATE has been supported by the Engineering and Physical Sciences Research Council (EPSRC) under grants GR/K25267 (Large-Scale Information Extraction), GR/M31699 (GATE 2), GR/N15764/01 (AKT), EMILLE, and by several smaller grants. We would like to thank the numerous people who have contributed to this work. Particularly Marin Dimitrov, Cristian Ursu, Oana Hamza, Borislav Popov, Atanas Kiryakov, Robert Gaizauskas, and Yorick Wilks for their contributions to GATE v2; and finally to Donia Scott and the three anonymous reviewers of this paper.

[1] And perhaps in view of decreasing integration and increasing strife internationally.

- the continuing growth in methods combining information theory and machine learning with symbolic and linguistic systems;
- the emergence of applications combining Information Extraction (IE) and Information Retrieval (IR) methods in very large scale contexts.[2]

These changes all create new challenges to Language Engineering architectures and infrastructures, a field which we may call Software Architecture for Language Engineering, or SALE. SALE systems aim to provide engineering support for research and development of language processing software, in a way similar to the support offered to programmers by software development environments like Visual C++.

GATE (Cunningham *et al.* 97; Cunningham 02; Cunningham *et al.* 02), a General Architecture for Text Engineering, was first released in 1996, then completely redesigned, re-written, and re-released in early 2002. The system is now widely-used (by thousands of people at hundreds of sites) and is a relatively comprehensive infrastructure for language processing software development. Key features of GATE are:

- Component-based development reduces the systems integration overhead in collaborative research.
- Open source, documented and supported software enhances the repeatability of experiments.
- Automatic performance measurement of the LE components promotes quantitative comparative evaluation.
- Distinction between low-level tasks such as data storage, data visualisation, discovery and loading of components and the high-level language processing tasks.
- Clean separation between between data structures and algorithms that process human language.
- Consistent use of standard mechanisms for components to communicate data about language, and use of open standards such as Unicode and XML.
- Insulation from indiosyncratic data formats – GATE performs automatic format conversion and enables uniform access to linguistic data.
- Provision of a baseline set of LE components that can be extended and/or replaced by users as required.

The focus of this paper is on the recent novel aspects of the infrastructure, that have been introduced in GATE version 2 (GATE v2) – the majority in response to the recent challenges discussed above. Some of these new elements were added to GATE because of problems identified with its first version (Cunningham 02):

- Non-extensibility of the tool model (extensibility was only available for processing modules, not for viewers, or other tools).
- Problematic multilingual support.
- Need for a pattern-matching engine to allow rapid prototyping and to simplify the development of LE components.

[2] See for example IBM's WebFountain product.

- Need for better support for lexical resources, e.g., lexicons, thesauri, and their use within the processing components.
- Need to extend towards new data formats, e.g., XML, DAML+OIL, that did not exist at the time when GATE version 1 (GATE v1) was implemented.
- Inappropriate processing model for language generation systems.
- Insufficient portability with respect to operating systems.

The rest of the paper is structured as follows. Section 2 introduces the GATE framework with a focus on important new elements of the architecture, such as support for distributed resources, application execution strategies, and the JAPE finite-state processing engine. Section 3 introduces the reusable LE components in GATE v2, including machine learning and ontology-based modules. Section 4 discusses the new types of language resources: ontologies and lexicons, their representation, editing, and use. Section 5 presents the new extendable model for reuse and customisation of viewers, editors, and other interface tools. A number of recent LE applications is then discussed in Section 6, followed by a review of related work (Section 7). Finally, we conclude by revisiting some recent challenges for language engineering and outline ongoing and future work on GATE in these directions.

2 GATE: A framework for robust tools and applications

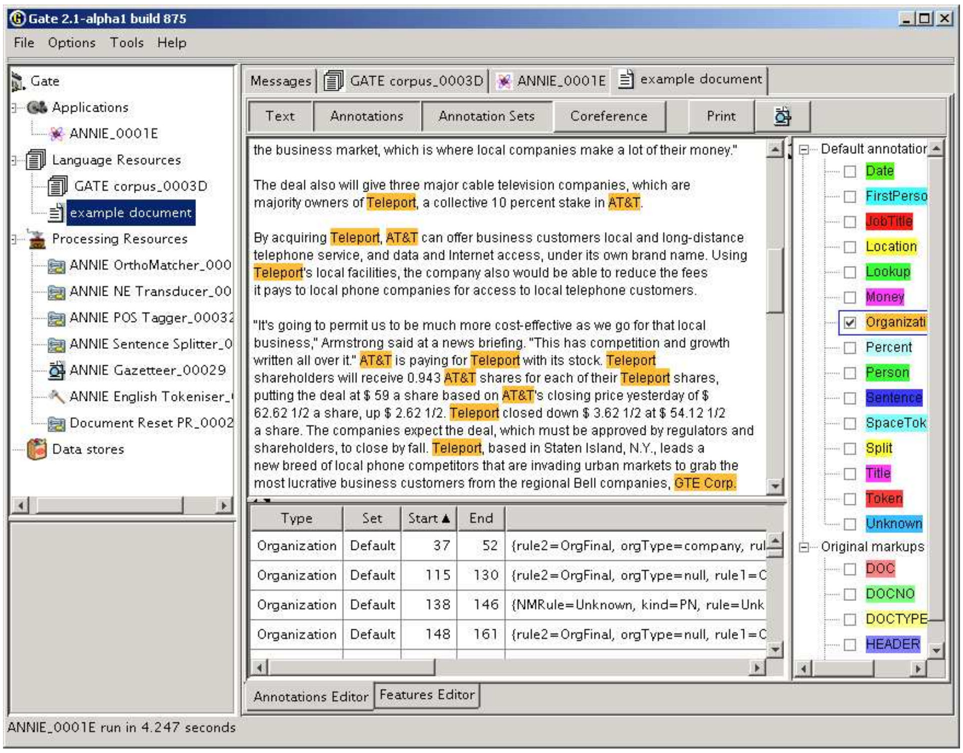
GATE[3] is an architecture, a framework and a development environment for LE (Language Engineering) applications. As an *architecture*, it defines the organisation of an LE system and the assignment of responsibilities to different components. As a *framework*, it provides reusable implementations for LE components 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 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.

Applications developed within GATE can be deployed outside its Graphical User Interface (GUI), using programmatic access via the GATE Application Programming Interface (API).[4] In addition, the reusable modules, the document and annotation model, and the visualisation components can all be used independently of the development environment.

GATE is engineered to a high standard and supports efficient and robust text

[3] GATE v2 is freely available for download from `http://gate.ac.uk`. It is implemented in Java and runs on a wide range of platforms.

[4] See `http://www.gate.ac.uk/releases/gate-2.2-build1350--ALL/doc/javadoc/index.html`



the business market, which is where local companies make a lot of their money."

The deal also will give three major cable television companies, which are majority owners of Teleport, a collective 10 percent stake in AT&T.

By acquiring Teleport, AT&T can offer business customers local and long-distance telephone service, and data and Internet access, under its own brand name. Using Teleport's local facilities, the company also would be able to reduce the fees it pays to local phone companies for access to local telephone customers.

"It's going to permit us to be much more cost-effective as we go for that local business," Armstrong said at a news briefing. "This has competition and growth written all over it." AT&T is paying for Teleport with its stock. Teleport shareholders will receive 0.943 AT&T shares for each of their Teleport shares, putting the deal at $ 59 a share based on AT&T's closing price yesterday of $ 62.62 1/2 a share, up $ 2.62 1/2. Teleport closed down $ 3.62 1/2 at $ 54.12 1/2 a share. The companies expect the deal, which must be approved by regulators and shareholders, to close by fall. Teleport, based in Staten Island, N.Y., leads a new breed of local phone competitors that are invading urban markets to grab the most lucrative business customers from the regional Bell companies, GTE Corp.

Type	Set	Start ▲	End	
Organization	Default	37	52	{rule2=OrgFinal, orgType=company, rul
Organization	Default	115	130	{rule2=OrgFinal, orgType=null, rule1=C
Organization	Default	138	146	{NMRule=Unknown, kind=PN, rule=Unk
Organization	Default	148	161	{rule2=OrgFinal, orgType=null, rule1=C

Fig. 1. GATE's document viewer/editor

processing. It is tested extensively, including regression testing, and frequent performance optimization. GATE v2 has proved capable of processing gigabytes of text and thousands of documents (most recently tested on the British National Corpus (Burnard 95)). The system is supported by extensive documentation, online tutorials and an established mailing list.

In the rest of this section, we provide an overview of the GATE architecture, discuss the new annotation graph based data representation, and introduce important new elements, such as multilingual support, distributed resources, execution strategies, and the JAPE finite-state processing engine.

2.1 An Overview of the architecture

The GATE architecture distinguishes between data, algorithms, and their visualisation.[5] Following the terminology established in version 1, GATE components are one of three types:

[5] Analogous to the model/controller/view architecture common in GUI toolkits.

- **Language Resources** (LRs) represent entities such as lexicons (e.g. WordNet), corpora or ontologies;[6]
- **Processing Resources** (PRs) represent entities that are primarily algorithmic, such as parsers, generators or n-gram modellers;
- **Visual Resources** (VRs) – added in version 2 – represent visualisation and editing components that participate in GUIs.

These resources can be local to the user's machine or remote (available over the (Inter)net), 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. programming and linguistics. 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 described in XML configuration files, which declare their name, implementing class, parameters, icons, etc. This component metadata is used by the framework to discover and load available resources.

Unlike version 1, GATE v2 offers comprehensive multilingual support 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 (McEnery *et al.* 00; Tablan *et al.* 02).

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 application results can be viewed in the document viewer/editor (see Figure 1). Applications can be saved, reloaded, and embedded in other systems.

2.2 Data Representation and Handling

GATE supports a variety of formats including plain text, HTML, XML, RTF, and SGML in order to enable the processing of a wide range of documents and improve interoperability with other LE infratructures. In all cases, when a document is 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 97) (originally used in GATE v1), which has been made largely compatible with the Atlas format (Bird *et al.* 00b), and is isomorphic with "standoff markup" as widely adopted by the SGML/XML community (Ide *et al.* 00). The annotations associated with each document are a structure central to GATE,

[6] Ontologies are often not considered as describing language and, therefore, their classification as LRs could be somewhat un-intuitive. Probably a more appropriate term will be data resources, but we adhere to the original terminology adopted in GATE v1. Fundamentally, GATE treats all declarative data resources like lexicons, ontologies, and documents in the same fashion, i.e., they can be provided as inputs or parameters to PRs and are visualised and edited in VRs.

because they encode the data read and produced by all processing modules, as well as the formatting information originally present in the documents.

Annotations are organised in *annotation graphs*, where the vertices are anchored in the document content. Annotations are the arcs in the graph; they have a start node and an end node, an identifier, a type and a set of features. Nodes have pointers into the content, e.g. character offsets for text, milliseconds for audio-visual content. There can be more than one annotation graph associated with each document. The support for separate annotation graphs (called families in IBM's TEXTRACT architecture (Neff *et al.*)) is needed to allow separation of information into different categories.

GATE has a single model for encoding information that describes documents, collections of documents (corpora), and annotations on documents, based on attribute name/attribute value pairs, called *features*. Attribute names are strings; values can be any object. Figure 1 shows annotations of type `Organization` and their features: `orgType` for the type of organisation, `rule` specifying which rule produced this annotation, etc. At present GATE uses features only as a representation formalism and does not provide operations like unification.

In order to facilitate the interpretation and validation of features associated with each annotation type, GATE v2 provides annotation schemas, which are encoded in the XML Schema language supported by W3C, and define all attributes and the type of their value (e.g., `orgType` for `Organization` annotations). If the value needs to be one of a predefined set of values (e.g., a part-of-speech tagset), then this set of values is also given in the annotation schema. GATE's annotation schemas were inspired by TIPSTER's annotation type declarations (Grishman 97), however GATE also uses them to facilitate and validate users' input during manual annotation (see Section 5.1).

2.3 JAPE: Finite State Processing Over Annotations

GATE v2 has new facilities for finite state processing over annotations based on regular expressions. JAPE – a Java Annotation Patterns Engine – is similar to the finite state pattern matching transducer part of the TEXTRACT architecture (Neff *et al.*). A JAPE grammar consists of a set of phases, each of which consists of a set of pattern/action rules. The phases run sequentially and constitute a cascade of finite state transducers over annotations. This cascade is similar to the finite state cascade used for parsing (Abney 96) in that rules in the current phase can refer to annotations created in earlier phases and there is no recursion.

The left-hand-side (LHS) of the rules consist of an annotation pattern (or a macro describing such a pattern), that may contain Kleene regular expression operators (e.g. `*, ?, +`). The right-hand-side (RHS) consists of annotation manipulation statements (in JAPE or Java). Annotations matched on the LHS of a rule may be referred to on the RHS by means of labels that are attached to pattern elements.

At the beginning of each grammar, several options can be set:

- Control – this defines the method of rule matching: `brill`, `appelt`, or `first`.

The `brill` style means that when more than one rule matches the same region of the document, they are all fired. With the `appelt` style, only one rule can be fired for the same region of text (the choice depends on the length of the matching region and rule priority). With the `first` style, a rule fires for the first match that is found and no other matching attempts are made for that region of the document.

- Input annotation types – restrict the input to the grammars in order to improve execution speed and control the range for matching in the document.

The example below shows a JAPE rule with pattern and corresponding action. (On the LHS, each such pattern is enclosed in a set of round brackets and has a unique label; on the RHS, each label is associated with an action.) In this example, the Lookup annotation is labelled "jobtitle" and is given the new annotation JobTitle; the TempPerson annotation is labelled "person" and is given the new annotation "Person".

```
Rule: PersonJobTitle
Priority: 20
(
 {Lookup.majorType == jobtitle}
):jobtitle
(
 {TempPerson}
):person -->
    :jobtitle.JobTitle = {rule = "PersonJobTitle"},
    :person.Person = {kind = "personName", rule = "PersonJobTitle"},
```

A large proportion of the processing resources bundled with GATE use JAPE.

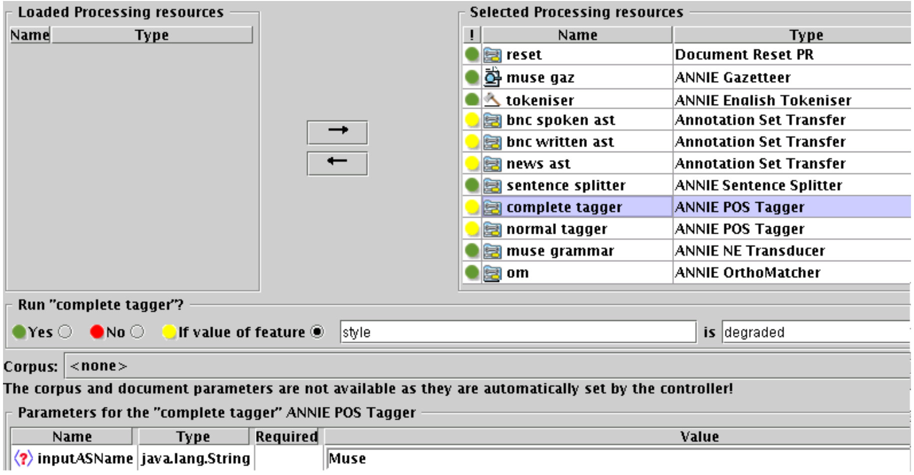
2.4 Execution strategies

The execution strategies in GATE determine how the algorithms (i.e. processing resources) are combined to form a complete application and how that application is executed on the provided input (i.e., language resources such as corpora). These execution strategies are implemented in GATE v2 as controllers and the user chooses the one most suitable for their application.

The two main execution strategies commonly found in LE applications are *pipeline* (i.e., sequential) and *parallel.* For example, GATE's pipeline controller executes the PRs in the specified order and with the given parameters (e.g., a document). A variation of this controller is the corpus pipeline, which executes the PRs sequentially over a given corpus, taking care of loading the documents from the disk (if necessary), processing each of them, and saving the results.

At present GATE does not provide a controller which supports parallel execution of PRs, although it is planned for future work. However, it does support execution of applications on different machines over data shared on the server (see Section 2.5), which enables simultaneous processing of documents.

Other types of execution strategies are *conditional* (equivalent to an `if` statement



Loaded Processing resources

Name	Type

Selected Processing resources

!	Name	Type
🟢	reset	Document Reset PR
🟢	muse gaz	ANNIE Gazetteer
🟢	tokeniser	ANNIE English Tokeniser
🟡	bnc spoken ast	Annotation Set Transfer
🟡	bnc written ast	Annotation Set Transfer
🟡	news ast	Annotation Set Transfer
🟢	sentence splitter	ANNIE Sentence Splitter
🟡	complete tagger	ANNIE POS Tagger
🟡	normal tagger	ANNIE POS Tagger
🟢	muse grammar	ANNIE NE Transducer
🟢	om	ANNIE OrthoMatcher

Run "complete tagger"?

🟢Yes ○ 🔴No ○ 🟡If value of feature ◉ style is degraded

Corpus: <none>

The corpus and document parameters are not available as they are automatically set by the controller!

Parameters for the "complete tagger" ANNIE POS Tagger

Name	Type	Required	Value
inputASName	java.lang.String		Muse

Fig. 2. GATE's conditional controller

in programming languages) and *iterative* (equivalent to loop statements). The conditional controller (see Figure 2) was introduced in GATE v2 in order to allow flexible processing of heterogenous data (e.g., a corpus consisting of documents from several domains, styles, and genres). The controller allows the user to specify for each PR the conditions when it can be executed: `always`; `never`; or `conditionally` – only if a particular feature with a given value is present on the document.

For example, a categorisation module that always fires can assign different genres to each document as features (e.g., sport, political, business). Then the application can contain PRs trained specifically for the given type of documents and at execution time, based on the document type, the controller will determine which PRs are executed. For example, a text about sport would use a different named entity recogniser from the one about politics (England is a team in the first case and a location in the second) or a text in all uppercase would use a different POS tagger (amongst other things) from a text in mixed case (see Figure 2). However, the application will list all alternative PRs (e.g. two POS taggers and 3 entity recognisers) but only those appropriate for the current document will be executed on it, whereas a different set of PRs might be executed on the next one.

2.5 Distributed Language Resources

With the availability of wider bandwidth and the increasingly distributed nature of research comes a new need for distributed language resources (LRs) which are best stored on server machines with large amounts of disk space, so they are installed once and shared among many researchers. An important step towards the creation of such distributed support is the provision of an efficient and user-friendly client-server architecture. This problem can be decomposed into two major tasks: (i) providing support for distributed resources stored on a server; and (ii) providing client tools that are easy to use.

GATE v2 was extended to offer support for server-based language resources by storing them in relational databases (similar to the way in which ATLAS corpora and annotations are stored and shared (Ma *et al.* 02)). The shared LRs are accessed by the GATE visual environment that acts as a client. Currently Oracle and PostgreSQL are supported, of which the latter is freely available.

Uniform models for storing and accessing resources in files and databases is an actively researched area in Information Retrieval (e.g., (Cutting *et al.* 91)) for storing indexes and document collections, and for more complex annotation models as required by language processing applications, e.g., the TIPSTER architecture (Grishman 97). The difference with GATE's and ATLAS-based uniform models comes from the annotation graphs which are more generic and support the association of more than one graph per document. However, this generality comes at the cost of more complex database schemas, which leads to slower performance due to the higher number of join and select operations required.

In order to avoid performance bottlenecks, GATE assumes that each resource is stored entirely in one place, e.g., file or a given database. It is not possible for parts of a resource (e.g., document attributes) to reside on different servers or storage media. Dynamically evaluated attributes and alignment of corpora are also not supported, which makes the GATE LR model a sub-set of more powerful models like the CQP (Christ 94). However, unlike in CQP, recursive attribute structures are supported (e.g., syntax trees).

GATE users are isolated from the ways in which LRs are stored. When choosing to store an LR in a distributed database, users just have to choose which of the available servers they want to use and provide their user name and password. Then GATE transparently converts all LRs in the corresponding database objects and stores them via JDBC. In this way, all GATE applications are separated from the technicalities of using a database server for distributed LR storage – distributed LRs are accessed in the same way as LRs stored on the local file system.

An important aspect of supporting distributed LRs is security and access control. GATE has role-based access control and every language resource is associated with security properties specifying the actions that certain users and groups may perform with this resource. When users create distributed LRs, they specify access rights for them, their user group and other users. For example, LRs created by one user/group can be made read-only to others, so they can use the data, but not modify it. The access rights available in GATE are similar to those of a Unix file system.

3 New GATE Processing Resources

Provided as part of the GATE infrastructure is a set of reusable processing resources for common LE tasks. (None of them are definitive, and the user can replace and/or extend them as necessary.) In version 2, these are packaged together to form ANNIE (A Nearly-New IE system), which is similar to the Vanilla Information Extraction (VIE) system in GATE v1. The PRs 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 recogniser.

ANNIE consists of the following main processing resources: tokeniser, sentence splitter, POS tagger, gazetteer, name entity tagger (based on JAPE), orthomatcher and coreference resolver (Cunningham *et al.* 02). The resources communicate via GATE's annotation model discussed in Section 2.2. Since these PRs have analogues in the VIE system, supplied with GATE v1 (Cunningham 02), they will not be discussed here due to space liminations.

The **Information Retrieval** (IR) module is a new optional PR that allows the combination of IE and IR in a single application. First, the IR module can be used to find relevant documents in a given corpus and then IE modules can be run to extract relevant information, e.g., in a question-answering application. Also, Lucene[7], the underlying retrieval engine, allows the inclusion of arbitrary features in the document index, facilitating experiments in NLP-based IR, for example.

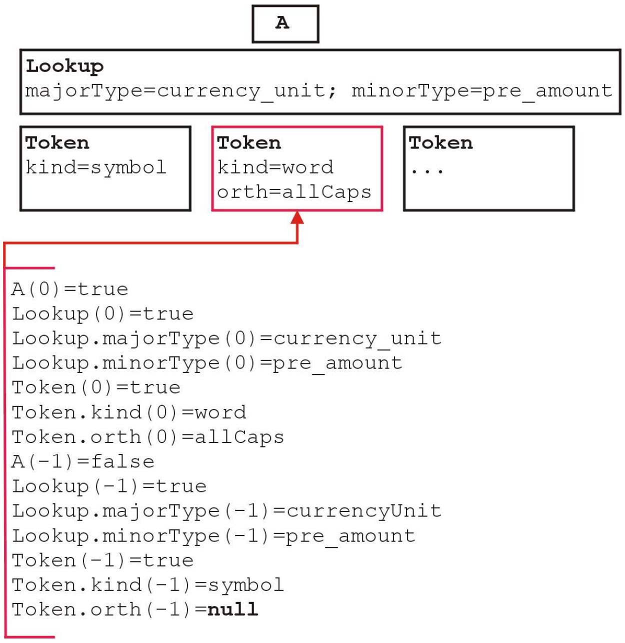
The architectural implications of integrating IR were that it lead to the definition of a generic IR API in GATE which shares commonalities with the TIPSTER IR model (Grishman 97). It consists of three types of resources: the IR index itself can be viewed as a GATE LR, the indexing and retrieval is a PR, and the user interface showing the ranked IR results is a VR. These distinctions are similar to those made in the object-oriented IR architecture discussed in (Cutting *et al.* 91). The difference is mainly in GATE's corpus abstraction model which is more generic and ehnahced to meet the requirements of language processing modules, in addition to IR. The integration of Lucene follows this three-partite GATE IR model and is done via wrapper code which translates between the two APIs.

3.1 Integration of Machine Learning

In order to be able to use Machine Learning (ML) in GATE, a new model was designed for interfacing to ML-based classifiers. The aim is by providing support for a wide range of ML algorithms to facilitate comparative experiments on the same dataset.

From an architectural perspective, the ML interfaces were designed to be generic and to make it easier to integrate different classifiers from different toolkits. In addition, the design provides linguistic information as input to the machine learning algorithms directly from GATE's model of annotations. In practice, this means that a generic mechanism for collecting instances from GATE's data model was implemented. It is based on a declarative XML-based specification of mappings between GATE's annotation types and their attributes and the input features for the classifier. Once collected, the data is exported in the format required by the ML algorithm, which is often a table where each row is an instance and each column is a feature.

[7] `http://jakarta.apache.org/lucene/docs/index.html`



```
A(0)=true
Lookup(0)=true
Lookup.majorType(0)=currency_unit
Lookup.minorType(0)=pre_amount
Token(0)=true
Token.kind(0)=word
Token.orth(0)=allCaps
A(-1)=false
Lookup(-1)=true
Lookup.majorType(-1)=currencyUnit
Lookup.minorType(-1)=pre_amount
Token(-1)=true
Token.kind(-1)=symbol
Token.orth(-1)=null
```

Fig. 3. Example of attributes and their values

When collecting training data, all the annotations of the type specified as instances are found in the given corpus (using GATE's annotation API), and for each of them the set of attribute values is determined. All attribute values, provided as features to the learning algorithm, refer either to the current annotation or to one situated at a specified relative position (e.g., +1 is the next annotation).

Boolean attributes refer to the presence (or absence) of a particular type of annotation overlapping at least partially with the given instances annotation. *Nominal* and *numeric* attributes refer to features on a particular type of annotation that (partially) overlaps the given annotation. One of the boolean or nominal attributes is marked as the *class attribute*, and the values which that attribute can take are the labels for the classes to be learned by the algorithm.

Since linguistic information is often used as attributes for ML in LE applications, GATE provides support for supplying a wide range of linguistic information – part-of-speech, sentence boundaries, gazetteer lists, and named entity class – as attributes for the machine learning algorithms. This information, together with tokenisation information (kind, orthography, and token length) is obtained by using the ANNIE PRs supplied with GATE (Cunningham *et al.* 02). The user chooses which of this information should be provided as features to the learning algorithm.

For example, Figure 3 shows that the attributes supplied to the learning algorithms are derived from the gazetter annotation overlapping the current instance (0), the token annotation overlapping the current instance (0), the token annotation of the previous instance (-1), etc. In this case, instances for learning algorithm are defined to be every occurrence of the Token annotation and what is being learned is the presence or absence of the boolean attribute A.

An ML implementation has two modes of functioning: training – when the model is being built, and application – when the built model is used to classify new instances. Our implementation consists of a GATE processing resource that handles both the training and application phases. It is responsible for detecting all the instances in a corpus and collecting the attribute values for them. The output writer is format specific, so new ones will need to be implemented if new ML libraries are integrated, which use format different from Weka's.

Depending on the type of the attribute, different actions will be performed when classification occurs. For boolean attributes, a new annotation of the type specified in the attribute definition will be created (e.g., annotation of type A). Nominal attributes trigger the addition of the feature on the annotation of the required type (Token in the example above), situated at the position of the classified instance. If no such annotation is present, it will be created.

Currently GATE supports the ML algorithms implemented in the open-source WEKA library (Witten & Frank 99)[8]; other implementations are being added in the SEKT project (see below). WEKA was chosen because it also provides tools for performance evaluation, testing, and attribute selection. In addition, most of the ML algorithms in WEKA can provide a probability distribution rather than a simple classification. This can be used by GATE applications for setting a confidence threshold, thus enabling a choice of balance between precision and recall.

3.2 Ontology-Based PRs

The addition of support for ontologies in GATE (Section 4.2) enabled the creation of ontology-based PRs, i.e., PRs that use the ontology as one of their data resources. One such new component is the ontology-based gazetteer, which treats phrases as instances of concepts from a given ontology. Based on this information, subsequent PRs can access the same ontology and perform reasoning with the information produced by the ontology-based gazetteer, e.g., to obtain the semantic distance between two concepts. The new gazetteer is not tied to any specific ontology format, because it uses GATE's generic ontology model (see Section 4.2).

One of the reasons behind connecting ontologies to the gazetteer lists is to make the process of editing gazetteer entries part of ontology maintenance. Another advantage is that it enables GATE PRs to annotate automatically in the texts the instances already available in the user's ontology.

The ontology-based gazetteer PR comes with a corresponding VR that supports editing of the gazetteer lists. The user can also specify how they are connected to ontology classes. Figure 4 shows the ontology with all instance names displayed on the right as a gazetteer list. There can be more than one list of instance names per concept, as shown in this example. In this way, lists provided with the GATE

[8] WEKA homepage: `http://www.cs.waikato.ac.nz/ml/weka/`

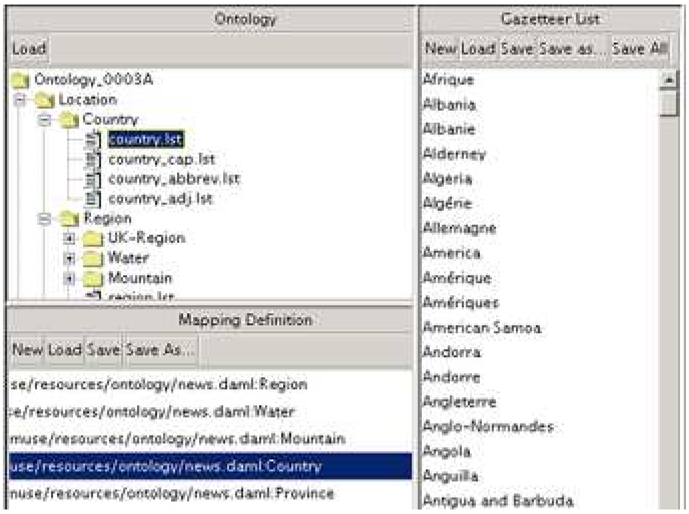


Fig. 4. Connecting the user's ontology to the GATE gazetteers

system can be used simultaneously with lists derived from instances in the user's ontology. In cases where text is annotated as matching a gazetteer list entry coming from the ontology, the annotation provides information about its ontological class and the identifier of the ontology itself. Using this information, subsequent PRs can find and query the ontology for reasoning purposes.

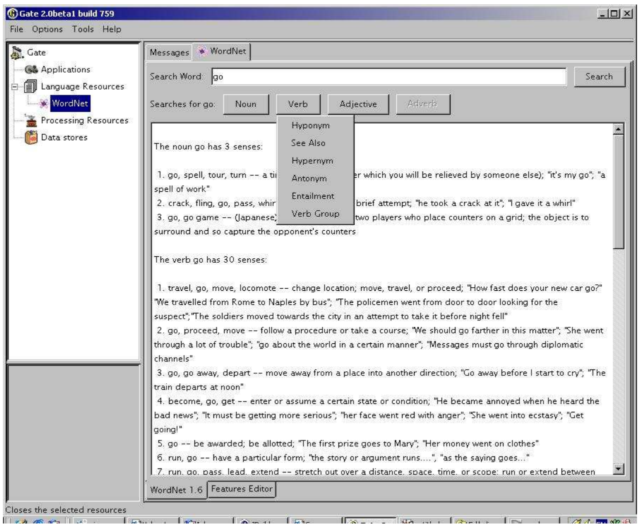
The JAPE transducer was also extended to provide access to ontologies on the right-hand side of JAPE rules. This allows JAPE rules to add new information to ontologies (e.g., instances) or to consult them (e.g., to obtain semantic distance between concepts). We are also planning to extend the JAPE transducer to take into account subsumption relations in the ontology when matching on the left-hand side. So for example, a rule might look for an organization followed by a location, in order to create the `locatedAt` relationship between them. If JAPE takes into account subsumption, then the rule will automatically match all subclasses of `Organisation` in the ontology, e.g., `Company`, `GovernmentOrg`.

4 New Types of Language Resources

The two most frequently used language resources in GATE are documents and corpora (collections of documents). Documents have content, features (e.g., encoding, language), and sets of annotations organised in annotation graphs (see Section 2).

However, experience from GATE v1 showed the need for other types of LRs, such as lexicons, ontologies, and thesauri, because they are used frequently in many LE applications and common APIs would greatly facilitate their reuse.

As already discussed, existing GATE PRs like gazetteer lookup and JAPE transducer were extended to use the new ontology LR. Similarly, other existing PRs would need to be extended (e.g., by adding a new parameter) if they need to ac-



The noun go has 3 senses:

1. go, spell, tour, turn -- a ti er which you will be relieved by someone else); "it's my go"; "a spell of work"
2. crack, fling, go, pass, whir brief attempt; "he took a crack at it"; "I gave it a whirl"
3. go, go game -- (Japanese) two players who place counters on a grid; the object is to surround and so capture the opponent's counters

The verb go has 30 senses:

1. travel, go, move, locomote -- change location; move, travel, or proceed; "How fast does your new car go?" "We travelled from Rome to Naples by bus"; "The policemen went from door to door looking for the suspect";"The soldiers moved towards the city in an attempt to take it before night fell"
2. go, proceed, move -- follow a procedure or take a course; "We should go farther in this matter"; "She went through a lot of trouble"; "go about the world in a certain manner"; "Messages must go through diplomatic channels"
3. go, go away, depart -- move away from a place into another direction; "Go away before I start to cry"; "The train departs at noon"
4. become, go, get -- enter or assume a certain state or condition; "He became annoyed when he heard the bad news"; "It must be getting more serious"; "her face went red with anger"; "She went into ecstasy"; "Get going!"
5. go -- be awarded; be allotted; "The first prize goes to Mary"; "Her money went on clothes"
6. run, go -- have a particular form; "the story or argument runs....", "as the saying goes..."
7. run, go, pass, lead, extend -- stretch out over a distance, space, time, or scope; run or extend between

Fig. 5. The GATE WordNet browser

cess the new LRs, e.g., Wordnet. One way around this problem is to pre-compile certain information from lexicon LRs as gazetteer lists, which are used instead at run-time by the original gazetteer PR. For example, lists of jobtitles were compiled semi-automatically from WordNet and then used for named entity recognition and anaphora resolution. Naturally, this approach is feasible only when a small, already known number of queries need to be made to the lexicon.

4.1 Lexicons

Lexicons are a required resource in many language analysis modules. More specifically, in the context of applying NLP technology for the Semantic Web, semantic resources like thesauri and semantic lexicons like WordNet play an important role (Buitelaar & Declerck 03). Consequently, GATE's LR model was extended to cover certain new types of lexical resources.

A difficulty in lexicon reuse is that each resource has its own representation syntax and covers a particular subset of linguistic phenomena. GATE models lexicons in an object-oriented way, yielding an initial set of lexicon-specific classes. The commonalities between resources are exploited by subsequent, incremental integration of these classes into one common object model. The representation of objects within this model is geared towards generally accepted linguistic concepts.

This approach has several advantages: the linguistic knowledge is modelled in a conceptual and maximally uniform way; the linguistic resources are represented by maximally uniform data structures; overlaps and differences between resources can be identified by means of the common object model; the model provides a level playing field for the evaluation of lexical resources. However, in describing the more specific parts of the model GATE sticks closely to the terminology and structures of the existing resources.

In other words, the lexicon model starts at a very abstract level defining a lexicon as a collection of lemmas which have part-of-speech and several other features, which tend to be shared among most lexicons (e.g., definitions). If a concrete lexicon integrated with this model does not provide some of the information (e.g., definitions) then the lexicon wrapper will return an empty value. At lower level of the lexicon class hierarchy are more specific models of lexicons, e.g., the WordNet model which also contains synonym sets and support for its semantic relations like hyponymy, hypernymy, and meronymy.

Each lexicon is integrated with GATE via a wrapper which is a class towards the bottom of GATE's lexicon class hierarchy. Typically the wrapper follows the structure of the given lexicon and thus defines the lexicon-specific model. However, due to the existence of the lexicon class hierarchy, a PR which only needs to perform lexical lookup in order to obtain part of speech information can use the more general lexicon model, and access the newly integrated lexicon via that more general model, without need for any adaptation.

This idea is similar to the way GATE supports different document formats, but the common data model of annotations insulates the PRs from having to handle each of the formats. At the same time, however, the formatting information is preserved, so the PR can use or analyse it if required.

The most extensive support is currently provided for WordNet, because it is the most widely used lexical database, is freely available, and also has semantic relations, which make it suitable for Semantic Web applications. The GATE object-oriented model of WordNet follows closely that of the original resource and is based on synonym sets and the semantic relations defined between them (Miller 95). This makes it accessible programmatically by any GATE PR via this WordNet API or the more generic lexicon API. In addition, a WordNet VR (see Figure 5) is provided to enable people to browse the resource using a graphical interface similar to that of the original WordNet browser. The advantages of implementing a new Java-based browser as part of GATE are:

- reusability: it is easy to reuse and integrate with other Java applications. It is also easily extendable, because it has a well-defined, public API and open source code.
- generality: it uses the GATE lexicon object-oriented model, which means that it can be used to display other lexicons with a similar structure, not just WordNet.

Recently the lexicon class hierarchy in GATE was extended to support editable lexicons, as required by language generation applications, which often contain domain-specific terms not contained in existing NLP lexicons. A corresponding lexicon editor was also implemented. This is ongoing work carried out as part of the MIAKT project (Bontcheva 04).

4.2 Ontologies

Recent Semantic Web (Fensel *et al.* 02) developments have increased the need for LE applications that use ontologies (Fensel 01) and knowledge bases in order to obtain a formal representation of and possibly reason about their domain (e.g., (Saggion *et al.* 03b; Handschuh *et al.* 02)). In addition, LE methods can be used to populate an ontology with new instances discovered in texts or even to construct an ontology automatically.

Consequently, GATE was extended to provide support for importing, accessing and visualising ontologies as a new type of Language Resource. Strictly speaking, ontologies (and knowledge bases) should not be called LRs, because they are language independent. However, for backwards compatibility, we decided to keep the original GATE terminology of LRs. Under this definition Language Resources cover all types of declarative resources used for language processing, e.g., grammars, lexicons, ontologies, as well as corpora and documents.

Some of the ontology functionality is provided through the integration of the Protégé editor (Noy *et al.* 01). We also developed GATE-specific ontology visualisation, because it facilitates the process of developing and testing PRs that use ontologies (e.g., gazetteers and JAPE-based modules). This viewer was also incorporated recently in a new GATE editor for connecting NLG lexicons to ontologies (Bontcheva 04).

The proliferation of ontology formats for the Semantic Web, e.g., DAML+OIL (Horrocks & vanHarmelen 01), OWL (Bechhofer *et al.* 03), RDF(S) (Lassila & Swick 99), means that LE applications need to deal with these different formats. In order to avoid the cost of having to parse and represent ontologies in each of these formats in each LE application, a common object-oriented model of ontologies with a unified API (Application Programming Interface) was created.

This approach has well-proven benefits, because it enables each PR to use this format-independent ontology model, thus making the PRs immune to changes in the underlining formats. If a new format needs to be supported, the PRs will automatically start using ontologies in this format, due to the seamless format conversion provided by GATE. From a language engineer's perspective the advantage is that they only need to learn one API and model, rather than having to learn many different and rather idiosyncratic ontology formats.

Since OWL, DAML+OIL and RDF(S) have different expressive powers, GATE's ontology model consists of a class hierarchy with a growing level of expressivity. At the top is a taxonomy class which is capable of representing taxonomies of concepts, instances, and inheritance between them. Multiple inheritance is not supported.

At the next level is an ontology class which can represent also properties, i.e., relate concepts to other concepts or instances. Properties can have cardinality restrictions and be symmetric and/or transitive. There are also methods providing access to their sub- and super-properties and inverse properties. The property model distinguishes between object (relating two concepts) and datatype properties (relating a concept and a datatype such as string or number).

The expressivity of this ontology model is aimed at being broadly equivalent to OWL Lite. In the case of a DAML+OIL ontology, GATE uses a sub-set of Jena's API to read in the model and instantiate the GATE ontology classes. Any information outside the GATE model is ignored. When reading RDFS, which is less expressive than OWL Lite, GATE only instantiates the information provided by that model. If the API is used to access one of these unsupported features, it returns empty values.

The ontology support is currently being used in several projects that develop Information Extraction (IE) to populate ontologies with instances derived from texts (Multiflora (Wood *et al.* 03), hTechSight (see Section 6)) and also in the MIAKT project (Bontcheva 04) for language generation from ontologies. The IE systems produce annotations referring to the concepts and instances in the ontology, based on their URIs (Uniform Resource Identifiers).

In order to enable quantitative evaluation of such ontology-based IE systems, as part of the EU-funded SEKT project[9], GATE will be extended with document annotation facilities that support semi-automatic annotation given an ontology, so appropriately annotated corpora can be created. This editor will be based on GATE's annotation schemas which will be extended with optional `ontology`, `class`, and `instance` features with string values.

5 An Extendable Model of Visual Resources

Visual resources are providing a graphical user interface for viewing and editing of LRs (e.g., documents) and PRs (e.g., gazetteer lists). The component model of GATE was extended in version 2 to include VRs, in order to enable extensibility of GATE's development environment and reuse of VRs in other applications, thus overcoming an important limitation of the earlier GATE release. In other words, the need for reusable and flexible GUI components resulted in one of the biggest changes in the GATE architecture from version 1.

VRs are discovered by GATE using the CREOLE XML meta-data configuration, used for PRs and LRs as well. The meta-data for VRs is as follows:

```
<RESOURCE>
  <NAME>Annotations Editor</NAME>
  <CLASS>gate.gui.DocumentEditor</CLASS>
  <!-- type values can be "large" or "small"-->
  <GUI TYPE="large">
    <MAIN_VIEWER/>
    <RESOURCE_DISPLAYED>gate.Document</RESOURCE_DISPLAYED>
  </GUI>
</RESOURCE>
```

The name and class attributes are the same as for any GATE resource. The GUI

[9] http://sekt.semanticweb.org. The SEKT partners are: British Telecommunications Plc.; Empolis GmbH; University of Sheffield; University of Karlsruhe; Jozef Stefan Institute; Institut fur Informatik der Universitat Innsbruck; Intelligent Software Components S. A.; Kea-pro GmbH; Ontoprise GmbH; Sirma AI Ltd; Vrije Universiteit Amsterdam; Autonomous University of Barcelona.

information specifies the size of the window in which it is to be displayed, whether it is the main viewer for this resource (resources can have more than one VR associated with them), and which resource type is displayed/editied by this VR.

The main GATE GUI uses this information to configure itself automatically. This means that if a new type of resource is introduced (e.g., ontology), a corresponding VR can be implemented and added to the CREOLE repository (e.g., an ontology editor) and without further programming this VR will become available in GATE's visual development environment, associated with the resources it can handle. Inheritance is used when determining which VRs can show which resources, e.g., a VR associated with documents will show the three subclasses of database, file-based, and transient documents. If a new (more specialised) type of LR/PR is introduced, existing less specialised VRs can be used until a new VR, supporting the full functionality of the new resource, is developed.

Currently GATE has 11 VRs in its default configuration and 6 optional ones. Example of available VRs are:

- Document VRs: display document content and the associated annotations (see Figure 1), including coreference chains.
- Annotation VRs: display/editing of annotations (see Section 5.1), including specialised editors for tree structures (e.g., sentence trees).
- Corpus VR: allows the creation and editing of corpora.
- Other VRs associated with LRs – ontology VRs, e.g., the onto-gazetteer VR (Figure 4); lexicon VRs: handle lexicons, such as WordNet (see Figure 5); Information Retrieval VR: displays the IR results as a ranked list of documents.
- VRs associated with applications: allow creation and editing of NLP applications (see Figure 2).
- VRs associated with PRs: enable editing of the resources used by PRs, such as the gazetteer lists used by the gazetteer lookup PR; also show the values of the PR parameters that were specified at creation time.

5.1 Corpus Annotation Facilities

Since many NLP algorithms require annotated corpora for training, GATE's development environment now provides easy-to-use and extendable facilities for text annotation. These facilities were tested extensively by building corpora of name entity annotated texts for the MUSE, ACE, and MUMIS applications (Maynard *et al.* 01; Saggion *et al.* 03b). The annotation tools follow the new VR model and are easy to embed and reuse in other 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 (similar to the pre-tagging functionality in the Alembic workbench(Day *et al.* 97)). 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, enabled the automatic annotation of location information in a large collection of 18th century court reports from the Old Bailey in London (Bontcheva *et al.* 02).

Since manual annotation is a difficult and error-prone task, GATE v2 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) 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 appear there or the user wants to provide more detailed information (not just the type), then an annotation editing dialogue can be used. The XML-based annotation schemas (see Section 2.2) are used to facilitate and validate the user input. This approach is similar to NITE's use of XML schemas for data validation and XML stylesheets for dynamic creation of annotation interfaces (Carletta *et al.* 03).

6 Applications

In this section we describe briefly some of the LE applications we have developed using the new features introduced in GATE v2. The **MUSE** system (Maynard *et al.* 01) 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. MUSE used extensively many of the new features like the regression testing tools (Cunningham *et al.* 02) and the conditional execution strategy (Section 2.4). The MUSE system also exploited the machine learning facilities in GATE in order to perform NE recognition in Arabic and Chinese.

The **hTechSight** project[10] performs ontology-based IE for semantic tagging of job adverts, news and reports in the chemical engineering domain. The system is built using the ontology support in GATE v2. More specifically, the domain ontology (in DAML+OIL) is accessed via GATE's ontology model, which makes the IE components independent from the concrete ontology format, i.e., the same IE system could be run with an OWL ontology without any change to the PRs. The terminological gazetteer lists were linked to classes in the ontology model, so gazetteer lookup provided ontological information. In addition, the GATE ontology model was used on the right-hand side of the hTechSight JAPE grammars, so newly discovered instances (e.g., new job position) were added to the ontology. The GATE ontology visualisation tools were used during development and debugging.

Another project that benefitted from the new ontology and lexicon support in GATE v2 is **MIAKT** (Bontcheva 04). It is developing a set of GATE-compatible tools for editing, storage, and maintenance of NLG lexicons. The model of NLG lexicons is an extension of the GATE lexicon model with methods for adding new lexical entries. The ontology model was used as a basis for a mapping tool which connects lexical entries to concepts and instances in the ontology. This was required by the MIAKT application in order to generate reports from ontologies in diverse formats, e.g., OWL, DAML+OIL.

[10] http://http://prise-serv.cpe.surrey.ac.uk/techsight/

GATE v2 has also been used for corpus annotation (e.g., in the American National Corpus project (Macleod *et al.* 02)), multimedia indexing (Saggion *et al.* 03b), automatic summarisation (Saggion *et al.* 03a), e-science (Wood *et al.* 03), and digital libraries (Bontcheva *et al.* 02) (e.g., the Perseus digital library (Crane 02)). Further details about some of the projects using GATE are available at `http://gate.ac.uk/projects.html`.

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 00).

The system most similar in terms of functionality and spirit to GATE is IBM's TEXTRACT architecture (Neff *et al.*). Similar to GATE, TEXTRACT has a graphical development environment, a number of reusable components (called plugins), a finite state transducer that allows rapid application prototyping, and support for processing of large corpora. The main differences from GATE are in the data representation model, the implementation language (C++ vs Java), and the restriction to language analyser applications. TEXTRACT has led to a new system called UIMA (Ferrucci & Lally).

Work on standard ways to deal with XML data is also relevant here, such as the LT XML work at Edinburgh (Thompson & McKelvie 97), WHAT (Schäfer 03), and NITE (Carletta *et al.* 03), as is work on managing collections of documents and their formats, e.g. (Brugman *et al.* 98; Grishman 97; Zajac 98). We have also drawn from work on representing information about text and speech, e.g. (Brugman *et al.* 98; Mikheev & Finch 97; Zajac 98; Young *et al.* 99), as well as recent annotation standards, such as the ATLAS project (an architecture for linguistic annotation) at LDC (Bird *et al.* 00b). Our approach is also related to work on user interfaces to architectural facilities such as development environments, e.g. (Brugman *et al.* 98; Carletta *et al.* 03) and to work on comparing different versions of information, e.g. (Paggio 98).

Another relevant area is work on indexing complex structures in annotations. The Corpus Query System (Christ 94) is a frequently cited source in this area, providing indexing and search of corpora and later of WordNet. Similar ideas have been implemented in CUE (Mason 98) for indexing and search of annotated corpora. Some work on indexing in the LT XML system is reported in (McKelvie & Mikheev 98). (Bird *et al.* 00a) propose a query language for the LDC annotation graph model, called AGQL. (Cassidy 02) discusses the use of XQuery as an annotation query language and concludes that it is good for dealing with hierarchical data models like XML, but needs extending with better support for network data models like annotation graphs. In comparison, GATE v2 indexes and retrieves annotations by storing them in a relational database, indexed by type, attributes and their values, allowing more than one annotation graph per document. In this way, it is possible to retrieve all documents that contain a given attribute and/or value or retrieve all annotations of a given type in a corpus, without having to traverse each document separately. The query language used is SQL.

Flexible and extendable visualisation of the complex linguistic structures frequently encoded in corpora was the focus of the NITE project (Carletta *et al.* 03). The goal is to provide XML-based tools to help users with building their own specialised graphical interfaces to manipulate complex data in heavily-annotated multimodal corpora. Some of the ideas are similar to GATE's VR model and both GATE and NITE allow the use of XML schemas to validate user input.

What makes this work particularly novel is that it addresses a very wide range of issues in LE application development in a flexible and extensible infrastructure. In addition, it promotes robustness, re-usability, experimental repeatability and scalability as important principles that help with the construction of practical LE systems. This paper showed how the original architecture was extended in response to new challenges by implementing models of lexical and ontological resources, machine learning, IR, and reusable VRs. Where possible, existing and widely used implementations of such components were integrated with GATE.

8 Discussion

The GATE architecture has been created with generality in mind. The rationale behind that design decision is to make GATE maximally extendable and independent of any particular NLP application (e.g., IE, IR). However, generality often entails more complex data structures and non-optimised implementations. For example, the new annotation graph data model is more generic than the TIPSTER one and can accomodate NLG components. On the other hand, we found it to be less efficient when storing LRs in databases, due to the larger number of database tables and existence of more than one annotation graph per document. In other words, there is a trade-off between generality and performance and while GATE's emphasis is generality, industrially-motivated LE architectures such as TEXTRACT (Neff *et al.*) make performance their main priority.

The new types of LRs: lexicons and ontologies – did not require any changes to the architecture. Instead, the LR hierarchy was enhanced with new classes. The enhancement of the LR model to include ontologies created a slight terminological problem because ontologies are not typically considered as modelling language. However, the essential distinction between data resources (LRs) and algorithms that use them (PRs) remains valid and maybe, in future releases of GATE, LRs need to renamed to DRs (data resources) to reflect their wider range and function.

The main change to the architecture, as already discussed in Section 5, came from the addition of Visual Resources (VRs) which enabled reuse and extensibility of the GATE GUI components.

9 Conclusion and Future Work

In this paper we have described recent extensions to a widely-used open-source infrastructure for language engineering software – GATE – which aims to assist the development of robust applications and resources for NLP through a generic set of tools and models. We believe that some of the main reasons for its success are its extensible and highly customisable nature, which allows the system to evolve easily in response to new challenges, and its open-source distribution and extensive documentation and online user support. The system now has thousands of users at hundreds of sites, and we hope to build on this user base to continue to meet the challenges posed by the rapid evolution of the LE field.

One future direction is 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. In order to facilitate reuse and interoperability we plan on building on the RAGS architecture for language generation (Mellish *et al.*). We are currently developing generation tools for the Semantic Web in the context of the MIAKT project. These requirements for NLG support come from the need for automatic documentation of ontologies and also for intelligent knowledge publishing (Bontcheva *et al.* 01).

Another future direction is towards providing better facilities for writing and debugging GATE's finite-state grammars, similar to those provided by the TEXTRACT architecture (Neff *et al.*). The main concern is to enable non-expert users to modify existing and write their own language processing tools, based on GATE's JAPE pattern-action language. The requirements arose in the context of our digital library projects (Bontcheva *et al.* 02).

The biggest challenge comes from grid-based e-science, the Semantic Web, and Web services, where LE applications can and should play a vital role. A number of existing projects, e.g., MIAKT, hTechSight, use Web services to provide interoperability between the system's components, which frequently run on different hardware and software platforms. Therefore, the existing GATE facilities need to be extended further in order to enable:

- easy deployment of grid-based LE applications, capable of utilising the vast computing facilities of the Grid to perform high-speed language processing on big corpora;
- providing a better synergy between Semantic Web and LE technology by extending the support for ontologies and knowledge bases and integration of reasoning and ontology support infrastructures such as Sesame[11] or KIM (Popov *et al.*).
- use of Web Services for building distributed LE applications.

A closer synergy between Semantic Web and LE infrastructures and standards is starting to emerge from the perspective of improving resource reusability and interoperability through development of standards for encoding annotation data. Recent

[11] http://sesame.aidministrator.nl/

efforts are aimed at the creation of a framework for linguistic annotations based on XML, RDF, and DAML+OIL/OWL standards for defining the semantics of the annotations (Ide & Romary). GATE will benefit directly from such standardisation efforts, because they will improve the compatibility between NLP components by specifying common definitions of language resources such as lexicons.

Among these, Grid-based computing and Web services support are most likely to have architectural implications for GATE, amounting to more than integration of existing tools or implementation of format converters. This work will be carried out as part of the SEKT project.

References

S. Abney. Partial parsing via finite state cascades. *Natural Language Engineering*, 2(4):337–344, 1996.

S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference. Technical report, W3C Proposed Recommendation 15 December 2003, http://www.w3.org/TR/2003/PR-owl-ref-20031215/, 2003.

S. Bird, P. Buneman, and W. Tan. Towards a query language for annotation graphs. In *Proceedings of the Second International Conference on Language Resources and Evaluation*, Athens, 2000.

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

K. Bontcheva. Open-source Tools for Creation, Maintenance, and Storage of Lexical Resources for Language Generation from Ontologies. In *Proceedings of 4th Language Resources and Evaluation Conference (LREC'04)*, 2004.

K. Bontcheva, C. Brewster, F. Ciravegna, H. Cunningham, L. Guthrie, R. Gaizauskas, and Y. Wilks. Using HLT for Acquiring, Retrieving and Publishing Knowledge in AKT: Position Paper. In *Workshop on Human Language Technology and Knowledge Management*, Toulouse, France, 2001. `http://www.elsnet.org/acl2001-hlt+km.html`.

K. Bontcheva, D. Maynard, H. Cunningham, and H. Saggion. Using Human Language Technology for Automatic Annotation and Indexing of Digital Library Content. In *Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries (ECDL'2002)*, Rome, Italy, 2002.

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

P. Buitelaar and T. Declerck. Linguistic annotation for the semantic web. In S. Handschuh and S. Staab, editors, *Annotation for the Semantic Web*. IOS Press, 2003.

L. Burnard. Users Reference Guide for the British National Corpus. `http://info.ox.ac.uk/bnc/`, May 1995.

J. Carletta, S. Evert, U. Heid, J. Kilgour, J. Robertson, and H. Voormann. The NITE XML Toolkit: flexible annotation for multi-modal language data. *Behavior Research Methods, Instruments, and Computers*, 35(3), 2003. Special issue on Measuring Behavior.

S. Cassidy. Xquery as an annotation query language: a use case analysis. In *Proceedings of 3rd Language Resources and Evaluation Conference (LREC'2002)*, Gran Canaria, Spain, 2002.

O. Christ. A Modular and Flexible Architecture for an Integrated Corpus Query System. In *Proceedings of the 3rd Conference on Computational Lexicography and Text Research (COMPLEX '94)*, Budapest, 1994. `http://xxx.lanl.gov/abs/cs.CL/9408005`.

G. Crane. Cultural Heritage Digital Libraries: Needs and Components. In *Proceedings of the 6th European Conference on Digital Libraries*, pages 626–637, Rome, Italy, September 2002. Springer-Verlag Berlin Heidelberg 2002.

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

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

H. Cunningham, K. Humphreys, R. Gaizauskas, and Y. Wilks. Software Infrastructure for Natural Language Processing. In *Proceedings of the 5th Conference on Applied Natural Language Processing (ANLP-97)*, March 1997. http://xxx.lanl.gov/-abs/cs.CL/9702005.

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.

D. Cutting, J. Pedersen, and P.-K. Halvorsen. An Object-Oriented Architecture for Text Retrieval. In *Proceedings of RIAO '91*, pages 285–298, Barcelona, 1991.

J. Davies, D. Fensel, and F. van Harmelen, editors. *Towards the Semantic Web: Ontology-driven Knowledge Management.* Wiley, 2002.

D. Day, J. Aberdeen, L. Hirschman, R. Kozierok, P. Robinson, and M. Vilain. Mixed-Initiative Development of Language Processing Systems. In *Proceedings of the 5th Conference on Applied Natural Language Processing (ANLP-97)*, 1997.

D. Fensel. *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce.* Springer-Verlag, 2001.

D. Fensel, W. Wahlster, and H. Lieberman, editors. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential.* MIT Press, 2002.

D. Ferrucci and A. Lally. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Natural Language Engineering.* This issue.

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

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

I. Horrocks and F. van Harmelen. Reference Description of the DAML+OIL (March 2001) Ontology Markup Language. Technical report, 2001. http://www.daml.org/2001/03/reference.html.

N. Ide and L. Romary. Standards for language resources. *Natural Language Engineering.* This issue.

N. Ide, P. Bonhomme, and L. Romary. XCES: An XML-based Standard for Linguistic Corpora. In *Proceedings of the Second International Language Resources and Evaluation Conference (LREC)*, pages 825–830, Athens, Greece, 2000.

O. Lassila and R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Technical Report 19990222, W3C Consortium, http://www.w3.org/-TR/REC-rdf-syntax/, 1999.

X. Ma, H. Lee, S. Bird, and K. Maeda. Models and tools for collaborative annotation. In *Proceedings of 3rd Language Resources and Evaluation Conference (LREC'2002)*, Gran Canaria, Spain, 2002.

C. Macleod, N. Ide, and R. Grishman. The American National Corpus: Standardized Resources for American English. In *Proceedings of 2nd Language Resources and Evaluation Conference (LREC)*, pages 831–836, Athens, Greece, 2002.

O. Mason. The CUE Corpus Access Tool. In *Workshop on Distributing and Accessing Linguistic Resources*, pages 20–27, Granada, Spain, 1998. `http://www.dcs.shef.ac.uk/-~hamish/dalr/`.

D. Maynard, V. Tablan, C. Ursu, H. Cunningham, and Y. Wilks. Named Entity Recognition from Diverse Text Types. In *Recent Advances in Natural Language Processing 2001 Conference*, pages 257–274, Tzigov Chark, Bulgaria, 2001.

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

D. McKelvie and A. Mikheev. Indexing SGML files using LT NSL. LT Index documentation, from `http://www.ltg.ed.ac.uk/`, 1998.

C. Mellish, D. Scott, L. Cahill, R. Evans, D. Paiva, and M. Reape. A Reference Architecture for Generation Systems. *Natural Language Engineering*. This issue.

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

G. Miller. WordNet: a Lexical Database for English. *Communications of the ACM*, Volume 38(Number 11), November 1995.

M. S. Neff, R. J. Byrd, and B. K. Boguraev. The Talent System: TEXTRACT Architecture and Data Model. *Natural Language Engineering*. This issue.

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

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

B. Popov, A. Kiryakov, A. Kirilov, D. Manov, D. Ognyanoff, and M. Goranov. KIM – Semantic Annotation Platform. *Natural Language Engineering*. This issue.

H. Saggion, K. Bontcheva, and H. Cunningham. Robust Generic and Query-based Summarisation. In *Proceedings of the European Chapter of Computational Linguistics (EACL), Research Notes and Demos*, 2003.

H. Saggion, H. Cunningham, K. Bontcheva, D. Maynard, O. Hamza, and Y. Wilks. Multimedia Indexing through Multisource and Multilingual Information Extraction; the MUMIS project. *Data and Knowledge Engineering*, 2003. To appear.

U. Schäfer. WHAT: An XSLT-based Infrastructure for the Integration of Natural Language Processing Components. In *HLT-NAACL 2003 Workshop: Software Engineering and Architecture of Language Technology Systems (SEALTS)*, pages 9–16, 2003.

V. Tablan, C. Ursu, K. Bontcheva, H. Cunningham, D. Maynard, O. Hamza, T. McEnery, P. Baker, and M. Leisher. A Unicode-based Environment for Creation and Use of Language Resources. In *3rd Language Resources and Evaluation Conference*, 2002.

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

I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.

M. M. Wood, S. J. Lydon, V. Tablan, D. Maynard, and H. Cunningham. Using parallel texts to improve recall in IE. In *Recent Advances in Natural Language Processing*, Bulgaria, 2003.

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

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