Natural Language Engineering **1** (1): 000–000. Printed in the United Kingdom

Architectural Elements of Language Engineering Robustness

Diana Maynard and Valentin Tablan and Hamish Cunningham
and Cristian Ursu and Horacio Saggion and Kalina Bontcheva and Yorick Wilks
Dept. of Computer Science
University of Sheffield
Regent Court, 211 Portobello St
Sheffield, S1 4DP, UK
[diana,valyt,hamish,cursu,saggion,kalina,yorick]@dcs.shef.ac.uk

(*Received 28 January 2002*)

Abstract

We discuss robustness in LE systems from the perspective of *engineering*, and the predictability of both outputs and construction process that this entails. We present an architectural system that contributes to engineering robustness and low-overhead systems development (GATE, a General Architecture for Text Engineering). To verify our ideas we present results from the development of a multi-purpose cross-genre Named Entity recognition system. This system aims be robust across diverse input types, and to reduce the need for costly and time-consuming adaptation of systems to new applications, with its capability to process texts from widely differing domains and genres.

1 Introduction

Let us say, for expository rather than dogmatic purposes, that Language Engineering (LE) may be defined as:

> ...the discipline or act of engineering software systems that perform tasks involving processing human language. Both the construction process and its outputs are measurable and predictable. The literature of the field relates to both application of relevant scientific results and a body of practice. (Cunningham 99a)

The relevant scientific results in this case are the outputs of Computational Linguistics, Natural Language Processing and Artificial Intelligence in general. Unlike these other disciplines, LE, as an engineering discipline, entails *predictability*, both of the process of constructing LE-based software and of the performance of that software after its completion and deployment in applications. The architect currently designing an extension to the home of one of the authors knows both how long his

job will take him, and how well the finished room will stand up to the rigours of the Northern English climate, when constructed according to his specifications.[1]

The issue of predictability of performance is often acknowledged in this context; just as important but less often discussed is predictability of the effort and necessary inputs to the construction process. We cannot claim to be engineers, however, if we request unbounded commitments of resources of any type for our construction processes. This is not at all to say that endeavours that cannot predict the necessary resources are not worthwhile – they are foundational in scientific research; merely that such enterprises are not in fact engineering.

We believe that it is useful to examine the project of robustness in LE in light of the above considerations, and to consider means by which predictability may be achieved. This paper will argue that software architecture (Shaw & Garlan 96; Cunningham 00) can contribute significantly to this task. The first part of the paper will discuss a particular architecure, the GATE General Architecture for Text Engineering (Cunningham 01; Cunningham *et al.* 97; Cunningham *et al.* 94) and its potential contribution to robustness in LE. The second part of the paper will look at a particular LE application that has been developed using GATE, namely a cross-genre entity recognition system, for which we will detail results measurements that are intended to predict the systems performance in an application setting. This system is under development as part of the MUSE (Multi-Source Entity finder) project (Maynard *et al.* 01).

MUSE addresses a different aspect of robustness, specific to Information Extraction (IE). Whereas most IE systems are designed to extract fixed types of information from documents in a very specific domain, MUSE aims to extract entity information from a variety of different genres.

Predictability of software construction processes is notoriously difficult (Pressman 94). Areas in which infrastructural support can help improve predictability in LE development include: data visualisation for debugging and measurement; component-based development in which alternative configurations can be compared straightfowardly; deployment and embedding of components; automated performance evaluation. Section 2 examines these areas.

The attainment of predictability of performance of a robust LE system is closely related to the ability to measure that performance precisely in a large number of test cases that exemplify as closely as possible the eventual execution environment of the software. Section 3 discusses GATE's implementation of automated measurement and its visualisation.

Some general issues concerning the need for robustness in IE systems, and in particular for named entity recognition, are discussed in Section 4, which gives rise to the motivation for the MUSE project. The system is described in more detail in the following sections: the processing resources used (Section 5); the scope of the project (Section 6); and the adaptation needed to achieve robustness (Section 7).

[1] At least we sincerely hope that he knows.

In the last part, we measure the performance of the system (Section 8) and draw some conclusions about the issue of robustness (Section 9).

2 Minimising Overheads in the Engineering Process

GATE (Cunningham 01) is an architecture, a framework and a development environment for Language Engineering. As an architecture, it defines the organisation of an LE system and the assignment of responsibilities to different 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.

From the architectural viewpoint, the GATE desiderata include:

- theory-neutral data types for LE;
- a component-based model of language processing modules;
- a unified model for managing data storage;
- support for LE applications, technologies, methods and components.

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:

- data structures for common information types used in LE tasks;
- transparent access to the data, regardless of its physical location (file system, relational database or the internet);
- facilities for export and import of data to and from various mainstream formats (including XML);
- a component model and facilities for loading, initialising and activating such components;
- facilities for automatic discovery and loading of language data, processing or visual LE resources over the web;
- automatic activation of the appropriate visual components according to the data type.

A set of reusable LE modules is provided with the backplane, which are able to perform basic LE operations such as tokenisation, gazetteer lookup, morphological analysis, part-of-speech tagging, and finite state transduction of the annotations on a document. The framework is open to new types of data, processing resources or visual components, which can be easily added and integrated into new or existing systems.

GATE can help improve the predictability of the LE system creation process in the following ways. During the design phase, the **architectural elements** are used as a guidance on the overall shape of the system. The **framework** helps during the development phase by providing ready-made implementations for parts of

the architecture and solutions for commonly occurring tasks. Finally, the **development environment** is used for convenient ways of exploiting the framework. The inclusion of **ready-made modules** for basic tasks eliminates the need to keep re-inventing them, and provides a good starting point for any new system. For tasks that are not covered by the existing components, new modules need to be designed and developed. The **framework** allows the language engineer to concentrate on finding a solution for the problem at hand, rather than solving other trivial issues, while the **development environment** helps with the overall development and debugging process for the new module.

The **component-based model** allows for easy coupling and decoupling of the processors, which makes it straightforward to compare alternative configurations of the system or different versions of the same module, while the availability of facilities for **easy visualisation of data** at each point during the development process helps in interpreting the results immediately.

GATE makes a clear distinction between the algorithms used and the data required (such as grammars or rule sets), so the two can be developed independently by language engineers with different types of expertise, e.g. programmers and linguists. 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. Work on standard ways to deal with XML data is relevant here, such as the LT XML work at Edinburgh (Thompson & McKelvie 97), as is work on annotation standards, such as the ATLAS project (an architecture for linguistic annotation) at LDC (Bird *et al.* 00).

2.1 Multilinguality

In recent years, the emphasis on multilinguality has grown, and important advances have been witnessed on the software scene with the emergence of Unicode as a universal standard for representing textual data. Most operating systems now support Unicode, which is the first step towards truly multilingual applications. Current requirements for NLP systems demonstrate clearly that, as a field which deals with human language, LE must situate itself at the forefront of mutilinguality.

GATE supports multilingual data using the Unicode standard as its default text encoding, which means that text in many languages can be stored and processed. Apart from being able to process Unicode data, GATE provides a means for entering text in various languages, using virtual keyboards where the language is not supported by the underlying operating platform (see Figure 1). Currently 28 languages are supported; because GATE is an open architecture, new virtual keyboards can be defined by the user 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. Currently the latest Java versions can handle a wide range of languages and more are planned to be added in future releases. 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

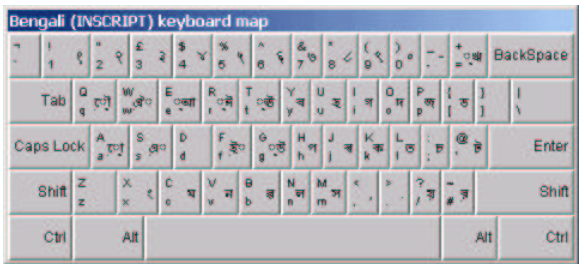

Fig. 1. The Bengali virtual keyboard

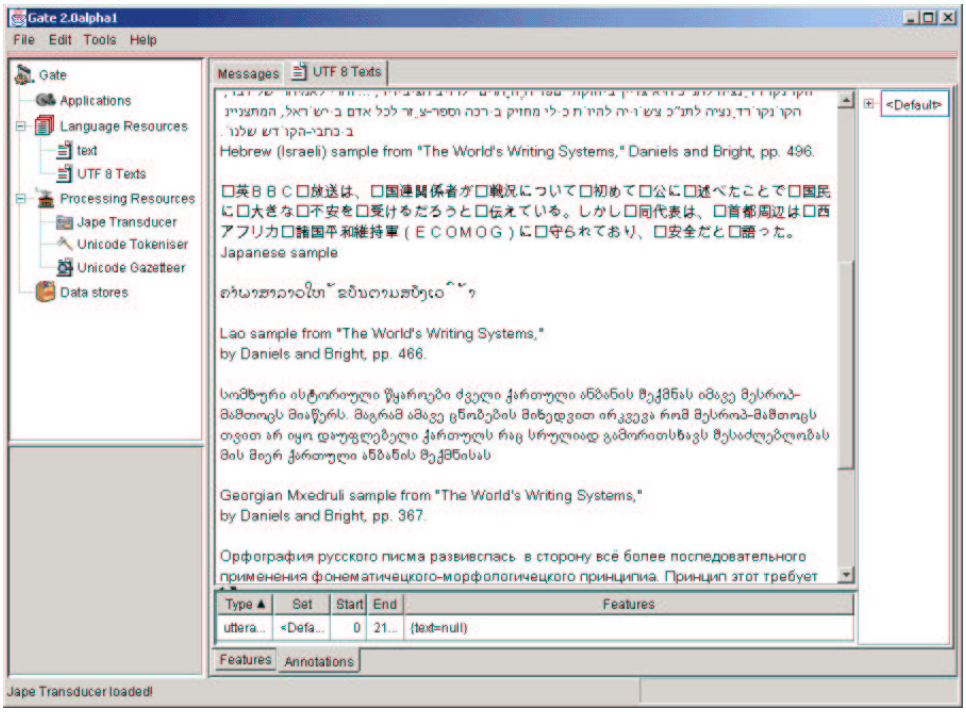

Fig. 2. Unicode text in Gate2

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.* 00), which focuses on constructing a 63 million word electronic corpus of South Asian languages, especially those spoken in the UK.

3 Performance Measurement

Automated performance evaluation is a valuable element of any LE system, and is particulaly important where predictability of performance is required. In this section, we shall describe the evaluation tool developed within the GATE architecture, known as the AnnotationDiff tool.

The AnnotationDiff tool enables two sets of annotations on a text to be compared. For each type of annotation, figures are generated for precision, recall, F measure and false positives. In this way, a text annotated by the system can be evaluated against a key text (ie. the same text correctly annotated by hand), or two versions of the system can be compared with each other to evaluate the impact the changes have made on the overall performance. The former is useful for any system; the latter is particularly useful when dealing with issues of robustness, in order to ensure that changes to one part of the system or to one set of rules does not negatively impact another.

In AnnotationDiff, the annotations are compared based on API methods defined in the gate.Annotation interface. These methods are: isCompatible(), isPartiallyCompatible(), overlaps() and coextensive().

- Two annotations are *coextensive* if their offsets are the same (they hit the same span of text).
- Two annotations are said to be *compatible* if they are coextensive and the features of one are included in the features of the other.
- Two annotations *overlap* if they share some part (or all) of a text span.
- Two annotations are said to be *partially compatible* if they overlap and the features of one are included in the features of the other.

The AnnotationDiff tool allows the user to select the relevant texts, annotation sets and annotation types to be considered, and to set the variables for the F measure and false positives. False positives are a useful metric when dealing with a wide variety of text types, because it is not dependent on *relative document richness*[2] in the same way that precision is.

When comparing different systems on the same document set, relative document richness is unimportant, because it is equal for all systems. When comparing a single system's performance on different documents, however, it is much more crucial, because if a particular document type has a significantly different number of any type of entity, the results for that entity type can become skewed. Compare the impact on precision of one error where the total number of correct entities = 1, and one error where the total = 100. Assuming the document length is the same, then the false positive score for each text, on the other hand, should be identical.

The metrics are defined as follows:

$$Precision = \frac{Correct + 1/2Partial}{Correct + Spurious + 1/2Partial} \quad (1)$$

[2] By this we mean the relative number of entities of each type to be found in a set of documents.

$$Recall = \frac{Correct + 1/2Partial}{Correct + Missing + 1/2Partial} \tag{2}$$

$$F - measure = \frac{(\beta^2 + 1)P * R}{(\beta^2 R) + P} \tag{3}$$

where β is a value between 0 and 1 reflecting the weighting of P vs. R. If β is set to 0.5, the two are weighted equally.

$$FalsePositive = \frac{Spurious}{c} \tag{4}$$

where c is some constant independent from document richness, e.g. the number of tokens or sentences in the document.

Note that we consider annotations to be partially correct if the entity type is correct and the spans are overlapping but not identical. Partially correct responses are normally allocated a half weight.

The AnnotationDiff viewer displays each set of annotations, marked with different colours (this is similar to 'visual diff' implementations such as in the MKS Toolkit or TkDiff). In the response annotation set, annotations have three possible colours according to their state:

- green for compatible ones;
- blue for partially compatible ones;
- red for spurious ones.

In the key annotation set, annotations only have two possible colours, again according to their state:

- white for those annotations that have a corresponding compatible or partially compatible annotation in the response set;
- orange for those annotations that are missing from the response set.

In the viewer, two annotations will also be positioned on the same row if they are coextensive. If not, they are on separate rows. In both cases, they are coloured according to their state.

4 Robust Entity Recognition

Most Information Extraction (IE) systems (Cowie & Lehnert 96; Appelt 99; Cunningham 99b) are designed to extract fixed types of information from documents in a specific language and domain. To increase suitability for end-user applications, IE systems need to be easily customisable to new domains (Soderland 97). Driven by the MUC competitions (e.g. (Sundheim 95; Sundheim 98)), work on IE, and in particular on named entity recognition (NE), has largely focused on narrow subdomains, such as newswires about terrorist attacks (MUC-3 and MUC-4), and reports on air vehicle launches (MUC-7). In many applications, however, the type of document and domain may be unknown, or a system may be required which will process different types of documents without the need for tuning.

Many existing IE systems have been successfully tuned to new domains and applications - either manually or semi-automatically – but there have been few advances in tackling the problem of making a single system robust enough to forego this need. The adaptation of existing systems to new domains is hindered by both ontology and rule bottlenecks. A substantial amount of knowledge is needed, and its acquisition and application are non-trivial tasks.

For systems to deal successfully with unknown or multiple types of source material, they must not only be able to cope with changes of domain, but also with changes of *genre*. By this we mean different forms of media (e.g. emails, transcribed spoken text, written text, web pages, output of OCR recognition), text type (e.g. reports, letters, books, lists), and structure (e.g. layout options). The genre of a text may therefore be influenced by a number of factors, such as author, intended audience and degree of formality. For example, less formal texts may not follow standard capitalisation, punctuation or even spelling formats. The MUSE project aims to identify the parameters relevant to the creation of a name recognition system robust across these types of variability.

The ACE program[3] proposes a step towards resolving the robustness issue by promoting:

1. faster system development from given linguistic resources, which encourages the development of general-purpose retargetable systems, using a variety of methods from richly annotated corpora;
2. the design of more general-purpose linguistic resources;
3. the development of general-purpose standalone systems.

The ACE entity detection and tracking (EDT) task goes beyond existing named entity recognition tasks, in that all mentions of an entity (in the form of a name, description or pronoun) must be recognised and classified (based on reference to the same entity). Although, as with MUC, the texts to be used for the tasks are newswires, the scope of the task is widened by measuring results not only on standard written texts, but also on texts produced from ASR and OCR output.

5 Processing Resources

The MUSE system is based on ANNIE, A Nearly-New IE system. ANNIE borrows a number of ideas from LaSIE (Gaizauskas *et al.* 95) (and a Brill-style part-of-speech tagger (Hepple 00; Brill 92)), but also contains a number of new developments[4], such as reliance on finite state algorithms and the JAPE language (Cunningham *et al.* 00). Figure 3 depicts a full IE pipeline based on a LaSIE backend with ANNIE shallow analysis.

The MUSE system comprises a version of ANNIE's main processing resources: tokeniser, sentence splitter, POS tagger, gazetteer, finite state transduction grammar and namematcher. The resources communicate via GATE's annotation API,

[3] http://www.itl.nist.gov/iaui/894.01/tests/ace/
[4] New in the current context that is, not necessarily in the field of IE itself.

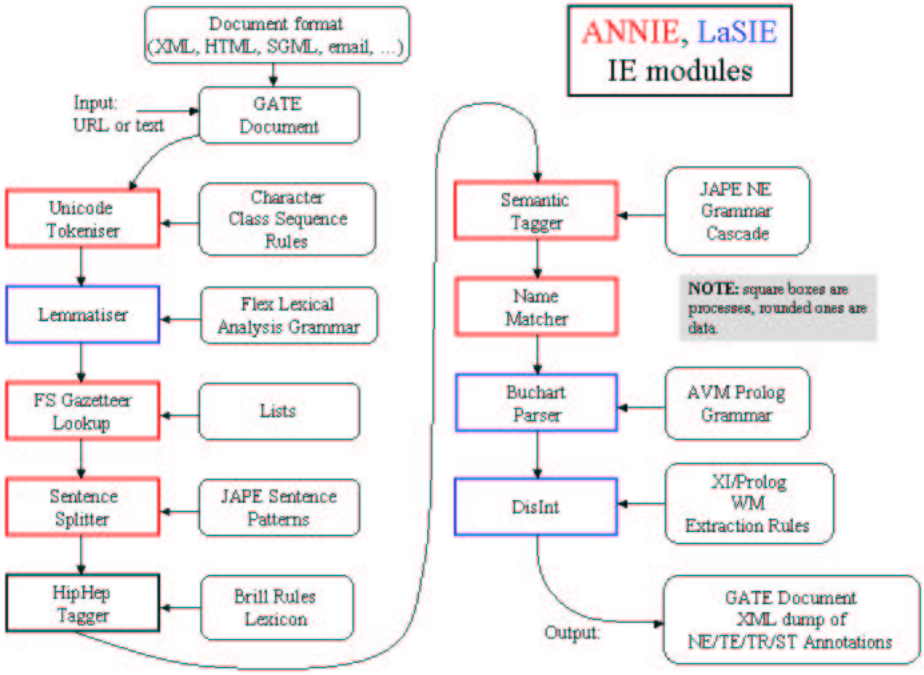

Fig. 3. ANNIE, A Nearly-New IE system

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 **grammar** consists of hand-crafted rules written in JAPE, which describe patterns to match and annotations to be created as a result. Patterns can be specified by describing a specific text string, or annotations previously attached to tokens (e.g. annotations created by the tokeniser, gazetteer, or document format analysis). Rule prioritisation (if activated) prevents multiple assignment of annotations to the same text string. Section 7.1 describes how the grammar rules can be adapted to deal with different text types.

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

5.1 Implementation

The implementation of the processing resources is centred on robustness, performance, 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 **tokeniser** is implemented as a finite state machine (FSM) that uses the classes of characters defined by the Unicode 2.0 specification as input symbols and outputs annotations on the document being processed. The use of Unicode-defined categories allows for generality, as the same tokeniser can be used to process text in virtually any language. We have successfully tested it on Western and Cyrillic languages, and the EMILLE project (McEnery *et al.* 00) is using it for processing Indic languages.

The **gazetteer** is also implemented as a FSM which is built at initialisation time starting from the list of phrases that need to be recognised. It runs directly over the text being processed and so it does not depend on any other processing resource. Again, the gazetteer is capable of handling Unicode input which makes it usable for text in any language.

The third type of processing resource used is a **Jape transducer**, (Java Annotation Patterns Engine – (Cunningham *et al.* 00)) which handles most of the workload in the Named Entity recognition system. It is implemented as a cascade of phases, each phase being a finite state transducer. The transfer of temporary results between phases is done via a GATE document, which is the only data source shared between the phases. One of the advantages of this approach is that the results of one phase can then be used by any number of the subsequent phases, or can constitute output of the whole system.

The way Jape transducers work can best be described as a regular expression matching mechanism that uses a directed graph of annotations as input. So far, we have successfully used Jape for named entity recognition and sentence splitting, and

we intend to experiment with it in other fields such as shallow syntactic parsing. The phases that compose a multiphase Jape transducer are independent entities, which means they can be reordered, reused in other applications, or even executed in parallel if they do not depend on each other's results. Like the other processing resources, the Jape transducer is capable of handling Unicode input.

Although at the moment we are only using hand crafted rules, it would be possible for an application to learn rules automatically for the Jape transducers in a manner similar to (Day *et al.* 97). These rules could then be verified or ammended by a human if necessary, as they are human readable.

The fact that all the processing resources employed are using the FSM 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 2.5 Kb of text per second on a PIII 450 PC with 256 mB RAM (independently of the size of the input file; the processing requirement is linear in relation to the text size), and we hope to improve on this figure in future.

6 The MUSE system

As outlined in Section 4, the main objectve of the MUSE project is the development of a named entity recognition system that is capable of processing texts of widely differing, and possibly unknown, genre. We define genre in very loose terms to encompass issues such as style, text type and domain.

6.1 Entity Types

The current system aims to identify the same types of entity as detailed in the MUC guidelines, but with two additional types: address and identifier. The entities and sub-entities are as follows:

- **Entity**: organisation, person, location
- **Time**: date, time
- **Number**: money, percent
- **Address**: email, url, telephone, ip
- **Identifier**

We have largely followed the MUC-7 guidelines for the definition and markup of entities (Douthat 98), but we have made a few changes in order to remove some anomalies and make the entities found more practical for further applications. For example, we include the title of a person in the markup, e.g. we annotate [Dr. John Smith] rather than Dr [John Smith]. We also combine sub-types of entity (e.g. combinations of dates and times which occur consecutively are annotated as dates). Like its predecessors with an open architecture and hand-crafted rules, the system can easily be extended to deal with different entity types, or modified to take account of new guidelines.

Domain	Format	Type
Natural/Pure Sciences	written	book
	email	mailing list
Computing	email	mailing list
Commerce/Finance	written	periodical
	spoken	monologue
Education	spoken	monologue
World affairs	written	misc.
Social sciences	written	misc.
Public/Institutional	spoken	dialogue
Imagination	written	misc.
Arts	written	misc.

Table 1. *Composition of Corpus*

6.2 Data

For training and testing purposes, we have compiled a corpus containing texts which are diverse in terms of domain, format, style and genre. This aims to ensure that the system can cope adequately with a variety of text types. The data comes from 3 sources: a subset of the BNC (British National Corpus) (Burnard 95) comprising both spoken and written text; a set of emails from a medical mailing list; and a set of emails from a computer helpdesk. The corpus is subdivided as shown in Table 1.

Processing the spoken corpus material is an easier task than processing the output of a speech recogniser, as the BNC transcriptions have capitalisation and punctuation added. In the MUMIS Multi-Media Indexing and Search project we are applying similar technology to the noisier output of speech recognisers; see (Declerck *et al.* 01) for more details. The ACE program, as we have mentioned, also requires information extraction from degraded textual data (the output of both ASR and OCR systems). Nevertheless, the MUSE spoken corpus still presents challenges to the system, for reasons explained in the following section and depicted in Table 2.

7 Processing different text types

The MUSE Named Entity recognition system is designed to process multiple types of text in a robust fashion, with minimal adaptation. It is hard to generate this kind of robustness in a system without sacrificing specificity (and thereby either precision or recall, or both). To overcome this problem, the system is designed so that it can be adapted to the situation through the use of a set of resource switches, which operate according to certain linguistic or other features of the text. For example, information about the domain of the text may cause the system to turn on or off a specific set of gazetteer lists related to that domain. Similarly, information about the text format may require different grammar rules to be used in order to preserve or ignore the layout of the text (e.g. addresses in letters and emails). In Table 2 we give an example of some features of different text formats which have an impact on our core NE recognition system.

	Written	Spoken	Email
Line Breaks	control char replaces space	control char replaces space	control char and space
Spacing	no extra spaces	some extra spaces	some extra spaces
Other spacing	none	none	reply separators
Spelling	few errors	some errors with names; stumbles etc. mid-word/entity	errors with all words
Punctuation	mostly correct	some missing	frequent spurious and missing
Capitalisation	mostly correct	some missing capitals	missing and extra capitals
Numbers	as figures	as words	as figures
Abbreviations		interspersed with spaces	

Table 2. *Features of different text formats*

7.1 Adapting the resources to the text type

So far, we have identified a number of features of different text types which require adaptation to the processing resources. Although changes may be necessary to both the grammars and gazetteer lists, the adaptation is only required in the grammars themselves, because the gazetteer lists are designed in such a way that they can be manipulated in different ways from the grammar. When calling for entries found in a gazetteer, we can specify a broader or narrower set, depending on our requirements (e.g. we can specify that military titles are to be included or excluded as part of a set of general titles). Currently, the correct resource set for a particular text type must be manually loaded, although it is intended to automate this facility by means of a lookup table associating certain textual characteristics with specific grammars.

Below we outline some examples of the switches we use to deal with different types of text format and domain.

7.1.1 Email-specific requirements

Emails tend to be the least predictable type of text in structural terms, because they may be very well structured or not at all. Much may depend on the particular email program used to produce the original text - for example, if line breaks are forced. They are also wildly different in terms of formality, which has an impact on features such as use of punctuation, correct spelling etc. However, there are certain clues given by the email header information which can be used to assist with processing. The grammar used for emails varies in two important ways from that used for other types of text.

1. More flexibility is permitted for emails regarding the use of spaces and control characters. This includes the use of the reply separator ">".
2. An extra grammar is used which processes header information (e.g. the "to:" and "from:" lines) and some information about hostnames and specifications (also usually found in the header lines).

7.1.2 Email and spoken text requirements

Both emails and (transcriptions of) spoken texts may be considered in some way as "degraded input" in that they do not necessarily conform to correct usage of English. In particular, capitalisation, punctuation and spelling norms are not always obeyed. Punctuation and spelling are issues that have not been fully tackled as yet. However, grammars for email and spoken texts both have a switch which turns on the use of names with no initial capital letter. By default, this is only fired when the name in question is not ambiguous with a common noun, although this can be overridden in the case where context makes it clear that a name is being used (e.g. following words such as "Dear" in a letter or email).

7.1.3 Scientific texts

In scientific texts, single initials (e.g. A, B, C, D etc.) are often used, for example when referring to points on a graph. In such situations, we attempt to recognise these, in order to prevent them being identified as part of a person or company's name. From analysis of the sample texts, we also find that most unknown proper nouns are names of people, so we set the default unknown switch to Person.

7.1.4 Religious texts

Although it may not seem intuitive that religious texts require any special treatment, they tend to involve sets of names not commonly used elsewhere (and which might have different meaning in other situations). For example, it is likely that names such as "God" and "Jesus Christ" found in non-religious text are being used as expletives rather than as real references to entities, whereas in religious texts we can be fairly sure they are being used to represent people. We therefore have a switch for religious texts which turns on the use of specific gazetteer lists for names of biblical people and places.

Clearly there are many other text types and features that would benefit from special treatment, and we have only outlined some examples above, which are most relevant to our small test set described below. Work is currently being undertaken to develop automated procedures for identifying textual characteristics. There exist already a number of (mainly statistical) procedures for text classification and genre identification, e.g. (Biber 89; Kessler *et al.* 97; Karlgren 94) although our goal in this is somewhat different, as we do not require classification as an ultimate goal, but only as a means to an end.

8 Evaluation

The system was tested on texts drawn from the test corpus, after minimal training on similar texts. It was also tested using the standard core set of resources to provide a baseline for evaluating the genre adaptivity features, as opposed to the specific resource set for that domain and text type. The test texts were split into 4 groups: medical emails (EML), spoken miscellaneous texts (SPOMISC), written scientific texts and spoken religious texts (SPOREL). Each group consisted of 4 randomly chosen texts.

The sample texts were evaluated according to precision, recall, and F-measure using GATE's AnnotationDiff tool, according to the formulae described in Section 3. The weighting for the F-measure was set to 0.5, i.e. equal preference was given to precision and recall. Although we could also have measured False Positives as an alternative to Precision, we found that our document test set was not significantly unbalanced enough in terms of document richness to bias the precision weightings unduly. We therefore preferred the use of Precision since it is a more widely recognised metric.

8.1 Results

The results depicted below are averages for each group of texts.

Figure 4 shows the precision, recall and F-measure by entity type for the SPOMISC group. The results are consistently high, with percentages and addresses achieving perfect scores of 100%. This is in line with MUC experiments, where the best systems achieved name recognition in the high 90s.

Figure 5 depicts the average precision, recall and F-measure for each group of texts. Somewhat surprisingly, email texts scored the highest here, with the miscellaneous spoken texts achieving slightly lower scores, and the religious texts scoring the lowest. However, given that more time has been spent on tuning the resources to emails, and very little time training on the religious domain, it is not so unexpected.

Finally, Figure 6 shows how the F-measure varies when a specific grammar set is used rather than the standard set. For emails there is very little difference, but this is perhaps coincidental, since a small number of test texts were used, and we might expect to find greater differences with other email texts. For religious texts there was a slight improvement, mainly for the Person entity. For scientific texts there was a marked difference, largely again in the Person entity.

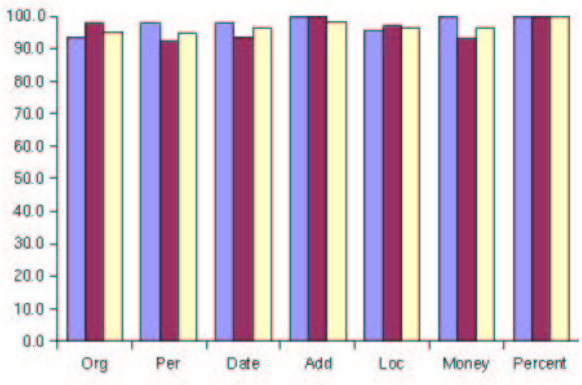

Fig. 4. Average results by entity type for spomisc

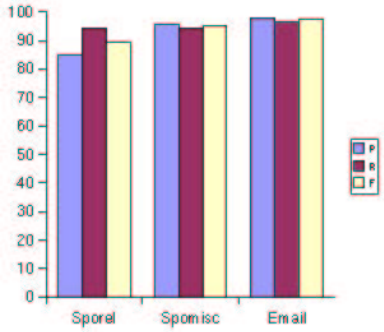

Fig. 5. Average results by text type

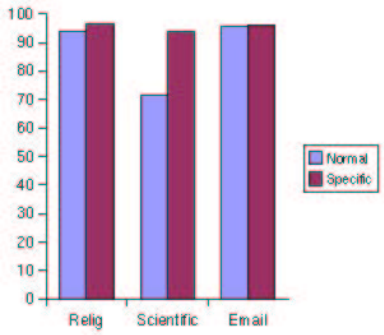

Fig. 6. Average F measure using different grammars

We shall discuss below in more detail the results for the different entity types found in each type of text.

8.1.1 Business monologues

On these spoken texts in the general domain of business, the system scored 100% precision and 93.3% recall, only producing errors for 2 entity types - person and date.

8.1.2 Religious monologues

On this text, the system scored well when used with the religious set of resources, falling down slightly on organisation and person. This was mainly due to wrongly spelled words and spacing errors (e.g. no space between two words). When this text was tested with the core grammar and gazetteer, precision fell very slightly and recall dropped a couple of points. This was largely due to religious names not being recognised with the standard grammar.

8.1.3 Scientific books

The results for the scientific books were slightly lower than for the religious texts, with Person scoring low on recall (63%). This was largely because of surnames being used on their own, without contextual clues. There was, however, a marked improvement in both precision and recall over the same text being processed with a standard set of resources, as discussed earlier.

8.1.4 Medical emails

The email texts also performed well, achieving 100% precision and recall on dates, and 100% precision on organisations, although they fell down slightly on recall of locations (due to typographical errors). There is little difference between using the standard grammar and the email-specific grammar, although we might expect this difference to be more noticeable with texts which make more use of certain stylistic features such as address layout.

9 Conclusions and Further Work

In this paper we have presented the GATE architecture and framework for Language Engineering, and the MUSE cross-genre Information Extraction system developed within GATE. We believe that our experiences with these systems demonstrate that robustness of LE systems can be significantly increased by the provision of dedicated infrastructural software for data storage and visualisation, automated performance measurement and component-based development and deployment. In this way the work reported here is a contribution to the *engineering* of robust systems with both *predictable performance* and *predictable development resource* requirements.

The results from the evaluation of the MUSE system also indicate that named-entity extraction robustness in face of multiple genres is feasible at performance rates comparable with single-genre systems. The results so far look very promising, in that the system is able to achieve high recall and precision scores with minimal alterations to the processing resources. At the same time, it is clear that these alterations are important to the overall success of the system. The next major step will be to automate the process of determining which set of resources to use in different cases. We also plan to improve the scope of the system and decrease time spent developing new rules by investigating methods of learning new rules, for example by detecting useful contextual information automatically.

References

D. Appelt. An Introduction to Information Extraction. *Artificial Intelligence Communications*, 12(3):161–172, 1999.

D. Biber. A typology of English texts. *Linguistics*, 27:3–43, 1989.

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.

E. Brill. A simple rule-based part of speech tagger. In *Proceedings of the DARPA Speech and Natural Language Workshop*. Harriman, NY, 1992.

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

J. Cowie and W. Lehnert. Information Extraction. *Communications of the ACM*, 39(1):80–91, 1996.

H. Cunningham. A Definition and Short History of Language Engineering. *Journal of Natural Language Engineering*, 5(1):1–16, 1999.

H. Cunningham. Information Extraction: a User Guide (revised version). Research Memorandum CS–99–07, Department of Computer Science, University of Sheffield, May 1999.

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. *[in press]*, ??(??):??, 2001. Accepted for publication by Computing and the Humanities, May 2001.

H. Cunningham, M. Freeman, and W. Black. Software Reuse, Object-Oriented Frameworks and Natural Language Processing. In *New Methods in Language Processing (NeMLaP-1), September 1994*, Manchester, 1994. (Re-published in book form 1997 by UCL Press).

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

H. Cunningham, D. Maynard, and V. Tablan. JAPE: a Java Annotation Patterns Engine (Second Edition). Research Memorandum CS–00–10, Department of Computer Science, University of Sheffield, November 2000.

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 NLP Systems (ANLP-97)*, 1997.

T. Declerck, P. Wittenburg, and H. Cunningham. The Automatic Generation of Formal Annotations in a Multimedia Indexing and Searching Environment. In *Workshop on Human Language Technology and Knowledge Management*, page ??, Toulouse, France, 2001. `http://www.elsnet.org/acl2001-hlt+km.html`.

A. Douthat. The message understanding conference scoring software user's manual. `http://www.itl.nist.gov/iaui/894.02/-related_projects/muc_sw/muc_sw_manual.html`, 1998.

R. Gaizauskas, T. Wakao, K. Humphreys, H. Cunningham, and Y. Wilks. Description of the LaSIE system as used for MUC-6. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*. Morgan Kaufmann, California, 1995.

M. Hepple. Independence and commitment: Assumptions for rapid training and execution of rule-based POS taggers. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL-2000)*, Hong Kong, October 2000.

D. Karlgren, J.and Cutting. Recognising text genres with simple metrics using discriminant analysis. In *Proc. of COLING 94*, Kyoto, Japan, 1994.

B. Kessler, G. Nunberg, and H. Schütze. Automatic detection of text genre. In *Proc. of 35th Annual Meeting of the Association for Computational Linguistics*, Madrid, Spain, 1997.

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

R. Pressman. *Software Engineering, a Practitioner's Approach (European Edition)*. McGraw Hill, New York, 1994.

M. Shaw and D. Garlan. *Software Architecture*. Prentice Hall, New York, 1996.

S. Soderland. Learning to extract text-based information from the world wide web. *Proceedings of Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*, 1997.

B. Sundheim, editor. *Proceedings of the Sixth Message Understanding Conference (MUC-6)*, Columbia, MD, 1995. ARPA, Morgan Kaufmann.

B. Sundheim, editor. *Proceedings of the Seventh Message Understanding Conference (MUC-7)*. ARPA, Morgan Kaufmann, 1998.

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