Extending a Message Understanding System

Susan Smith

A dissertation presented in part fulfilment of the requirements of the Degree of MSc in Information Technology at the University of Glasgow

September 2007

Abstract

An extension to an already existing Information Extraction system was created and integrated, extending its functionality to cope with a greater range of input. The application processes short email and text messages on a given domain, with the aim of updating a data repository. Intended for use within a Web application for collaboratively developed websites, the program originally handled only sentences having their predicate in the form of the verb 'to be'. The main purpose of this extension was to equip the application with the ability to process sentences having more variation in the verb structure. A new set of language structures have been incorporated, and are subject to the same range of processing as the original set, whose handling also remains fully functional.

The application employs a Pattern Matching approach to the Extraction process, comparing incoming sentences with a set of patterns and creating update statements to the data based on interpretation of these. The new structures conform to the same format as those already existing, and are subject to the same processing stages, although with some tailor-made algorithms. The extension has been developed in conjunction with the lexical resource WordNet, and uses externally devised Java interfaces to this program. An aspect of the work carried out for this Project has been to research and experiment with these facilities.

Additionally, a period of research and exploration was undertaken to perform the necessary groundwork for enabling the application to recognise different forms of tense inflection. This phase of the Project was primarily research oriented, although the submitted program contains access to a resource developed as part of this work.

The application is expected to be under continual development over the coming time.

Acknowledgements

I would like to thank Dr Richard Cooper for his advice, support and patience over the course of this Project.

I would also like to extend my gratitude to those people who contributed to the Evaluation process by sending me short messages.

Contents

Appendices

1 Introduction

The present Project comprises an extension to an Information Extraction system that increases its range of processable data. The system extended extracts information from short email or text messages and uses this information to update a database. The application adopts a 'lightweight' approach to the Extraction task, and is intended for use within Web applications for collaboratively developed sites. The idea is that the site would hold data related to a certain domain, and that contributors would send short messages indicating additions or amendments to that data. A basic database, modelling information related to the domain, is therefore systematically updated by the application, with information gleaned from the incoming messages. For example, given a domain in which the information held is about books and authors, a message submitted might indicate the name of the author for a certain book. The existing system is further explained in section 2.2 of the Survey chapter.

The application employs a pattern-matching technique. A set of Templates define the structures of sentences that are of interest to the extraction process. These Templates are then used to generate a larger set of Patterns, which are realisations of the Template structures. The sentences in the incoming message are then compared in turn to these Patterns, to determine whether or not they match; if so an update to the database is created and executed, using the information retrieved from the message.

At the start of the program run, a number of text files are read in, informing the application with the structures and database, as well as a definition of the relationship between these. The interpretation of these files equips the system with the ability to 'interpret' incoming messages. This interpretation consists of the way in which matched incoming structures result in data updates. In other words, the application itself is a definition of how incoming language excerpts are to be 'understood' in terms of the database. Although the system does indeed accord to a 'lightweight' approach within the Information Extraction context, it does employ a number of complex algorithms and is motivated by many subtle linguistic observations.

The details of the existing application, as well as an introduction to the field of Information Extraction are provided within chapter 2. In terms of the Project itself, an overview of the Analysis and Design aspects feature in chapter 3, the Implementation details are discussed within chapter 4, a series of linguistic concerns and justifications are listed within chapter 5, and Testing and Evaluation details in chapters 6 and 7. A number of resources are also provided within the appendices for this paper, as indicated throughout the chapters that follow.

1.1 Aims

The main aim for this Project was to extend the number of language structures that the system is capable of handling. Presently the application handles only sentences having their predicate in the form of the verb 'to be'. The brief for the present extension was to equip the application with the ability to process sentences with other verb forms.

The development of the application has always been from the position that there is a certain set of information that is of interest to the system, which is modelled by the database and the domain at hand. Any sentences featuring information other than this are simply ignored. The assumption for the present Project therefore was that the structures featuring varied verb forms that were to be incorporated would similarly be those that realise this type of information. Building on this assumption, the target structures were taken from the outset to be an alternative formulation of those already handled by the system. For this reason a constraint was accepted that can be framed in the following way: the additional structures should ideally be handled without extending those parts of the application that deal directly with the database. In order to deliver this objective, it was decided at an early stage that the focus of the work would be to 'translate' a new set of language structures into those that are within the set already handled by the application. The first stage was therefore to arrive at the set of language structures at which the extended functionality would be aimed. In terms of the application, these structures would necessarily be defined in the form of new Templates. Please see Appendix B for the list of Template structures.

Once the set of new Templates had been established, the work of implementing the ability for the system to process them could begin. However the initial stages of the implementation happened to require a substantial period of research and exploration, as is discussed in sections 4.1-4.3 of the Implementation chapter. Many of the complexities within this project lay not only in the subtle and often complex linguistic implications of the work, but also in the range of external tools employed to bring about the successful deployment of the extension.

A secondary aim for the Project was also laid out in the early stages. The assumption was accepted from the outset that the type of information to be handled by the extension would be time invariant. (Time invariant information is that whose truth is not dependent upon the context of its expression, for example, the author of Emma was Jane Austen 100 years ago, just as it is now, whereas the Prime Minister of Great Britain is Gordon Brown now, but was not a mere few months ago, because that is time variant information.) This means that, for the purposes of the extension, the tense used within the incoming sentences is essentially irrelevant. In this sense it was reasonable enough to treat all tense variations within the new structures

accounted for as though they were the same. However, it was implied that future extensions to the system might require the recognition of different tense variations, in circumstances where the tense would in fact carry some informative implication. These would be situations in which the system would be required to detect the differences in tense encountered, and to interpret these differences accordingly; to recognise the informative value of tense and to follow this through to appropriate amendments of the data. A recent extension to the application (Cooper and Manson 2007) dealt with time variant information in terms of the data model, although naturally tense was not a factor in this work since the application had not yet been extended to deal with the verb structures now incorporated.

In order to provide some starting point for this work, a secondary aim for the Project was to find some way of realising tense recognition in the system. Specifically, this aim was to find a way to generate the various inflections of a word on querying with its base (infinitive) form. It had been discovered during the early development period for the extension, that the WordNet Java interfaces used in the application were equipped with the ability to return an item's base form on querying with an inflected version. However, it was also established that the reverse was not catered for. The process can only be executed in one direction, since WordNet stores only the base forms, and resolves inflections into these using tailored algorithms during the program run.

The approach taken was therefore to explore the feasibility of building a repository of base and inflected forms for use within the application. This element of the Project was, however, unexpectedly complex, although it has hopefully performed the necessary exploration for such a facility to be further developed and used in future work. A repository was indeed built, using the WordNet dictionary files and other utilities, outlined in section 4.4 of the Implementation chapter. While it does form part of the present Project, this utility is somewhat separate to the main area of work, in that it is not actually used by the extended application other than for demonstration purposes.

The Appendices to this dissertation provide further details of the extension and resources used within the Project.

2 Survey

The application under development falls within the broad category of Information Extraction. For this reason, a brief introduction to the field follows in section 2.1, before the application itself is explored in 2.2, and the lexical database WordNet, which has been used within the present Project, is discussed in section 2.3. Please see the Proposal document for this Project in Appendix A for more details on these topics.

2.1 Information Extraction

Information Extraction is the process whereby information that is relevant to a specific purpose is derived from a textual source. The information will normally then be held in some structured format, or indeed go on for further processing. In contrast with other language processing activities, it is not generally in the interests of IE systems to achieve a full syntactic analysis of the text, but rather to focus on extracting only the information that is required for some specified need. The nature of this information that is relevant will therefore be defined in advance, as will the format in which it is required. Many IE facilities are developed with a particular domain and ultimate data repository in mind. The extent to which IE systems attempt to grammatically parse input text varies greatly, as does their portability between domains. See Cunningham (1999) for a coherent introduction to IE.

While Information Retrieval accomplishes the acquisition of documents relevant to some broad information need, IE selects information from within the content of documents. As indicated above, the type and structure of the information sought by an IE system is known in advance. To this end, IE systems generally avoid some of the more labour-intensive linguistic processing activities common within the wider field of Natural Language Processing, such as rigorous grammatical parsing of sentences, which are difficult to perform to high standards of precision. IE systems tend to value robustness and efficiency, a testament to which is the fact that many systems have gone on to enjoy commercial application. Extensive linguistic analysis is therefore neither necessary nor desirable within an IE application, although naturally there has been considerable influence from NLP techniques as the field has developed.

The IE area is often characterised by Projects whose application is somewhat specific, but whose output can be variable in structure. Often systems will comprise a component within a larger application, as with the present Project. IE applications are usually thought of as comprising a number of relatively distinct processing stages, with the design of each, and naturally of the application architecture in general, being necessarily informed by the overall purpose of the activity. The development of an IE application will likely therefore feature a series of design decisions, often requiring

'trade-offs', for example sacrificing recognition of subtlety in order to deliver increased robustness. The degree of manual input also varies greatly according to the context of a project.

The Message Understanding Conferences, which ran from 1987 to 1997, were largely responsible for shaping the direction of IE in its early years. Sponsored by the US Defence Agency DARPA, the Conferences brought together Projects having the same overall IE aims. The MUC process provided standard means to quantify and evaluate the success of an IE system, defined specific goals at which Projects could direct their focus, and served as impetus for the advancement of the field in general. Participation in the MUC event meant developing a system that would perform a specific IE task on a given set of texts, issued prior to the Conference. At an earlier stage, participants would have been supplied with training texts, of the appropriate type and domain, on which they could base the design of their systems. The testing process carried out at the Conferences themselves was formally developed as the conferences progressed, providing a fair reflection of performance across the board. The format of the output of the applications was often also defined in advance, making for a clearer assessment of the relative success attained by participants. Gaizauskas and Wilks (1997) provide a detailed overview of the MUC process.

MUC systems' output was the production of filled templates containing the information that had been extracted from the input texts. These were used to measure performance in terms of Recall and Precision, standards adopted from the field of Information Retrieval but given particular interpretations within IE. Recall measures the amount of information that has been successfully retrieved, whereas Precision calculates the accuracy of this information. As the Conferences progressed the IE task was further divided into subtasks, on which projects could focus. Named Entity Recognition was the identification of objects whose existence is evident within the text. Coreference Resolution was the handling of scenarios on which more than one lexical item within the text is referring to the same object, for example with pronouns referring to a previously mentioned noun. Template Element Production was the association of information with some recognised entity, resulting in data that is in generic form suited to further use, i.e. to fill Templates with. Template Relation Production and Scenario Template Extraction further refined the relationship between elements of the information and their format, in preparation for further processing.

In terms of IE system architecture, approaches are varied. However, one structure has prevailed to a degree within the field. As mentioned above, there will typically be a series of basic stages of processing within an IE application, although the methods employed to deliver each may vary greatly. Generally, each stage will inform the next, exerting extra pressure on the performance at each phase. Firstly, the incoming

document will likely have to be tokenised in some way, arriving at a set of units whose size will be determined by the particular needs of the application. Most systems will then employ some form of tagging technique on these units, marking-up the text for the next stage of processing. The accuracy of this stage will depend largely on the quality of the training period undergone by the component during its development phase. Systems that are aimed at processing within a relatively narrow domain are also likely to enjoy greater success in this phase than those with a broader remit. Next, any featured syntactic analysis is carried out; this will typically be 'shallow' or 'lightweight' in comparison with NLP systems, focussing on those ingredients that are presumed to be of significance (e.g. certain parts of speech) and ignoring the rest. Contextual issues will generally be resolved next, for example, coreference and anaphora as mentioned above. Often, a template structure, reflecting some predefined conceptual categorisation of the 'entities' and their relationships in the domain, will be filled with the extraction results. These templates may then go on to be further processed, for example to populate a database.

The success of an IE system can depend on a number of factors, including the complexity of either the input text or the expected output, the domain, how appropriately the structures for output have been chosen to represent the information, and so on. Many IE systems incorporate hand-crafted rather than automatically generated structures. Hand crafted elements will normally be produced by domain experts, however the gain in reliability is balanced with the resources required during the development period. Other systems employ learning algorithms, which are more amenable to change, but whose success depends largely on the training data used.

There is no one ideal IE system, and the field as a whole is characterised by the need to make appropriate decisions, and systems that are tailor-made for some predefined purpose. The success of IE applications can be difficult to measure, not least because, when it comes to natural language, the correct interpretation of a sentence may be contentious even among speakers of the language!

2.2 The Existing System

The existing system, described by Cooper and Ali (2006), is designed to perform Information Extraction within the context of a Web application. The development of the system was prompted by the need to process input from contributors to information-bearing web sites. In addition to providing structured means to submit information, for example by filling in a form, the application aims to provide the option for users to send a short message containing the information that they wish to submit. The present system is therefore tasked with processing these messages and updating a database holding site information acquired thus far. The data repository is designed to store information that is related to a known domain, and is structured in a way that informs the extraction process. For example, given a domain relating to authors and books, the following message might reasonably be expected:

```
Jane Austen is the author of Emma.
```

Depending on the database Schema being used, the system might then locate the book entitled 'Emma' within the data, or insert it if it isn't already there, and update it to reflect its author as 'Jane Austen'. Within the application, the above message must therefore be recognised and then translated into an update statement (or series of statements) to execute on the data.

The component is therefore required to extract from the message, information that is relevant to the domain, and that can be described in terms of the overall data model. In other words, sentences of interest to the application are those expressing factual assertions, that are realisable in the form of update statements compatible with the database. In relational database terms, the messages will contain data values that are to be set for the Attributes of Entities. This system is then concerned with identifying the data values as well as the Entity and/ or Attribute to be updated, through examination of the Message.

The extraction approach chosen within the system is to pattern-match the incoming message. The application takes as input a series of text files outlining definitions of the data structure and other aspects of a single domain. Among these is the set of Templates to be used in the extraction process. The Templates are definitions of the structures of those message excerpts that are of interest to the system. The Templates are comprised of constant terms, placeholder terms for data values and placeholders for metadata terms, indicating elements in the database Schema. Each Template describes a linguistic structure that matching sentences will be expected to conform to. For example, the following Template definition:

```
8 The <attribute1> of the <entityType1> is <<value1>>.
```

describes the structure of the following sentence:

```
The author of the book is Jane Austen.
```

(within a domain where there is a 'book' entity type which has an 'author' attribute – as in the 'library' domain, see Appendix F). The number listed at the start of the Template definition indicates the number of a Template Type, which is, in turn, a definition of the database update statement that should result in a match for this structure. The application takes these Templates and uses them to generate a set of Patterns, normally several for each Template. The Patterns are potential realisations of the Template structures, for example:

```
The author of the Book is <author>.
```

could be a Pattern generated from the above Template, and matching the above sentence from the message. The term appearing between the angled brackets is a placeholder, which indicates that whatever occurs there is a data value. The process of generating Patterns replaces the metadata terms with some of their possible variations, arrived at through a series of complex algorithms.

The Patterns generated are therefore compared to the sentences contained in the message, and used to instantiate update statements when a match is encountered. The terms occurring at placeholder positions are used as the data values within the update statements, whose form is determined by the Type of the Template from which the matching Pattern was generated. In addition the metadata terms, the application additionally uses lexical criteria such as synonymy to establish the validity of the incoming message content. As in the above example, the only form of predicate currently recognised within the application is the verb 'to be'.

The pattern matching procedure comprises a 'lightweight' approach to Information Extraction, avoiding as it does syntactic parsing of the input text. The approach is justifiable mainly because the structures processed will likely fall within a very narrow range of discourse and conform only to loosely defined informal grammatical structures. There is naturally little benefit to detailed grammatical analysis on sentences other than those in perfectly formed Standard English. In fact, the application is, by definition, only interested in resolving the set of structures framing the range of ways that data updates to the repository might occur in the form of short English language sentences. For this reason, the approach at each stage in its development has been to set about equipping the application with the ability to recognise and process structures containing exactly the type of information that can be sensibly turned into updates. Needless to say, the structures concerned will comprise but a small fragment of the information that it is possible to impart through the medium of natural language.

This notion of capturing information defined for a specific purpose has been central within the field of Information Extraction, as is discussed in section 2.1.

However, traditionally Information Extraction systems have employed, to a greater or lesser degree, some form of grammatical parsing. In this sense, a notion, however abstract, of grammar is often enshrined within the program logic. The parsing within the present system, such as it is, is extremely light, and involves informally defined notions of linguistic structures defined by the Templates. Chapter 5 on Linguistic Considerations goes into the implications of this approach in more detail.

Given that the system is designed to have a very specific purpose within the context of a larger Web application, it is possible to arrive at a definition of the set of information that it is aimed at. The database schema for a domain, in fact, performs this function. Many common language structures can happily be discounted given this, a clear advantage in the development of the system. On the other hand however, the program context also means that incoming structures are likely to conform only to loose, unpredictable conventions, for example in terms of spelling, implying considerable difficulties. Thankfully however, these problems have not presented a particular issue for the present extension, which fits within the bounds of what has already been achieved, building on top of it rather than branching out in new directions.

The initial set of Template structures that have been handled by the system were naturally those translating most intuitively to database terms. Those sentences resembling the natural language equivalent of a data update statement were the obvious starting point for the application. The ability to extend the update facility to other statements has been systematically built on top of these, as the application has developed.

All sentences then, that it is in the interests of the program to resolve, will necessarily be describable in terms of updates. For certain structures the logic to this interpretation process is clearly apparent, for example:

```
The author of Persuasion is Jane Austen.
```

In the above sentence, the equivalent in database terms (assuming the schema definition in the 'library' domain –Appendix F) will update the Entity having 'Persuasion' as its key, setting the author Attribute value to 'Jane Austen'. However, the update ingredients will not always be so immediately evident, in fact there could be many different ways of expressing the information, given the descriptive capabilities of any natural language. As is discussed in the following chapters, the work for the present Project was to translate new varieties of structure into data updates having the same form as those already handled.

The Templates used in the present application essentially comprise the program's grammar, in the sense that they describe the structure of acceptable sentences. The

pattern matching approach spares the application the need to incorporate any more complex syntactic notions. The application also resolves such contextual issues as anaphora using a model of the discourse context. The contextual aspect of the extraction problem is further minimised within the present application, due to the nature of the application context, in which much contextual information can be assumed rather than resolved, a major issue in many extraction systems.

The assumption has been made throughout the development of the current application, that sentences that are relevant are likely to contain terms that are recognisable to the domain. That is, the terms in a candidate sentence for extraction will contain words that are either metadata terms, or that are semantically related to the metadata terms via relationships such as synonymy. This assumption, in addition to having the advantage of preserving the simplicity in the design of the system, is also likely to be a necessity, since structures not having these recognisable elements are unlikely to contain expressions that are factually reliable enough to sensibly result in data updates. Given that the interpretation of many language excerpts can be ambiguous from the point of view of speakers of a language, the resolving of such structures is well beyond the scope or indeed interests of a system such as this.

An essential element in the successful deployment of the program is the design of the database Schema, which has to correspond conceptually to the human model of understanding for the domain. For this reason, the starting point for the database is an abstracted description of the domain, but one that is realisable through actual data structures. The type of data model chosen is entity based, since this bears relatively close resemblance to human perception. The application also implements the notion of database keys, using a twofold system that distinguishes between humanly intelligible keys and those that actually function as keys within the database. In other words, the human keys are the natural language excerpts that would identify an entity to a speaker of the language, for example: 'I'm reading *Pride and Prejudice*'.

As mentioned above, the Templates within the system are grouped into Template Types, which each define the nature of the update statement to result from a match with a Pattern based on the Template. The Template Types each have a set of parameters corresponding to the non-constant terms in the corresponding Templates. This includes those for data values, for which the text is simply copied straight out of the message structure whenever a match is encountered. The updates must also be informed by the context model, so that the correct entities be identified for update. For example:

```
She wrote Persuasion
```

relies on the ability of the application to recognise 'she' as referring to the entity that is identified within the database as Jane Austen (or whatever the key value is), or indeed as whatever female author has most recently been mentioned.

The pattern matching process is complex and relies not only on the accurate interpretation of patterns examined, but also on the matching having occurred in an appropriate order. The main reason that the order of comparison is important, is that it can help to avoid invalid matches. When the Templates are not ordered correctly, the process will often result in whole chunks of sentences being treated as though they were data values, and the repository updated accordingly. This is because the content of terms occurring at placeholder positions is not actually examined in itself, but rather the surrounding elements are checked for comparison with the matching pattern. If the surrounding elements match a pattern, the terms at placeholder positions are assumed to be the data values. This is unavoidable since the system cannot be expected to judge the legitimacy of the data. The correct ordering was also an issue for the present Project, as is discussed in chapter 4.

The most recent extension to the system involved the capturing of temporal information from messages containing data that is time variant. This notion is touched on within the Implementation chapter, in section 4.4 on the Tense and Inflection aspect of the present Project. More details on the present application can also be found in the Implementation chapter.

The existing set of Templates is listed in Appendix B.

2.1 WordNet

WordNet is a lexical database that attempts to store words according to a model of the way in which speakers of a language conceptualise them. The structure of the WordNet dictionary is dictated by the meanings of the entries within it. Words are held in logical groupings according to semantic criteria; a word is viewed as consisting of two elements, its form and its meaning. One of the basic units within WordNet, apart from the words themselves, is the *Synset*. A *Synset* is a set of synonyms, whose interchanging within a clause could feasibly be said to not affect the overall meaning, in a particular context. (True synonymy within a natural language is extremely rare, since the exact meaning of a word is both subtle and contentious, as well as varying according to context.) For example, the word 'car' is in the same *Synset* as the word 'automobile'. These sets of words are defined further in terms of the relations that they have with the other sets; this is the 'net' aspect of the utility. Miller (1995) outlines the main functions in WordNet.

WordNet's model of the relationships between the *Synsets* is implemented through the use of *Pointers*. These *Pointers* are classed according to type, e.g. hypernymy, holonymy, antonymy and so on. To reference the above example, the word 'car' has the word 'vehicle' as a hypernym (because 'car' is a kind of 'vehicle') and the word 'taxi' as a hyponym (because 'taxi' is a kind of 'car'). The *Pointer Types* are framed in terms of two broad categories, lexical and semantic. Lexical *Pointers* hold between words, whereas Semantic *Pointers* hold between *Synsets*. Additionally, there are other relations that hold between entries, which are not implemented through the *Pointers*, but rather comprise searches that are likely to be common. The WordNet browser, through which the facility is available as a desktop application, provides access to searches based on all the *Pointer Types* that are applicable to a given entry, functioning much like a thesaurus. However, the program is more commonly used as a component or resource within other applications, such as the present Project. There are numerous Application Programming Interfaces to WordNet, providing access to its Dictionary files and functions in a variety of programming languages (see below for more on the Java interfaces).

The WordNet Dictionary files contain an 'index file' and a 'data file' for each part of speech stored in the database (noun, verb, adverb and adjective, i.e. data.pos and pos.index, for example data.noun or verb.index). Additional files contain the algorithms used within the application to implement various searching functions. The *index* file for each part of speech contains a list of all the words held, with each word listed alongside the offset number indicating the positions of all *Synsets* in which it occurs, in the *data* file. Entries in the *index* file include the lemma (root form) for the word, the number of *Synsets* in which it occurs, as well as the number of *Synsets* pointed to by the *Synsets* in which it occurs, the different types of *Pointer* that the

Synsets have, and the number of senses of the word that are ranked according to frequency. The *data* file contains all *Synsets* for the part of speech together with the list of *Synsets* that each points to. Entries in the *data* file also include the lexicographer file number for the *Synset* (more on this below), the number of words in the *Synset*, the number of *Pointers* that the *Synset* has, as well as the *Pointers* themselves, and the gloss information for the *Synset* (text in which the meaning of the *Synset* is defined, and/ or example sentences are given).

Entries in the WordNet dictionary are further classed in terms of the Lexicographer File categories. The Lexicographer Files are used in the creation of the database and correspond to semantic groupings for each part of speech. The Files are prepared by lexicographers, and are not actually included in the WordNet release. However, the Lexicographer File classification for an entry is listed in the *data* file, as mentioned above, in the form of a number. There are currently 45 lexical categories in WordNet, each having a number, a name and a short textual description of what the category means, see Appendix H for the full list.

In addition to being available as an online facility, WordNet has been used within a wide variety of different applications. There are several Java interfaces to WordNet available, each offering slightly different access to its content and functionality. The Implementation chapter section 4.3 further explores those that were experimented with during the development of this Project. Developers working with WordNet are also well supported; there are a whole host of resources available due to the fact that the program is very widely used in a variety of contexts.

Aspects of WordNet that are related especially to the present Project are further explored in the Implementation chapter, section 4.3.

Five Papers on WordNet (URL listed in Bibliography section) explores in detail the structure of the Lexical Database.

3 Analysis and Design

3.1 Project Aims

As indicated in the preceding chapters, the main aim outlined for this Project was to extend an existing Information Extraction application to cope with a wider range of input. Specifically, the objective was to account for relevant verb structures, expressing the same kind of information as the structures already processed, but in a different form.

There was an inevitable initial requirement to examine the system as it stood at the commencement of the Project, and particularly to explore those aspects of its functionality that would have bearing on the work at hand. Given the nature and complexity of the application, it was difficult to arrive at a formalised design in advance of the implementation period. Although code delivering the present extension affects only some of the phases within the Extraction process implemented within the system, it was of course necessary throughout to ensure compatibility of any additions within the overall functioning of the application. For this reason, the consequences of potential changes had to be thought through as comprehensively as possible in advance, and then added incrementally, with frequent rounds of testing during the whole development process.

In arriving at the set of structures that the extension would focus on accommodating, the starting point was the set of structures already catered for. These were used to create a secondary set of structures capturing some of the same information as the existing set. The details of these structures are outlined in section 4.8 of the Implementation chapter.

The general aim then, was to incorporate a new set of structures of a defined type, with which the system could carry out the full range of functionality that was applicable to the existing structures. This overall aim implies a series of individual requirements that are further explored in the following sections.

Additionally, the Project had a secondary aim whose nature was inevitably less well defined. While a recent extension to the system (Cooper and Manson 2007) incorporated the recognition of temporal information, there is as yet no provision for verb tense. For certain domains, this is unproblematic, for example, with domains whose information is time invariant, such as that catered for by the main extension implemented in this Project. However, there are likely to be domains for which temporal information, not only that expressed explicitly (as is accounted for in the aforementioned recent extension), but that implied by the tense within a sentence.

To this end, the secondary aim for the Project was to investigate the handling of verb tense within the application, using WordNet and whichever Java interfaces to it could assist. As is discussed in more detail in chapter 4, while WordNet resolves inflected forms into the base or infinitive form of a verb, it does not readily provide the list of inflected forms that correspond to a particular base form. This part of the Project was therefore approached from the point of view of creating some sort of facility to retrieve inflectional forms on having queried with a base form. Given that this particular aim comprises exploration more than development of actual software, requirements for the format of the end result were undefined at the commencement of the Project. See section 4.4 in the Implementation chapter for details of what was achieved during this part of the Project and the nature of the end result.

3.2 Constraints

The obvious constraint upon the development was that it be compatible with the existing system and comply with the overall aims and purposes of this. Another was that the incorporation of the new structures, ideally, interfere as little as possible with the central structures and algorithms already operating within the system. Specifically, there was a desire that any new Template structures be related to existing Template Types, rather than requiring the creation of new ones. This constraint was in the interests of the system generally, not only to preserve the 'lightweight' aspect of the approach, but also because it would make for maximum efficiency, since that processing associated with the Template Types and their associated updates is complex and labour intensive. Intuitively, the building of new structures on top of those already present is legitimate in any case, because the structures aimed at do not frame information that is different in nature to that already processed, but rather represent an alternative formulation of the same.

The set of Templates that existed within the application before the beginning of this Project, as well as the new set are listed in Appendix B. Naturally, the main difference between the two set is the verb structures within them; the details are explored further in chapter 4.

Although not an explicit constraint, the use of the WordNet program was also strongly in the interests of the application as a whole. The main way in which the WordNet dictionary contributes to the system is in the generation of terms, whether synonyms in the existing application or verbs in the present extension. This means that words generated will likely be subject to processing by the WordNet interfaces at a later stage, for example in the matching phase, and so to lend coherence to the whole program, it was sensible to use the WordNet resource throughout.

3.3 Requirements

An outline of the requirements arrived at for the present extension follows.

- It should achieve the definition and processing of a new set of Templates featuring varied verb forms that are relevant to the domain at hand, i.e. new structures based on the Templates must successfully result in updates to the data.

- The new structures should be subject to the same level and extent of processing to which existing structures are.

- It should conform generally to the constraints that are implied by the present system, and to the processing stages within this.

- Its execution should not hamper in any way the functionality that the system enjoys at present.

- The full range of its functionality should be available/ accessible through the same interfaces that deliver the present facilities.

Each of these requirements will necessarily involve a series of smaller requirements in order to deliver it, for example, the requirement to generate verbs that are associated with the schema terms. These aspects are further explored within Chapter 4.

For the secondary aspect of the Project a single requirement is defined:

- A period of investigation and experimentation should inform the options that are available in terms of inflectional form generation.

3.4 Design

Given that the brief is to extend an existing application, there were naturally a number of design aspects that were determined in advance of the present Project. The system features a single package containing all classes delivering the current functionality. It was sensible to implement some of the new classes within the same package, for example those providing alternative versions of components already existing (and indeed, in one case a class that inherits from one of the original classes). However, two additional packages were created to specialise in certain areas. There was also the need to provide additional code within the existing classes for those parts of the procedure that required a tailor-made response for the new elements.

The decision was taken to implement the new Templates and associated Patterns by building on top of those elements already present, details of which are provided in chapter 4. One extra phase in the process would also be required to facilitate the use of the new structures. This would involve the generation of verbs associated with Schema terms, whose instantiation would then be used to deliver the incorporation of the new structures. The reason that this additional processing phase is required is that the present extraction bases its matching algorithm on the names of Schema elements (metadata terms), whereas the matching for Verb Patterns will require the recognition of terms that are not actually present within the Schema itself, but that are related to it.

In terms of the secondary aspect of the Project, there has been a set of components built in to the submitted application for demonstration purposes. However, these components are not actually required for the delivery of the main part of the extension, but rather are provided for illustration only. The details are outlined in chapter 4.

4 Implementation

4.1 The Application

In addition to the inevitable period of examination and familiarisation required by any extension to an existing application, this Project required a degree of research into other applications and facilities used in the development.

In order to fully explore the extension, a brief overview of the system's original functionality follows (building on the briefer introduction to the application in section 2.2 of the Survey Chapter). The application is substantial, featuring a number of complex algorithms, as well as assumptions. Naturally, the logic of these required careful consideration before the necessary additions could be designed and eventually implemented. A period of thought and experimentation with the system prior to embarking on the work of the extension itself was therefore carried out. The design of the extension is naturally dictated in good part by the shape of the existing system, and so it makes sense now to outline those parts of it that are most relevant.

For the purposes of illustration, examples of input to the system will conform to the 'library' domain used in the application development. This domain was already in existence when the present Project began, having been successfully integrated with the application's execution. Given that the extension seeks to achieve the translation of new structures into those already handled by the system, it seemed reasonable to develop the extension using this domain. The extension was then tested and evaluated using a secondary new domain, since this would make progress easier to gauge, as the development advanced (see chapters 6 and 7 for details). The Schema within the 'library' domain consists of two Entity Types: Book and Author. The Book Entities have the Attributes: ISBN, title, author, year. Author Entities have Attributes: ID, name, dob, gender. Those input files associated with the 'library' domain are listed in Appendix F.

As it stands, the application, written in Java, takes as input a set of text files representing the Database and Schema, the set of Templates together with their Types, Synonyms and Context information already generated and the Message itself. The Templates each represent a single structure, normally a sentence, to be potentially identified in the incoming Message. A Template consists of a series of placeholder and constant terms representing the words and phrases expected in a matching sentence from the Message. For example, the Template:

```
<spronoun1> is the <attribute1> of <<hkey1>>.
```

would constitute a match for the following sentence, contained in the incoming Message:

```
She is the author of Persuasion.
```

Placeholder items can also match multiple terms in the Message, for example if the above Message featured Pride and Prejudice rather than Persuasion. This matching process is accomplished using the generation of a set of Patterns for each Template; possible realisations of the structure described by the Template. The incoming Message structures are then checked against each of these Patterns in turn, to establish whether or not information in this form is present. For example, the above Template would give rise to the generation of the following Pattern, a match for the sentence given:

```
She is the author of <Book>.
```

where the book term is a placeholder, where the data value for the stated author attribute should occur in a successful match.

Each Template is associated with a Template Type, which specifies how a match for the Template should be interpreted, i.e. what form the resulting update to the Database should take. For the above example this would involve updating the Entity with Persuasion as its identifying key value, to reflect the name of the presently talked about person as its author Attribute (hopefully Jane Austen). The name of the author would be held within the Context information having been previously explicitly mentioned earlier in the Message, hence the use of the personal pronoun in this instance. The Template Types specify not only the structure of any resulting updates, but the type and number of parameters contained in Templates of that Type (parameters being the terms other than constants within the Template). Those terms occurring at the placeholder positions are therefore filled into the specified update statement to complete it and hopefully add information to the Database. In this sense the placeholder terms indicate the data in the sentence, as it is framed by the application.

4.2 The Extension

The initial brief for the present Project was to extend the system to account for an increased range of Message structures, but ideally without extending the Template Type system. This meant that the new structures were to be 'translated' into updates already identified, sparing the extension the difficulty of implementing the update process for a whole new set of Templates (one of the most complex algorithms on which the application functions). This was hoped to also preserve the 'lightweight' aspect of the approach so far taken. The most straightforward way to accomplish the desired effect therefore seemed to create a new set of Templates for structures with the varying verb formats desired, and to associate these with the existing Templates, treating any match with the new type as though it were a match for the original. It was hoped that this would allow the extension to avoid interference with the database aspects of the application, disturbing the existing functionality as little as possible.

Verb Templates, as with ordinary Templates, are linked to a Template Type according to the parameters that they contain. The main superficial differences with Verb Templates and their relation to a Type concern both the verb placeholders themselves and the pronoun terms. Verb Templates feature verb terms where the original type would have featured attribute terms, for reasons outlined below. Also, the type of pronouns in the Verb Template and original Template need not be the same, i.e. a Verb Template having a pronoun term can be linked to a Template Type having any pronoun type (subject, object or possessive). For example, a Verb Template having placeholders: <verb>, <value> and <spronoun>, might be assigned a Template Type having the parameters: attribute, value and ppronoun. (-'spronoun' stands for 'subject pronoun', 'ppronoun' for 'possessive pronoun'.)

Another difference for the new Verb Templates is the way in which they are used to generate Patterns. This inevitably affects the manner in which a match for the Pattern is turned into an Update, explained in section 4.5.4. With the original functionality, the matching process automatically creates Match objects whose content informs the update phase. However, with the new Templates this is not sufficient, and is therefore complemented with additional code to 'translate' a match for a Verb Pattern into a match for an original Pattern (that the update code is already able to deal with). This removes the need for any extension to the update component itself.

To this end, the program has been extended as follows: as prior to the extension, the application initially reads in the text files (Templates etc) and generates a set of Patterns on user request (and according to user input regarding synonyms). These Patterns are then used to generate further Patterns according to the Verb Templates, which have the same general format as the original Templates (except that they

feature additional placeholders for `<verb>` and `<passiveVerb>`). For an overview of the set of Verb Templates that have been successfully accounted for within the extension, see section 4.8.

At the point in the original program run where the user has elected to choose synonyms for Template terms, to be used in the matching process, it is now possible to generate and edit a set of verbs to be associated with any of the Attributes in the Schema. This is done through a dedicated GUI (the Verb Frame) where verbs related to the Schema terms are produced by querying WordNet through two Java interfaces. Verbs associated with Attribute names (assumed to be nouns) are generated and filtered according to user selection of lexicographic categories. A set of verbs arrived at through this process is then attached to the object representing the Schema element whose name on which the set has been generated. The reasoning here is that the translation process associates verbs with Attributes (more on this relation follows). See section 4.3 for more details of the use of WordNet within this phase.

The end result of this process is that some of the Attributes listed in the Schema will have a set of verbs that have been identified as relevant to them. This indicates that Verb Templates featuring these may contain information that is of interest to the system. For example, in the 'library' domain, the Author attribute might have a set of verbs (e.g. featuring 'write', 'script' etc) associated with it such that, when any of these verbs are encountered in an appropriate structure within a Message, they will indicate some information about the Attribute.

The first set of Patterns, having already been generated by virtue of the existing functionality, are queried to identify those required for the secondary phase of generation on Verb Templates. Those Patterns referring to Attributes, that have had a set of verbs identified during the verb generation process, are collected (i.e. any Patterns having a term that represents an Attribute that has verbs associated with it). These Patterns are then used in conjunction with the Verb Templates to arrive at a second set of Patterns representing the various verb structures modelled.

Verb Patterns are each associated with an original Pattern of the same Template Type, so that in Database terms, the two Patterns model the same kind of information. The assumption here is that this new set of structures being modelled are not to accomplish the inclusion of new types of information, but rather are an alternative realisation of the information already theoretically captured. That is, the information contained in one of the new Patterns will be, for the purposes of the Database (and therefore the application as a whole), the same information that is captured by an already identified Pattern. To illustrate, given the 'library' domain, as far as the Database is concerned:

```
Jane Austen is the author of Emma.
```

contains the same information as:

```
Jane Austen wrote Emma.
```

and therefore can sensibly be treated in exactly the same way when encountered in a Message. The form of the transformation between the two types is also clearly demonstrable in this example: in the second sentence, the verb term provides the same information as the predicate phrase indicating the Attribute in the first ('is the author of' becomes 'wrote'). The idea that this transformation is legitimate has formed a central principle within the Project and is crucial to the justification of the approach taken. (Please see the Chapter 5 for a detailed discussion of some of the linguistic aspects of the extension.)

In practical terms, the reason for using the existing Patterns is twofold. Firstly, any match for the verb Pattern will be treated as a match for the original, achieving the translation of new structures into existing Update types. Second, the process of generating Pattern terms (IEPatternItems in the Java code) is extremely complex, especially with regard, for example, to gendered pronouns. For this reason, it seemed sensible to use the terms already generated as far as possible, instead of executing the same arduous process a second time for Verb Templates. Appendix C lists Patterns generated as part of this Project.

When constructing a Verb Pattern, everything other than constant terms, pronouns and the verb items themselves are copied directly from the original Pattern to which it is related. Constant terms are unproblematic in that the Pattern Item is simply created using the text from the Template itself, whereas verb items constitute placeholders and therefore have match code tailored to them (although as Pattern Items they are relatively unproblematic to generate). The most complex part of the Pattern generation process for the Verb Templates is in fact pronoun terms. It is essential that pronoun generation be achieved correctly, since the successful update phase depends on it. The application maintains a Context object throughout the extraction process in order to resolve linguistic phenomena such as reference. A common example is the use of a pronoun to refer to a noun that has previously been explicitly mentioned, for example 'she' when 'Jane Austen' has already appeared. The details of the Context management part of the system are complex and, for the most part, of little relevance to the present Project. Suffice it to say that the generation of correct pronoun terms is essential to this part of the program logic and therefore to the successful execution of the process as a whole.

To illustrate, if the original set of Patterns for the above example Template included the following Pattern realisations:

```
She is the author of <Book>.

He is the author of <Book>.
```

the Pattern generation algorithm has arrived at these having examined the original Template and established its relation to the Schema. (It has not, for example, generated the following:

```
It is the author of <Book>.
```

because the author Attribute of the Book Entity Type is specified as gendered according to the Schema.) A Verb Template associated with the example might be:

```
<spronoun1> <<verb1>> <<hkey1>>.
```

with the corresponding example Pattern structure:

```
She wrote <Book>.
```

In order that the extension need not carry out the same procedure as the original code to arrive at the correct pronoun variations, the original Patterns were initially used to determine which gendered pronouns are appropriate. This involved copying the gender of the pronouns in the original Pattern with which a Verb Template has been linked. However, this approach became problematic when dealing with passive structures, in which not only are the subject and object reversed (in relation to their active counterparts), but the gender appropriate to the new structure may not even be present in the original. This is due to the fact that the original structures all feature active predicates in the form of the verb 'to be' (see Chapter 5 for more discussion on this point).

After substantial experimentation, using a combination of the original terms and the Schema information to generate the appropriate gender for pronouns in the new structures, a uniform approach was eventually taken. For subject pronouns in active sentences, e.g. 'she' in:

```
She wrote the book.
```

the gender is assumed to correspond to the gender of the Attribute referenced in the sentence. In the above example the subject is referring to the author Attribute, which is gendered, and it is therefore sensible to generate Verb Patterns featuring both the male and female varieties of pronoun, but not neuter. For object pronouns in active sentences, e.g:

```
Jane Austen wrote it.
```

where 'it' would have been represented within the Template as an object pronoun, the algorithm infers that the object pronoun refers to the Entity related to the Pattern, i.e.

the Book, which is not gendered (so there is no need to generate Patterns having 'him' or 'her' instead of 'it'). For subject pronouns in passive sentences, e.g. 'it' in:

```
It was written by Jane Austen.
```

the pronoun is assumed to refer to the related Entity (the Book) which in this case is neuter. For object pronouns in passive sentences, e.g. 'her' in:

```
Persuasion was written by her.
```

the pronoun is assumed to be referring to the Attribute related to the Pattern (Author, which is gendered). In general this production of the pronouns offers two options, either the pronouns are generated according to the gender of an Entity associated with the original Pattern, or according to that of an Attribute. Subject pronouns in active and object pronouns in passive structures use the Entity; object pronouns in active and subject pronouns in passive sentences use the Attribute.

While this approach has the natural advantage of being comprehensive, it did originally make for a fundamental alteration to the generation algorithm. A single original ('non-verb') Pattern might now merit the generation of more than one Verb Pattern, where previously this was done on a 'one-to-one' basis. The assumption had been that all appropriate gender variations would be present in the original Pattern set. That is, it was assumed that Patterns featuring both 'he' and 'she' would be encountered during the Pattern generation process for Verb Patterns and would consequently produce a variation for each in the new set. The obvious drawback to this feature is the needless repetition of generation resulting in identical (and therefore redundant) Patterns, but this is almost certainly outweighed by the associated increase in accuracy and consistency. Clearly the assumptions involved in this process will exert further constraints on the construction of the Schemas and their relation to the given domain used as input to the application. However, these constraints are likely necessary for the inclusion of the desired structures to be feasibly achieved.

The use of the initial set of Patterns in the generation of Verb Patterns has allowed these Patterns to behave as a variation on the original. When a Verb Pattern is matched to the incoming Message, the extended code replicates the behaviour of a match being encountered for the original Pattern to which this one is related. In order to do this, some of the terms from the original are necessarily copied across to the Match objects, whose instantiation results in the eventual update to the Database. Additionally, some of the details in the Match objects are deduced through examination of both the present and original Patterns (see section 4.5 on Changes to the Code).

While the relation of Verb Patterns to those already created by the program achieves the recognition of the desired incoming structures, it does impose one or two

constraints on the new functionality (not, however, limitations that are thought to be detrimental to the application's utility). For efficiency, the Verb Patterns are only generated if some of the existing Patterns do indeed feature terms that have had verbs identified for them. If no verbs have been identified, no Verb Patterns will be generated at all. This seems reasonable because the matching process will produce no results whatever for the Verb Patterns if there are no verbs generated, so there is little point generating these redundant Patterns in the first place. Also, the process for generating the new Patterns relies generally on the original set of Patterns having already been generated, and having been generated accurately, since the generation of the terms in the new Patterns (as well as their successful matching and resulting update), is entirely dependent on the content of the originals.

The relating of the new Verb Templates to Template Types (and, by implication, to the original Templates of the same type) is crucial for the extension to function accurately. A sentence corresponding to one of these new structures must effectively contain exactly the same information (in Database terms, not in terms of human discourse –see Chapter 5 for more) as one corresponding to an ordinary (non-verb) Template associated with the same Type. In addition, the information required by the existing code in order to generate Updates correctly must be present in any Pattern generated so that it can be used to aid the process of generating the new type of Pattern. It is thought that in general, the legitimacy of this relation will be further informed as new domains are encountered, as discussed in chapters 7 and 8.

The process of relating Verb Templates to existing Templates and Template Types is one that also needs to be understood for the system to function effectively. On input, the Verb Templates, like the ordinary Templates, will carry an indicator of which Template Type they are to be linked with. As mentioned above, this will dictate the update structure adopted on successful extraction, and so is essential to the functioning of the extension. The Template Type for a Verb Template must be dictated not only by the type and number of parameters specified in the Template Type definition, but also by the nature of the relationship between parameter and update (especially with regard to Attributes). A Template featuring a verb or passive verb placeholder must be related to a Template Type having an Attribute parameter that corresponds to the verb/ passive verb in the Verb Template, as opposed to the attribute element in the original Templates. This means that a match for the Verb Template will result in updating the specified Attribute, as required. The linkage therefore needs to be implicit in the creation of the Verb Templates text file, read in at the start of the program run.

For the successful use of the extension, there is naturally a required level of user input to the system, since the new code will in fact only be executed when the user

has opted to generate verbs. Additionally, there will be a minimum level of understanding of the relation of Verb Templates to Types that is required for the extension to function accurately. In this sense, a basic constraint on the extended application will relate to the understanding of the user. However, this can presumably be said for the application in general, and it is hopefully not the case that additional unreasonable expectations are implied by the extension. These issues are discussed in more detail in the Chapter 7.

4.3 WordNet and Java

The implementation of the Project required an initial period of research and experimentation in order to establish which tools were to be used in the development. The application utilises the lexical Database WordNet to deliver its current functionality, accessed through the use of a Java interface to the WordNet 'Dictionary files'. There are several Java APIs to WordNet available, each providing slightly different access to its data and utility. Having established that the required functionality was indeed available within the WordNet program through the application's main user interface, it was critical in the initial stage to figure out which of the Java interfaces, if any, would provide similar functionality for use within the program. It was necessary therefore to experiment with a number of the available libraries, and additionally to carry out a small amount of research into the construction of the WordNet application itself, before the development of the Project could get fully underway.

The existing code uses the JWNL API developed by John Didion (URL listed in Bibliography – Web Resources), an interface to the WordNet dictionary files offering access to many of the search and query functions provided by the main WordNet browser-accessible program. It became clear at an early stage in the exploration of the interface that the JWNL methods did not fulfil entirely the functionality required for the advancement of the Project. As outlined already, the extension is intended primarily to process sentences of the form:

```
Jane Austen wrote Emma
```

where currently it accounts for sentences such as:

```
Jane Austen is the author of Emma.
```

As far as the application is concerned, these sentences are considered to be equivalent; that is, in terms of the data modelled, these two sentences contain exactly the same information, and should therefore result in the same modification to the existing data. The approach adopted was therefore to perform some kind of transformation from the first sentence into the second sentence, in order that such sentences be processed without requiring a great degree (if any) of change to the data/ Template Type structures as they currently stand. As is discussed at length in the following sections, this 'translation' comprised the main area of work for the Project.

To explore the relation between the two sentences above, it is necessary to find the association between the words 'wrote' in the first sentence and 'author' in the second, this connection encapsulating the semantic link between the information contained in the sentences. This type of association is visible within the WordNet browser when selecting the 'derivationally related forms' option on having initially queried either of

these types of words. (E.g. when the noun 'author' is queried for derivationally related forms it returns, among others, the verb 'to write'.) Where most relations in WordNet hold between members of the same part of speech, this relation exists between verbs and nouns (in both directions). As is explained in Chapter 2, the WordNet program links words and Synsets (those sets of words that are deemed synonymous in some context) to their related forms through 'pointers'. These Pointers each conform to one of the set of 'pointer types' e.g. hypernym, holonym etc. The Pointer Types are classed as either Semantic (one whose target is a Synset) or Lexical (one whose target is a word). However, the WordNet program also provides other kinds of relation between words that are not framed in terms of the Pointers, for example in the aforementioned relationship between words having some derivational link.

On studying the WordNet source code files, written in the C language (particularly wnglobal.c and search.c), it became apparent that the relations of interest to the extension are available as the result of searches, but do not have Pointers associated with them. In order to establish how these relations are implemented, the WordNet dictionary files were then examined. As is outlined in Chapter 2, each part of speech has an 'index' file in the WordNet dictionary (with the .index suffix), containing every word held for that part of speech, together with the associated offset numbers (indicating the location in the dictionary files) of all those to which it has a relation. At this point it was clear that the requisite information was indeed available within the dictionary files, but the problem with the JWNL interface remained. The JWNL classes provide access to Synsets and their relatives through various methods, but it is generally necessary to indicate the type of Pointer, through which related Synsets are to be searched, in order to carry out the method. For the required relationship, there is no Pointer type and therefore the methods could not be executed appropriately. The JWNL interface does implement a Pointer Type called 'nominalization', which returns noun forms associated with a verb, however it only works in one direction and the main relation required in the extension goes in the other (noun to verb, e.g. 'song' to 'sing'). The results of this 'nominalization' pointer can also be somewhat unpredictable; see Appendix G for a test program using this function.

After a period of experimentation with the JWNL classes and examination of the source code to determine how the methods were actually implemented, a series of test programs finally established the one method that returns all Synsets that are related to a given one, without reference to any particular Pointer Types. While this meant that verb Synsets could be associated with the relevant nouns (and vice-versa), it also meant that some form of filtering would be expedient. The required Synsets were therefore acquired by retrieving only those having the desired part of speech,

considerably reducing the number of results. Having reached this stage however, it was apparent that further refinement of the retrieved words would also be useful.

Through using the WordNet browser, it had been observed that Synsets were categorised according to a further type of classification, the Lexicographer File category. On querying a word in the browser it is possible to be presented with information of the following form ('write' has been searched for):

1. (112) {01683134} **<verb.creation>** write#1, compose1#3, pen#1, indite#1

The section of the result in bold indicates this classification, and was felt to be of potential benefit in the application. Further research was therefore carried out to ascertain the implementation and indeed availability of this information for use within the program.

The WordNet dictionary is constructed using the 'grind' program, which builds the dictionary using a set of files prepared by lexicographers. A system of 45 lexical categories accounts for the Synsets held in the Database, grouped in terms both of syntax and semantics. Each category has an associated number to which it is mapped in the code (the Lexicographer File categories together with their descriptions are listed in Appendix H). The Lexicographer Files, which are prepared by a team of Lexicographers, are not released as part of the WordNet distribution, however, the lexical classification of each entry is available within the Dictionary files, and the categories are explained in the WordNet documentation. The 'data' files for each part of speech (files with .data suffix) contain the numerical representation of the Lexicographer category for which a Synset is listed. On examining the files it was deduced that the Synset numbers were allocated in order of Lexicographer File category. However, this information is not readily available through the various Java interfaces to WordNet. On discovering this, it was necessary once more to research and experiment with the different available APIs in the hope that one would provide access to the Lexicographer File information for a Synset. Unfortunately, most of the Java interfaces to WordNet do not use the Lexicographer File information; among those experimented with were: MIT's JWI, WNJN (URLs listed in Bibliography – Web Resources).

The sole interface that was discovered to have implemented the lexicographic information was the Java WordNet Library (wn.jar) developed by Dan Bikel at Stanford (URL listed in Bibliography – Web Resources), a lesser-known utility than the JWNL release. This interface provides limited access to the information in the form of 'file numbers', which can be mapped to the corresponding categories. This means that the syntactic classifications can be represented as returned to the WordNet browser, so that query results can be filtered more appropriately. The interface successfully provided the desired information, but was in general trickier to work

with, for example than JWNL (partly because it lacks the online resources that are dedicated to facilities such as JWNL that are widely used by developers). For this reason, the wn.jar utility was used solely to provide the necessary lexicographic information, and the JWNL interface for all other WordNet related processing required by the extension. The lexicographic information was intended from the outset to complement user input in the extraction process, as it was assumed that relevant categories of word would be most sensibly identified by the user of the application, according to the domain at hand.

It was also necessary, during the initial implementation stage, to experiment with different releases of the WordNet package itself, in order to reconcile the required functionality with both the Java interfaces and the operating system on which the system was being developed. Additionally, a slight change had to be made to the jwnl.jar source code in order to deliver this compatibility, and similarly a small change needed to be made to the wn.jar source code in order that it could deliver the full functionality.

4.4 Tense and Inflection

As outlined in Chapter 3, a secondary brief for the Project was to explore the possibility of retrieving the set of inflected forms of a verb on having started with the base form. While this requirement was not integral to the achievement of the primary aim of the Project, i.e. to successfully process additional verb structures, it was indicated that it would be in the interests of future development. For this reason, further work was carried out with the general objective of producing a set of valid inflections for a given word.

The WordNet application performs the evaluation of inflected forms into their base (source code morph.c), but not vice-versa. In other words, on processing an inflected form, the application resolves its base form and performs whichever functionality is required on this. However, although it handles these inflections, it does not store them explicitly within the Dictionary, with the exception of irregular forms. The program uses the Rules of Detachment (explained in the WordNet documentation) to remove inflectional suffixes and resolve the base form for regular verbs, nouns and adjectives. Additionally, the inflections for irregular words are listed in the Exception Lists within the Dictionary (with .exc suffix). As suggested above, these inflected forms are not available as the result of a query.

The JWNL interface is equipped with the Morphological Processor object, which provides, in Java, the ability to return the base form of a word on searching with an inflected form. This replicates the behaviour of the WordNet browser (for example, the form ‘wrote’ returns the form ‘write’). This facility has been used within the extension, to resolve incoming inflections into base forms as part of the extraction process. While this limited handling of tense is sufficient and indeed desirable to preserve the simplicity of the present application, it will be necessary, for future work, to provide the converse, i.e. the list of inflections for a base form.

Once it was established that neither the WordNet resource itself, nor the various interfaces to it, would provide the required facility, the decision was taken to attempt to build a repository of base and inflected forms, in order to explore the feasibility and indeed usefulness of such a component. For this reason the package wninflector was developed -see section 4.5.2 for details of the package. The package consists of a repository whose construction involves the reading in of both the WordNet index files and exception lists, and then the attachment of suffixes to forms found to produce inflections. The Rules of Detachment are here performed in reverse (not the activity for which they have been devised), attaching suffixes to base forms. While this results in the generation of valid inflectional forms, it has the unfortunate side-effect of also producing spurious forms. Checking these forms for validity with WordNet itself is not productive, since the program resolves these forms as though they were

legitimate. The solution to this problem, then, was to filter these results, either manually or (preferably) automatically. Given the fact that the English language does not conform to entirely predictable rules when it comes to inflection, this filtering seemed a necessary part of the process.

There followed exploration of the various English dictionaries available for download, including XDXF (Extensible Dictionary Exchange Format) files (whose suitability for Java would have been ideal within the application given the existing APIs for processing XML data). However, it was found that, in general, these dictionaries do not explicitly list inflectional forms, but rather, as with WordNet, resolve them using tailored algorithms. The sole facility that was found to be of use was the Automatically Generated Inflection Database developed by Kevin Atkinson (URL listed in Bibliography –Web Resources).

The AGID facility is a lengthy word list, featuring words for all parts of speech listed together with their inflected forms. The list is far longer than is required by the application, and features far more information than is necessary for its purposes. For this reason, the process of reading the file in for use within the program is unjustifiably labour-intensive, as discussed in section 4.5. In order to accomplish the this part of the Project, the code reads in only the verbs listed, since this is mainly what is likely to be required. However, it still results in a storage structure far larger than is desired. Experiments using the WordNet frequency information (the classification system in the WordNet dictionary relating to the frequency of use for terms within the corpora that are used to build the WordNet database) proved unsuccessful; storing only the terms that are classed as more frequent resulted in problematic omissions. The end result of this exploration is a repository that is far larger than necessary, however the basic principle that it is possible to retrieve inflected forms has, to a certain degree of success, been proven. The code provides the option to filter the forms manually if that suits the needs of the user (or if resources are limited, in which case the use of the list will almost certainly be prohibited) and to forego use of the AGID list. More details are provided in section 4.5.2.

4.5 Changes to the Code

An overview of the new Java classes and packages, as well as changes made to the existing application components as part of the present extension, now follows. Two new packages have been included, package verbutil, containing five classes, and package wninflector, with four classes; additionally six classes have been added to the default package within the application, as well as five test classes (many of the new and existing classes also have test code clearly marked within them). The two new packages will be looked at first, then new classes in the default package and finally changes to the existing classes. The code changes are explained in detail below, because the logic of the extension is in some places complex, and accords to reasoning that is not always immediately apparent on examination of the code itself. Lengthy comments have also been included within the Java code, please see the code listings included on the submitted CD for more information.

In the following section, for clarity, terms that refer specifically to code elements (as opposed to concepts that are considered to exist out-with any application) will appear in a different font to the surrounding text, for example, Synset refers to the JWNL 'Synset' class, whereas Synset refers to the concept of a set of senses as framed within the WordNet application.

4.5.1 Package verbutil

The verbutil package comprises 5 new classes, with the overall aim of providing linguistic information, available within WordNet, that is specific to the needs of the extension. Classes in the package are used to aid the verb generation process and the to resolve inflected verbs.

IEDictAssistant

This class provides utility functions required by IEVerbProcessor class, predominantly involving access to the WordNet library files using the JWNL interface outlined in section 4.3.

IEDictAssistant class contains five methods, each tailored specifically to the needs of the IEVerbProcessor class, for which it is intended as a helper. Method getNounSyns(String) returns a Synset array of the senses listed in WordNet for the specified word, whose lemma matches the String value passed as parameter (if any). Similarly, method getVerbSyns(String) returns a Synset array of senses associated with the verb indicated by the passed parameter String. Method getVerbWords(int) returns an array of Words corresponding to the Synset at the dictionary entry located at the position indicated by the number passed as parameter (the WordNet dictionary files are organised into Synsets, each having a corresponding offset number which

indicates its position in the dictionary file in which it is located; each Synset can have several Words associated with it). Method getPTargets(Synset) returns an array of PointerTargets (either Words or Synsets in WordNet that comprise the targets of a set of Pointers, the links that make up the 'Net' in WordNet as mentioned above) indicated by the passed Synset parameter value. It is possible using JWNL methods to obtain only the targets of a specified PointerType (e.g. hypernym, holonym etc) but, as discussed above, this was found to be insufficient for the purposes of the present extension. Using the generic method like this requires subsequent filtering of the results, since this method returns all Synsets and Words that are targets for pointers held by the Synset on which the method is called. Method getVerbGlosses(String[ ], int) returns a String array containing formatted Strings each representing the WordNet gloss information for its counterpart in the passed array. The String array passed as parameter will contain verbs listed as Words in WordNet and the integer value indicates how many are present in the array (which may contain null values); the information is formatted for the purposes of displaying it in the IEVerbFrame GUI.

IELexicographer

The IELexicographer handles information relating to WordNet's Lexicographer Files, which classify the entries in the dictionary files according to semantic criteria (as discussed in sections 2.3 and 4.3). IELexicographer's constructor reads in the 45 lexical categories specified in the WordNet documentation, together with a textual description of each. Files listing the categories and their descriptions are held at the same location as the other text files read in at the start of the program run (the IEFiles directory). The categories are composed of the name of a part of speech and a one-word indicator of the semantic classification (e.g. <verb-creation>, see Appendix H for the full list). Get methods are provided for arrays containing the file categories and their descriptions, and the total number of categories that are recognised. The method getPOSFiles(String) returns a String array containing only the lexical file categories that are applicable to the part of speech specified by the passed parameter value. Similarly, getPOSDescs(String) returns a String array containing only the lexical category descriptions applicable to the passed part of speech. The method createItem(String, String, WordNet) returns a reference to an IELexicalITem object according to the part of speech and word specified by the passed String parameters, and using an instance of the danbikel.wordnet.WordNet class (secondary Java interface to WordNet described section 4.3), as well as the IELexicographer object itself.

IELexicalItem

Class IELexicalItem models the lexicographic information for a single item (i.e. a word comprising an entry in the WordNet Dictionary files). A single IELexicalItem object will be associated with one or more lexical categories as determined by the WordNet object provided by the wn.jar interface. Get methods return the String representations of the word and part of speech for the item, the number of lexical categories listed for it, a boolean array representing its status for each category, and a String array containing the names of the lexical categories it is associated with. Additionally, the method hasfile(String) returns a boolean indicator of whether the category indicated by the passed String parameter value is in fact associated with this item. While the class IELexicographer provides general lexicographic utilities, this class describes one lexical item.

IETenseProcessor

Class IETenseProcessor resolves inflected forms into their 'base form', the standard infinitive form listed in the WordNet index file for verbs. The object uses an instance of JWNL's MorphologicalProcessor class, which uses the Rules of Detachment specified in the WordNet documentation, to return the base form of any inflected form handled by WordNet. Given that the class is provided as a utility for the present extension, its sole method is getPresent(String), which simply returns the String representation of the base form for the passed verb String. The present implementation provides only for verbs, however the class could easily be extended to cope with other parts of speech, should this be desirable in future work, given that the facility is readily provided through the JWNL interface.

IEVerbProcessor

The IEVerbProcessor class comprises a helper to the IEVerbGenerator class in the main part of the application. The class performs generation of verbs on Schema terms using JWNL utilities as well as information from the WordNet documentation, for filtering and minimising irrelevant results. The IEVerbProcessor also deals with maintaining and querying the set of verbs currently subject to processing during the verb generation phase. Get methods return the verbs currently held in a String array and the number presently in existence. Method produceVerbs(String, String) populates the set of verbs on the passed String comprising the name of a selected Schema element, and the second parameter String value indicating the lexical category verbs generated are to match (without this filtering aspect, large numbers of spurious verbs tend to be generated, requiring increased levels of user input). Method removeVerb(int) simply removes the verb at the specified index, rearranging the set as appropriate, while the emptyVerbs() method sets the number of verbs to zero and the verb set to a new empty array. The class also provides the method getGlosses()

which returns a String array of formatted gloss information corresponding to the current set of verbs held for display in the IEVerbFrame GUI. The method findVerb(String, int, int) is intended mainly for use by the IEVerbGenerator class but can optionally be used elsewhere. The method attempts to find the passed verb in the current set, searching between the two indices indicated by passed integer parameters, and returning either the position of the found verb in the array, or the index at which it would belong were it to be inserted –given that a positive integer will be returned whether the verb is present or not, this method is best used only by the existing classes as it is likely to cause confusion elsewhere.

4.5.2 Package wninflector

The wninflector package has been provided primarily to demonstrate the fruits of research, carried out into the generation of inflected forms (particularly of verbs) using the WordNet dictionary files as a base. As indicated in Chapter 3, the requirement to generate the set of inflected forms of a verb on the base form was separate from the main Project, in that it was not a function that was required in order to achieve extraction on the outlined structures for the main part of the extension. However, the assumption underlying the extension is that the information captured and described by the schema is time invariant, for example, with reference to the Library Schema, the following sentence:

```
Jane Austen is the author of Emma.
```

contains exactly the same information as:

```
Jane Austen was the author of Emma.
```

because the information is either true or false regardless of the temporal context in which it is placed; there is and always has been only one author of the book. For this reason the present extension resolves inflectional variations into their base forms prior to processing, so that all inflected forms of a verb are treated as though they contain the same information (which is true when the information is indeed time-invariant). However, there will likely be situations in which a Schema will provide a more authentic representation for a domain having time-dependent information within it by recognising the informative value of tense variation. (For example, as illustrated by Cooper and Manson (2007), the information about the monarch of a country will vary according to the period referred to.) Although the assumption was accepted that the domain for the present extension described time-invariant information, the secondary requirement for the Project was to explore the possibility of generating inflected forms for the purposes of future extensions. This is so that, if necessary, Patterns can potentially be generated for numerous inflected forms and matched separately. The main purpose of the classes in package wninflector therefore is to demonstrate the fruits of this investigation, and it is assumed that most of the functionality within this

package is unlikely to be used as it is within future developments, although certainly it is hoped that some of the components may prove useful.

IEInflectionRespository

IEInflectionRepository class instantiates and provides access to the IEFormSet object, optionally using an IEDictionaryReader object if one is available. Initially the constructor attempts to read in a Serialized instance of the IEFormSet class (see below section on IEFormSet and IEDictionaryReader for details), and only creates an instance if there is none available. When creating the IEFormSet object, the algorithm similarly attempts to read in a Serialized instance of IEDictionaryReader, but if none is available the IEFormSet is instantiated without it. (Optional code is provided to instantiate the IEDictionaryReader also but this requires a substantial amount of available virtual memory, and so feasibility for this will depend on circumstance -it is generally desirable for the objects to be instantiated once and then re-used, as explained below.) The class provides a single public method getInflections(String), which returns a Linked List of String representations of the inflected forms for a word (presumably a verb) passed as parameter.

IEFormSet

Class IEFormSet provides a repository of inflected forms together with their base forms, held in a Hashed Map structure for efficiency in searching and retrieval, the most likely common operations for the repository. Each entry in the map comprises a key in the form of the base String representation of the word (which will be unique within the structure), and a Linked List of Strings representing the inflections in no particular order. The constructor creates a map for a single part of speech passed as parameter from the IEInflectionRepository. It then populates the structure using WordNet dictionary index files, the Rules of Detachment and the Exception Lists, referring to the IEDictionaryReader if one is present. Initially the algorithm reads through the Exception List for the relevant part of speech (e.g. verb.exc), a text file listing irregular inflections for WordNet entries together with their base form (e.g. inflection ‘wrote’, base form ‘write’).

Next the WordNet index files are read through (e.g. index.verb) for all base forms listed in the dictionary for the present part of speech. These base forms are combined with the listed suffixes using the IESuffixAttacher class, to generate the possible inflections for each entry. When the IEDictionaryReader class is not utilised, the end result of this process is the production of both relevant and spurious inflected forms (that are nonetheless resolved by WordNet, as explained above), requiring a certain degree of user input in removing the irrelevant forms. When the IEDictionaryReader object is called on, only relevant forms (i.e. those contained in the Dictionary Reader for the same base form) are added to the repository, maximising accuracy and

minimising the size of the resulting structure. The single public method in the class is getInflectList(String) which returns the Linked List of inflected String forms for the passed String parameter value when called from the IEInflectionRepository object. The object is Serializable because the instantiation process is demanding of resources, involving as it does a lengthy period of input/ output operations; for this reason is was felt to be desirable that the object potentially only have to be instantiated once, since its state should not change between program runs.

IEDictionaryReader

As mentioned above, the generation of inflected forms using only WordNet data results in spurious terms; for this reason the IEDictionaryReader is provided to filter the results of this process. The class reads through the list of inflected forms contained in the AGID text file (explained above), storing only verbs and inflections that are resolvable by the WordNet interfaces, since these are all that the program will be interested in. The text file contains many entries that are simply ignored by the program, indeed most of the file will be irrelevant for the purposes of the class, since it contains all parts of speech and not just verbs. This means that the process of reading the forms in is a demanding one, and for this reason the class is also Serializable so that the process theoretically need only be carried out once (or indeed not at all if resources are limited). The use of the IEDictionaryReader object is optional, and can certainly be reasonably replaced by an increased level of user interaction.

IESuffixAttacher

The IESuffixAttacher class performs the concatenation of base forms with their relevant suffixes, producing all possible inflections indicated by the WordNet Rules of Detachment (as well as some impossible ones). As already outlined, the Rules are carried out in reverse of their intended use, but for regular verbs at least the results appear promising, providing that some form of filtering is present elsewhere in the program, either through user input or the IEDictionaryReader class (or indeed by some method developed through future work). The suffixes are laid out in a text file read in by the IESuffixAttacher constructor code, different sets for the different parts of speech and for variations within these. The single public method getForms(String, String) takes the part of speech and base form of a word and performs the attachment of (mostly) relevant endings, returning these in a Linked List structure. The algorithm uses both the part of speech and ending of the base form to compute the generated forms.

4.5.3 New Classes in the Default Package

IEVerbFrame

The IEVerbFrame GUI is provided to allow for user prompted generation and editing of verbs on Schema terms. The GUI is presented on pressing the Generate Verbs button in the already existing Synonym Frame GUI. References to the current Schema and the JWNL Dictionary object in use are passed to the IEVerbFrame object. The visual design of the IEVerbFrame is intentionally similar to the two already existing GUIs (the main GUI and Synonym Frame), for increased consistency and to maximise on usability within the application (and because the generation of verbs should not sensibly be a time-consuming task). The user is presented with the current Schema terms and Lexical Categories, together with any verbs generated thus far. Options are present to generate verbs on the current term and category, to delete selected verbs and to bring up the IEInflectionGUI (described below); there is also a section providing feedback and information tailored to the current selection. The end result for deployment of the IEVerbFrame GUI is that one or more Attributes will have an attached set of verbs. Initially this part of the process was designed so that verbs could be allocated to Entities as well as Attributes, however, although this aspect has not been utilised to accomplish the present extension, the option to add verbs to an entity has been left within the code as an option, should it prove useful for future domains.

IEVerbGenerator

The IEVerbGenerator class is largely responsible for the production and editing of a set of verbs. The class is referenced in the IEVerbFrame GUI, which creates instances of it and uses these to instantiate the IEVerbSet for a Schema element. The IEVerbGenerator takes pointers to the Schema and JWNL Dictionary instances as parameters to the constructor and additionally creates a WordNet object and instances of IELexicographer and IEVerbProcessor. Get methods return the current set of verbs in a String array, the corresponding glosses, the current number of verbs held, the index at which a verb is held, the lexicographic categories and descriptions for verbs, and a reference to the Dictionary instance. Much of the required functionality is delegated to the IEVerbProcessor object outlined above, created through refactoring practice. The method generateVerbs(String) causes the object to create a set of verbs related to the String value passed as parameter, by calling on the IEVerbProcessor. The methods removeVerb(int) and emptyVerbs() also call on the corresponding IEVerbProcessor methods. Methods setCat(String) and useTerm(String) allow for verb generation on a specific term or lexical category. To facilitate the setting of verbs via the IEVerbFrame GUI, the class implements the Cloneable interface, so that each IEVerbSet holds a reference to a deep copy of the IEVerbGenerator instance

with which it is related, and the original IEVerbGenerator pointed to within the IEVerbFrame can continue to be used for further processing on other Schema elements.

IEVerbSet

The IEVerbSet class comprises the verbs that have been identified as relevant for a particular Schema element. The class has its verbs populated by the setVerbs(IEVerbGenerator) method, called from within the IEVerbFrame GUI. The method makes a deep copy of the Cloneable IEVerbGenerator object (since this will likely then be subject to change), which is used to set the String array of verbs, the number of verbs and the JWNL Dictionary instance. Get methods return the number of verbs currently held, the array of verbs and the IEVerbGenerator itself, as well as the set of inflections for a single verb, or for the whole set (not used for the present extension but provided to illustrate the secondary aim of the Project, as indicated above). The method isAuxiliary(String) returns a boolean indicator of whether the passed verb is an auxiliary form, by checking through a small String array held within the class, that lists auxiliary forms that are relevant, i.e. the forms: 'have', 'has', 'had', 'did' and 'does'. WordNet does not store auxiliary verb forms, and so the decision was taken to store this selection explicitly within the code in this manner. (See Chapter 5 for more on this.) The method setInflections(LinkedList[ ]) sets the inflected forms associated with the verb set. Methods hasVerb(String) and hasVerbAt(String) return boolean and int values respectively indicating whether or not a verb is present and if so at what index it resides. The method hasSynonym(String) returns a boolean indicator of whether the passed String has a synonym that is contained in the verb set.

IEVerbTemplate

The class IEVerbTemplate extends the existing IETemplate class; it was problematic to use the existing class for verb Templates because they need to be read separately and have additional fields, as well as some of their own distinct methods. However, to reuse the existing code where possible, the class is a Subclass of IETemplate and inherits several of its methods. The class is fundamentally distinct from the IETemplate class when it comes to the Pattern generation process, which accords to a completely different algorithm to that of its Superclass. The method loops through the set of already existing patterns that: 1 -have the same Template Type as this, and 2 -reference an Attribute having a verb set. Each Pattern is used in conjunction with the Template to generate each item in the new Pattern in turn. New Patterns are added to the set using the addPattern(IEPatternSet, IEPattern) method.

In order to generate each term, the code uses the terms in the original Pattern, the placeholder terms in the Verb Template and pronouns having their gender dictated by Schema elements, as discussed above.

IEVerbTemplateSet

The IEVerbTemplate class is similar to the IETemplateSet class but naturally differs when it comes to its handling of the IEVerbTemplates themselves. The constructor for the class is virtually identical to that of the IETemplateSet class, however it was necessary to provide a separate IEVerbTemplateSet class given the differing requirements for the IEVerbTemplates held. The class provides a method to display the Verb Templates in the main GUI according to much the same format as the IETemplateSet code. The generateVerbPatterns(IEPatternSet, IEPatternSet) method loops through the Templates and, for each one, collects from the set of original (non-verb) Patterns those corresponding to the same Type, to pass to the IEVerbTemplate Pattern generation method.

IEInflectionGUI

The IEInflectionGUI class serves solely to illustrate the functionality within the program for the generation of inflected forms, such as it is. The display is really just intended to serve as a demonstration of the functionality so far developed. Indeed, the only reason that the inflection generation components have been included with finished product is to show what has been achieved in this area, since its facility is not actually required by the current extension. However, the Verb Frame GUI provides the option to view the IEInflectionGUI by pressing the 'Get Inflections' button, which results in the display of the currently selected verb set together with their inflected forms, and the option to delete any. The lists visible are populated by creating and querying an instance of the IEInflectionRepository class. The end result of this process is to set the inflections associated with the IEVerbSet held by the relevant Attribute, again, not something that is integral in any way to the main part of the extended application. It is assumed that at least some of the developed functionality will be useful for future work, as discussed above.

IESuperTest and IEVerbTest

As indicated, the default package within the application contains five test classes. IEVerbTest is responsible for carrying out the tests within these, which are implemented within the classes: IEDictTest, IELexTest and IEGenTest, all of which are subclasses of the abstract Superclass IESuperTest. Please see Chapter 6 on Testing for more information on these.

4.5.4 Code changes within existing classes

IEAttribute

The class IEAttribute has been extended to hold the set of verbs that have been identified as related to the Attribute name. To this end, the class now holds a pointer to an IEVerbSet object handling its verb terms. A boolean field indicates whether or not verbs have been generated on the Attribute because, while a pointer to an IEVerbSet instance will always be present within the class, there may in fact be no verbs held within it, for example if the verbs have been manually removed, or if none have been generated at all. Three more boolean fields also exist within the class to assist the hasVerbFor method, discussed below. The IEAttribute constructor now instantiates the IEVerbSet object for the instance, and initially sets the boolean hasVerbs field to false. The method getVerbs() returns a pointer to the IEVerbSet for the Attribute. The method addVerbs(IEVerbGenerator) sets the verbs in the IEVerbSet object through its setVerbs(IEVerbGenerator) method; the boolean hasVerbs field is set to true when the IEVerbGenerator contains a set of one or more verbs. The public method hasVerbs() simply returns the value of the hasVerbs field. The hasVerbFor(String) method is called during the Pattern matching process, returning a true value when the passed String parameter represents a verb held in the IEVerbSet, an inflected verb whose base form is listed in the IEVerbSet, a verb form which is recognised as an auxiliary in the IEVerbSet, a verb for which either it or its base form has a synonym according to the IEVerbSet, or a word which is recognised as part of a passive or other structure (i.e. some form of the verb 'to be' or 'by' on either side of the main verb or the word 'also' for two of the new Templates), which must be listed in some form within the IEVerbSet in order for a valid match. The class fields foundAux, foundBe, foundAlso, foundMain and foundPassive assist the algorithm as it checks each term in the incoming message in turn. See the extension to IEPattern class for more on this.

IEEntityType

As mentioned above, the original intention within the extension was to allow a set of verbs to be associated with any Schema element, either Entity or Attribute. For this reason code was originally added to the IEEntityType class similarly to the IEAttribute (fields pointing to an IEVerbSet, whether the element has verbs, and methods to return these). Although this aspect of the code was not actually used to accomplish the present extension, which uses only Attributes, the fields representing verbs associated with the Entity Type name as well as the ability to set these via the IEVerbFrame GUI have been included in the final product, in case they should prove useful in future extensions or for different domains. In general, the suitability of the

extension, and indeed the application as a whole is somewhat dependent on the suitability of a user's chosen domain and the various structures associated with it.

IEGUI

The main GUI has had a number of code elements added in contribution to the present extension. In terms of the class components themselves, there is a new Menu Item for Loading Verbs, a new File object for the Verb Templates input file, and pointers to an IEVerbTemplateSet and an IEPatternSet for the verb Patterns. There is also a pointer to a PrintWriter for the output of system test results, although naturally this is not integral to the overall logic of the application. For this reason, the code within the IEGUI class that writes this test information out to file will merit no further mention in this section (see Chapter 6 for more details). The sole addition to the IEGUI constructor is the inclusion of the aforementioned Menu Item for loading verb Templates. actionPerformed code within the class has been extended to include handling of the new elements in a number of places. When the 'Load All' option has been chosen, the code has been extended to include the loading in of the Verb Template text file, and the subsequent instantiation and display of the IEVerbTemplateSet. A segment accompanying the selection of the new 'Load Verb Templates' option has also been included within the actionPerformed method. Code responding to the selection of the Pattern generation option has also been extended so that Patterns are generated for the Verb Templates after the Patterns corresponding to ordinary Templates have been generated. The helper method getVerbPatterns firstly collects those already existing Patterns having some item representing an Attribute with verbs identified, and passes these to the IEVerbTemplateSet generateVerbPatterns method. On return, the new Verb Patterns are added to the already existing IEPatternSet and displayed within the GUI.

IEMain

The application's main method remains unaltered with the exception of test code used to carry out the bulk of the unit testing (system testing is largely carried out within the normal program run). See Chapter 6 for more.

IEMatchSet

The class IEMatchSet features a trivial alteration to accomplish the successful implementation of the Pattern matching process. When a match on a Verb Pattern Item is encountered, an IEMatch object is created, as with other Pattern Items. However, unlike ordinary Patterns, the Match objects for Verb Patterns are not actually used in the Update process, but rather are replaced by new Match objects related to the non-verb Pattern with which the current one is linked (and tailored to achieving an equivalent result for the present Pattern). Nevertheless, in order to

tamper as little as possible with the complex algorithm employed within the Pattern matching code, the Verb Pattern Items instantiate a Match object also, although one which is never again referred to. For this reason, the IEMatchSet features a small addition in its set method to account for the inclusion of these Match objects; the code replicates the behaviour of a Match for an Attribute item since this is its nearest equivalent in terms of the data.

IEPattern

The class IEPattern features substantial extension, since the 'translation' of matches to Verb Patterns into matches for existing Patterns comprises the central principle behind the logic of the extension. Among other elements, the IEPattern class holds an array of Pattern Items (representing the elements in the Pattern in the order in which they are expected to occur) and a pointer to the Template on which a Pattern object has been generated. The extended version therefore holds a pointer to the Verb Template from which one of the new Patterns has been produced. Additionally, IEPattern objects representing Verb Patterns also hold a field for the original Pattern to which they are related. A second constructor for the class is provided for Verb Patterns, taking the number of items and the Template on which it is based (as with the original IEPattern constructor), and also the Pattern with which it is linked. The field in the class which had previously held the Template on which the object is based, for Verb Patterns holds the Template from which the already existing related Pattern has been generated. The remainder of the extended code in this class falls within the match method, and conforms to a detailed algorithm, as outlined below.

The match method within the IEPattern class attempts to compare and match the Pattern with an incoming sentence from the Message. The algorithm works by comparing each of the Pattern Items against the terms in the Message in turn, taking into account the fact that a single term in the Pattern might match multiple terms in the Message excerpt; the full process is complex, however the following account will focus mainly on those aspects that are relevant to the extension. For terms other than placeholders in the Pattern, the method calls on the match method for the particular Pattern Item concerned, retrieving a boolean value as the result. For each term that is matched in this way, an IEMatch object is created, and the algorithm moves onto the next Item in the Pattern. For Items in the Verb Patterns other than verbs and passive verbs, the behaviour at this point remains the same, although the method behaves in a different way at the next stage. However, the match code for verb and passive verb Items is required to take a slightly different approach because, like placeholder terms in the original code, these may occupy multiple terms in the incoming message, and there are specific requirements on what forms these terms can take in order to constitute a match for the Item.

In addition to standard verb structures e.g:

```
She wrote Persuasion.
```

the verb Item is required to recognise structures featuring auxiliary verbs e.g:

```
She did write Persuasion.
```

or:

```
She had written Persuasion.
```

where the auxiliary verbs serve no informative purpose other than to reinforce or modify the main verb, which is where the information required by the system lies. Similarly, structures featuring passive verb forms e.g:

```
Persuasion was written by her.
```

where 'was written by' constitutes the entire passive verb element in the sentence (as far as the system is concerned –see chapter 5 for more) will naturally require recognition, given that they are likely to occur regularly. For this reason, the match code for verb and passive verb Items accords to a similar structure to the code for placeholder terms, but tailored to the specific needs of the extension.

On attempting to match a verb or passive verb Item, the IEPattern.match method calls the IEPatternItem.match method, which in turn calls the hasVerbFor method on the IEAttribute referenced in the Pattern. As explained above, this method will return a true value not only if the term is identical to a verb held in the verb set for the Attribute, but also if the base form of the term is listed, if the verb has a synonym in the set, or if the term is some form of the verb 'to be', an auxiliary verb listed, the word 'by' or the word 'also'. IEPattern.match computes whether a match has in fact been encountered using boolean fields set within the IEAttribute object regarding these Items. If the match method returns a true result, the code then checks fields foundBe, foundAux, foundPassive, foundMain and foundAlso within the Attribute object. For active verbs, an auxiliary verb followed by a matching main verb constitutes a match. For passive verbs, some form of the verb 'to be' followed by a matching main verb, followed by the word 'by' constitutes a match. Given that many of the original Templates contain the verb 'to be', it is essential that the verb matching code be tailored to deal effectively with this structure, in order that erroneous matches not occur. In other words, the algorithm must ensure not only that the required elements alone are present for a match, but also that they are present in the correct order. Additionally, for two of the new Templates, the word 'also' is recognised as part of the verb element, although one, like auxiliaries, that is essentially ignored once it has been established as forming part of a valid structure. It is assumed that only verb phrases featuring at most one auxiliary verb should be treated as a match because

any more is likely to imply uncertainty in the expression e.g. ‘she might have written the book’.

The end result of this first phase of the matching process is an IEMatchSet holding the set of IEMatch objects for the matched terms in the Pattern. For non-verb Patterns, this set of Matches is then passed to the appropriate IETemplateType, which uses them to generate the Updates that are then added to the current set of Updates created during the present extraction process (an IEUpdateSet holds updates identified for the incoming Message which are then executed once the matching process is complete). When it comes to Verb Patterns however, there is an extra stage to the process, whereby the set of matches is replaced by a new one corresponding to the original Pattern with which this one is related. A new IEMatchSet object is created and each Match object set according to the following algorithm. Placeholder terms are simply copied from the original Match Set, since the Match objects for these should be instantiated in exactly the same way for the new Patterns, using the information gleaned during the first phase in the process; for terms other than placeholders, a new Match object is created, this includes constant terms, pronouns, attributes and entity types. The process whereby these new Match objects are created is also complex, and uses a combination of the information associated with both the original and current Patterns. This new Match Set is then used to create the Updates to be executed. For the newly created Match objects, the code replicates the behaviour that would be executed when a Match Set is created for the original Pattern to which this one is related; that Pattern having been identified as referring to an Attribute that is also referred to by the verb or passive verb Item in this. Given that the new Verb Templates have been accurately linked with the correct Template Type, it should be safe to assume that a Match for the new Pattern can be treated as a Match for the other. The creation of this ‘artificial’ match for the original Pattern also spared the extension the difficulty of resolving anaphora and in general the functionality that is reliant on Context in the existing application.

IEPatternItem

An IEPatternItem object represents a single element within a Pattern, albeit one that may constitute a match for multiple terms in an incoming message. New Pattern Items were required to cope with the active and passive verbs, and so a new constructor was added to the class that could cope with both. One trivial addition to the class is to the array of colours used in the display of Pattern Items in the GUI to cope with the new types of Item. Also, the match method in the class has been extended for these types, both calling on the IEAttribute.hasVerbFor method, having taken a pointer to the relevant IEAttribute as parameter to the constructor; as with the other Items, the match method returns the Item Type number when a Match is encountered.

IESchema

Aside from a small amount of test code, the sole addition to the IESchema class has been to include any verbs generated within the IEGUI display; this code occurs within the DisplaySynonyms method.

IESynonymFrame

As the Synonym Frame GUI (where synonyms for schema terms to be used in the matching process are produced on user request) seemed a sensible place to offer the user the opportunity to generate verbs, a button was added to the interface. actionPerformed code within the class therefore brings up a further small GUI dedicated to the production and editing of verbs to be used in the extraction. The class also features a small amount of test code.

IETemplateItem

IETemplateItem objects represent a single term in a Template. Additional types of Template Item have been designed for verbs and passive verbs, appearing within the Verb Templates as:

`<verb>` and `<passiveVerb>`.

To deliver these new types, code has been added to the constructor of the class, and type numbers allocated to identify them. Two methods have been added that return boolean values indicating whether the item is a verb or a passive verb (isVerbItem and isPassiveItem). The method generatePatternItem has been extended to cope with the new verb and passive verb Items, creating new IEPatternItem objects and passing appropriate parameters when called from the IEVerbTemplate class. Code has also been added to accommodate object pronouns, working in the same way as that for the possessive pronouns.

All additions to existing classes are clearly marked with commentary in the submitted application

.

4.6 Installation

In order to run the extended program, a number of components are required.

To install and compile the source code, the program will require the jar files jwnl.jar and wn.jar, all of which are included with the submitted application. Additionally, the properties file within the JWNL release must be set to the appropriate version and location of the WordNet application on the machine. To use all JWNL functions, it is also necessary to install the files commons-logging.jar, utilities.jar and tests.jar, all of which come with the standard JWNL release. Naturally, the WordNet dictionary files must also be present (included on the CD –it is not actually necessary to install the program as long as the Dictionary files are in the expected location). Also, in order for the wn.jar related functions, the WordNet dictionary file location is specified within the code as: C:\Program Files\WordNet\2.1, the interface code will then look for the directory 'dict', at this location in which the WordNet dictionary files should be present. The files listed below must also be present.

To run the application from the provided jar file (IEProj.jar), it is necessary only that it be within the same directory as the 4 JWNL jar files provided, and that the WordNet Dictionary files be at the location above. Also, the files listed below are required. For convenience, the Inflection Repository element in the JAR file is constructed without the use of the Dictionary Reader, explained above, and so inflected forms displayed through the JAR demonstration will include nonsense forms suitable for manual editing.

In addition to the standard set, a number of new files are additionally required within the 'IEFiles' directory used by the existing application; for the present Project, these have been placed at: D:\\IEFiles, inline with the original code. Already, this directory contains the input files to the application. Additionally, the directory must now contain the following files: VerbTemplates.txt, LexCategories.txt, LexDescriptions.txt, DetachmentSuffixes.txt and infl.txt. Also, the Message files Message4.txt and Message5.txt are required to run the example domains included, by selecting Load – Load All – then either LibraryLoad.txt or SongLoad.txt. (These files are all included with the submitted application, as is the WordNet resource.)

The new templates are contained in the file VerbTemplates.txt, which conforms to the same structure as the Templates.txt file. The file currently contains 11 template definitions, comprising a variety of verb structures and featuring two new types of placeholder for verbs and passive verbs, as discussed above.

The files LexCategories.txt and LexDescriptions.txt hold a textual representation of the Lexicographer File information, also discussed above. These files are read

during the verb generation process within the extension, and are used to resolve the file numbers for WordNet entries returned by the wn.jar interface. The categories are listed in order of file number, so that the entries can be stored in a key indexed array, where the key is the integer representation of the file number.

The file DetachmentSuffixes.txt is required by the inflection generation part of the Project and is therefore not a necessity if this area is not required. The file contains the information from the WordNet Rules of Detachment and is used to compute inflected word forms.

Similarly, the file infl.txt is required only by the inflection component in the extension, and in fact only when the Dictionary Reader object is to be used, which is, as discussed above, not always necessary. The file comprises the AGID word list discussed above.

The present load files (LibraryLoad.txt and SongLoad.txt -used when the IEGUI 'load all' option is chosen) indicates that the message to be read in is contained in either the file Message4.txt or Message5.txt, which must naturally be present, and currently contain a messages whose extraction demonstrates the functionality of the extension.

At the end of the program run, the IEFiles directory will also contain a number of output files, mostly used for testing purposes. The files PatternTest.txt, TestingOutput.txt, VerbsOut.txt, TemplateTest.txt and SystemTest.txt provide various angles on the processes executed for a single extraction task.

In order to access a demonstration of the extended functionality, the following steps are recommended:

- copy the JAR files (and .ser file) from the cd into the same directory

- copy the WordNet folder to C:\\Program Files

- copy the IEFiles folder to the D drive

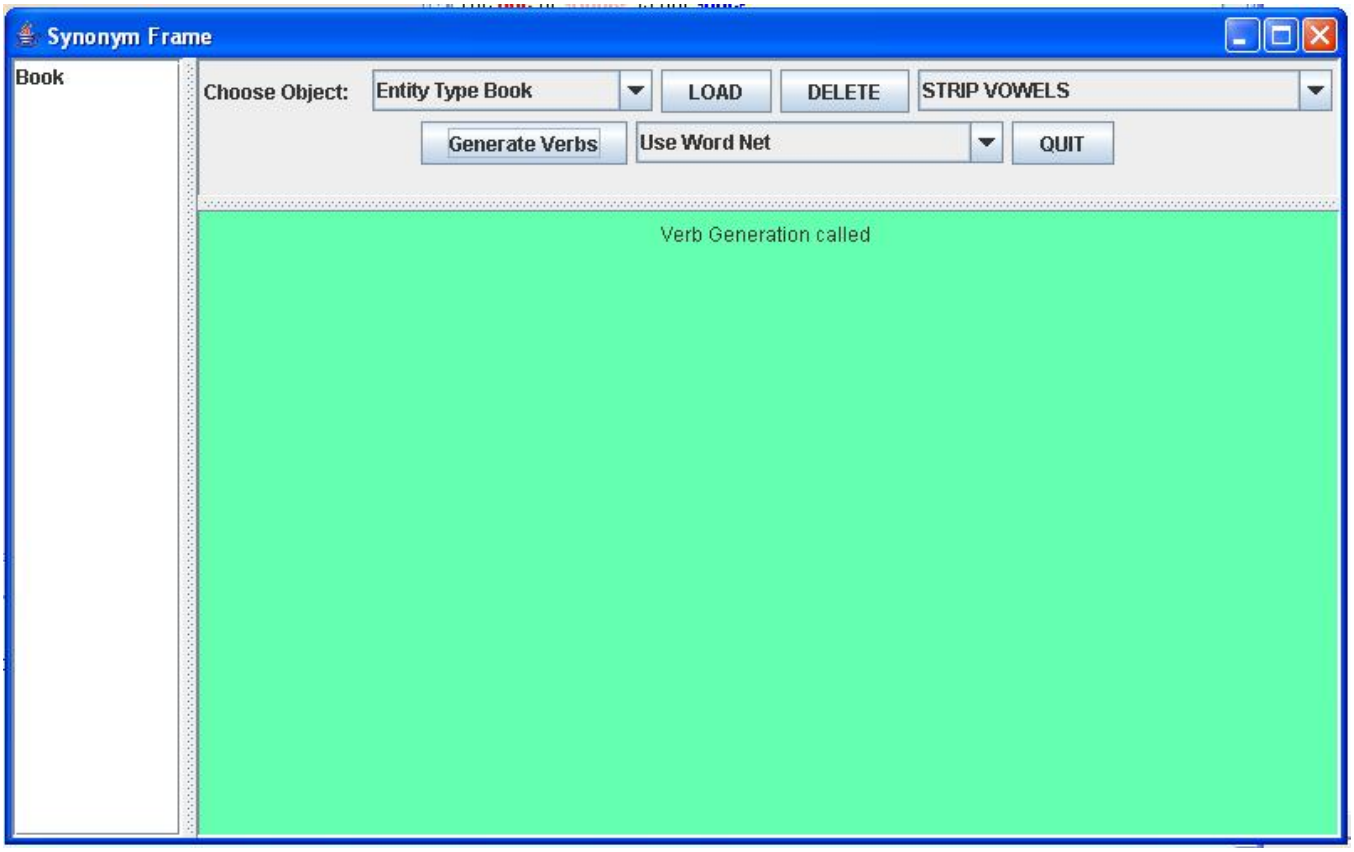
- double click IEProj.bat

- follow the instructions in the file demonstration.txt

General instructions for using the application are in the following section.

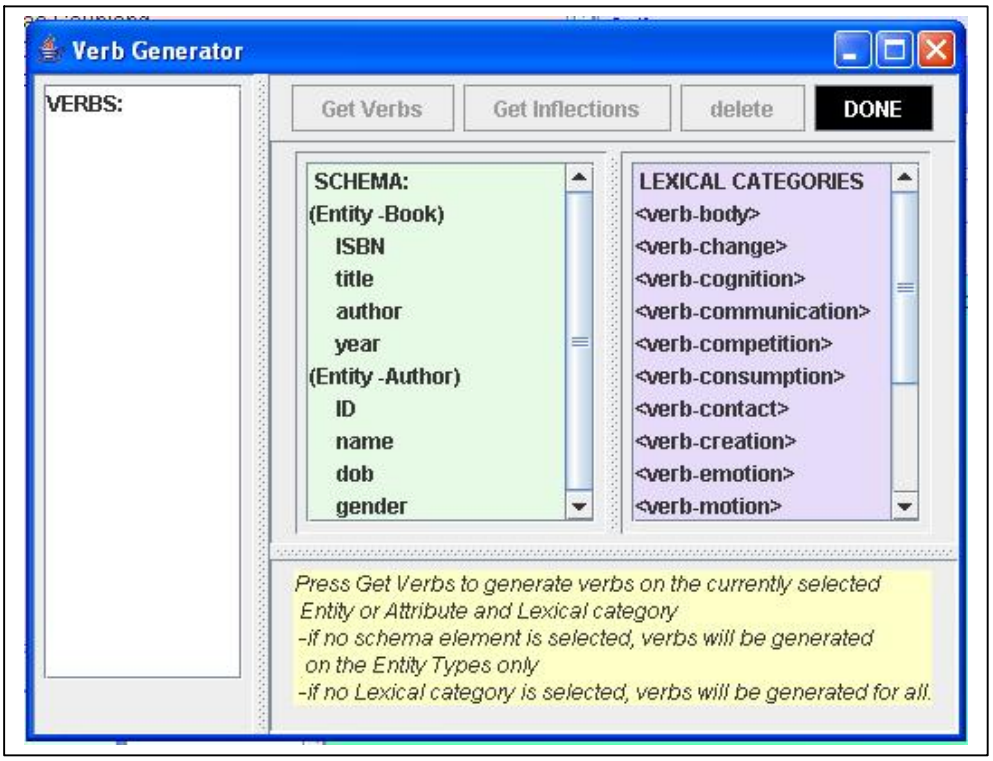
4.7 New Instructions for Program Execution

From the point of view of a user of the extended application, there are a series of additional activities required to fully explore the added functionality. The initial stage remains unchanged, the user selects the appropriate 'load' options from the main program GUI, which will now load in the additional files necessary. When the option to generate synonyms is then chosen, which is now necessary before the pattern generation process is chosen, the synonym GUI now features the Generate Verbs button:

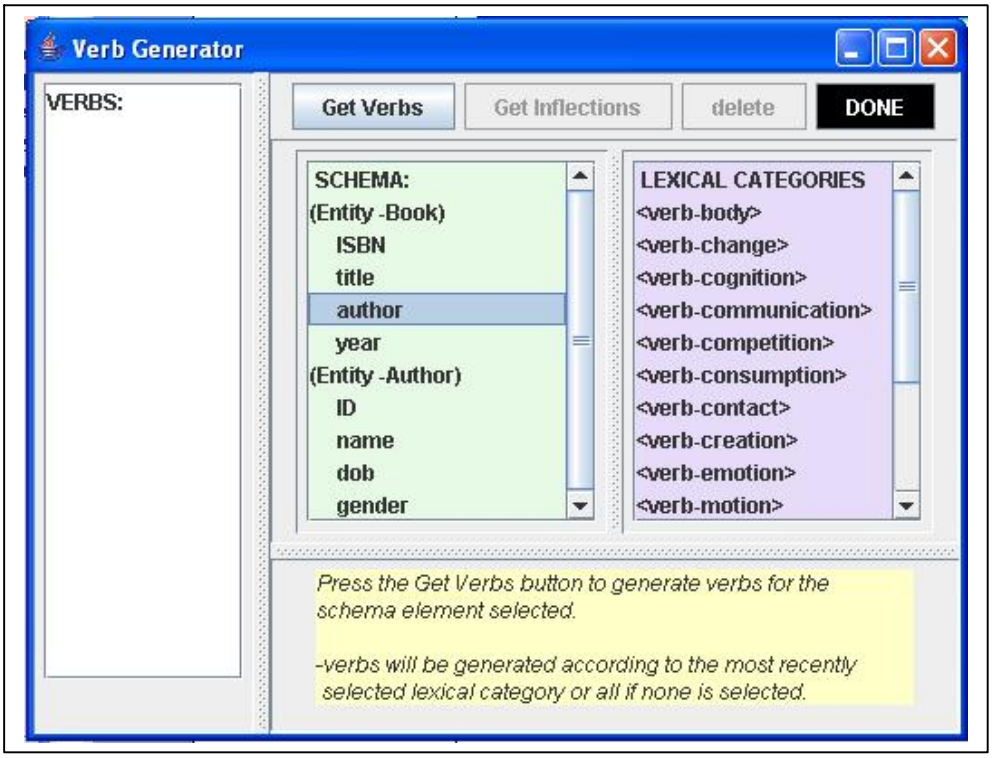


The Generate Verbs button must be selected if any Verb Templates are to be used in the upcoming extraction. On selecting this button, the user is presented with the Verb Frame GUI, from which the sets of verbs are generated and allocated to Schema elements.

The Verb Frame GUI contains four panels:

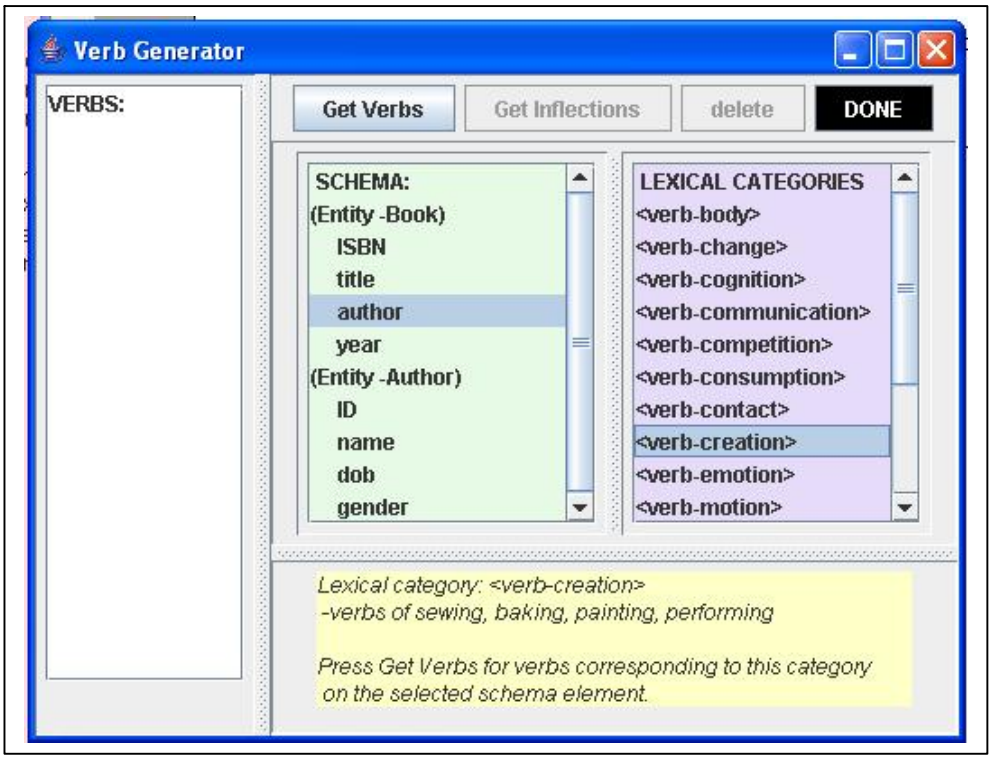


There are a number of ways in which verbs can now be generated. When the user selects a Schema element:

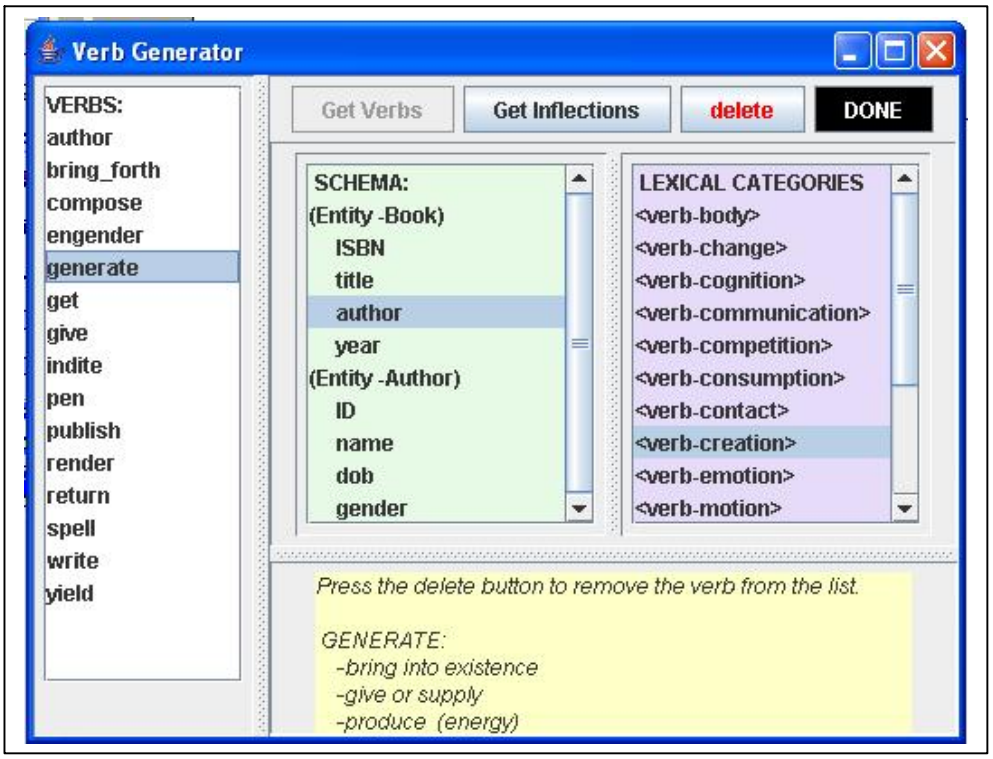


It is now possible to generate verbs on that element name.

If no lexicographic category has been selected, verbs will be generated according to all of them. If a category has been selected, only verbs corresponding to this will be included:



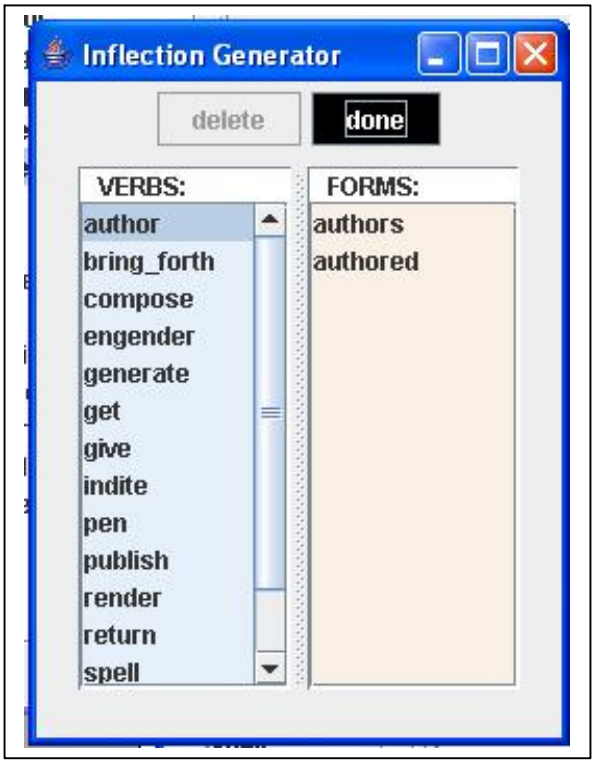
The delete button can now be used to remove selected verbs:



The information panel at the bottom provides feedback throughout.

Once the set of verbs has been generated and edited appropriately, the user must select the 'done' button to return to the Synonym Frame. Otherwise the process is unchanged –patterns must be generated and then the 'extract' option chosen.

The Inflection GUI with filtered results:



Changes to the GUI display include the following. Once files have been loaded, the Verb Templates are displayed within the GUI after the normal Templates. Patterns generated from Verb Templates are visible at the end of the normal display of Patterns in the GUI. Verbs generated are visible alongside the synonyms that have been generated for Schema terms.

4.8 Structures Incorporated

The following is a brief explanatory overview of each new Verb Template –see Appendices B and C for the list of the original Templates and the Patterns generated from these.

```
6 <spronoun1> <<verb1>> <opronoun1>

6 <spronoun1> <<passiveVerb1>> <opronoun1>
```

Both above Verb Templates are linked to the same Template Type as the following original Template:

```
6 <spronoun1> is <ppronoun1> <attribute1>
```

The subject pronoun in the first Verb Template refers to the same object as the subject pronoun in the original (the Attribute), while the subject pronoun in the second Verb Template refers to the same object as the possessive pronoun in the original (the Entity). The object pronoun in the first Verb Template refers to the same object as the possessive pronoun in the original (the Entity), while the object pronoun in the second Verb Template refers to the same object as the subject pronoun in the original (the Attribute). The verb terms in the Verb Templates identify the attribute referred to in the original.

```
3 <<value1>> <<verb1>> <opronoun1>

3 <spronoun1> <<passiveVerb1>> <<value1>>
```

The above Verb Templates are linked to the same Template Type as the following original Template:

```
3 <ppronoun1> <attribute1> is <<value1>>
```

The object pronoun in the first Verb Template refers to the same object as the possessive pronoun in the original (the Entity), as does the subject pronoun in the second Verb Template. The value terms are the same in all three Templates. The verb terms in the Verb Templates identify the attribute referred to in the original.

```
8 <<value1>> <<verb1>> the <entityType1>

8 the <entityType1> <<passiveVerb1>> <<value1>>
```

The above Verb Templates are linked to the same Template Type as the following original Template:

```
8 the <attribute1> of the <entityType1> is <<value1>>
```

The value and entity type terms are the same in all three Templates. The verb terms in the Verb Templates identify the attribute referred to in the original.

```
5 <spronoun1> <<verb1>> <<hkey1>>

5 <spronoun1> also <<verb1>> <<hkey1>>

5 <<hkey1>> <<passiveVerb1>> <opronoun1>
```

The above Verb Templates are linked to the same Template Type as the following original Template:

```
5 <spronoun1> is the <attribute1> of <<hkey1>>
```

The subject pronouns in the first and second Verb Templates refer to the same object as the subject pronoun in the original (the Attribute), as does the object pronoun in the third Verb Template. The hkey terms are the same in all four Templates. The verb terms in the Verb Templates identify the attribute referred to in the original.

```
2 <<value1>> <<verb1>> <<hkey1>>

2 <<hkey1>> <<passiveVerb1>> <<value1>>
```

The above Verb Templates are linked to the same Template Type as the following original Template:

```
2 the <attribute1> of <<hkey1>> is <<value1>>
```

The value and hkey terms are the same in all three Templates. The verb terms in the Verb Templates identify the attribute referred to in the original.

Further implications of these features are discussed in Chapter 5. Templates are listed in the order used for matching in Appendix B.

5 Linguistic Considerations

The following is a discussion of some of the linguistic aspects that are relevant to the Project. Many of the theories mentioned will be discussed in terms of their relation to the system as a whole, and not exclusively to the design of the extension.

There are a series of linguistic notions that are broadly applicable to the statements that the system is required to process. For example, those sentences that are candidates for extraction are only those comprising assertions of some kind. Statements of fact, whose truth can theoretically be assessed, or that have 'truth conditions' will be the target for the extraction process. For this reason, the sentences are liable, for the most part, to take the indicative mood. Mood is a grammatical property of a language excerpt, which indicates something about the speaker's attitude to the state of events referred to, or more generally the relation of the predicate within the sentence to the external world. Any statements not taking the indicative mood are likely to be expressions of information that is not in the interests of the present system. Only sentences declaring facts that the contributor is claiming as true will be subject to extraction.

More grammatical properties of the sentences at which the extraction process is aimed are discussed in section 5.4 on Grammar with Reference to the Present Project. Some linguistic aspects of the new Template structures are also explored in section 4.8 the Implementation chapter.

5.1 Discourse Analysis

Discourse Analysis is the field of study dedicated to the actual use of language, spoken or written. Typically, the analysis examines discourse at above the level of the sentence, concerned rather with the larger communication structures within a text. Since its early development in the 1950s, Discourse Analysis has found specialised application in many other fields such as psychology, sociology and computational linguistics. The following section, however, will confine itself to some of the central linguistic notions within the area that are relevant to the present Project.

Discourse Analysts attempt to account for the way in which language is used to achieve communication. This inevitably requires elements of different areas of study, from linguistics to psychology, to theories about wider cultural issues. The context in which the communication occurs is given a central role. Indeed, recent developments in the field have related to the fact that while historically, spoken and written discourse were studied as relatively distinct fields, some of the new modes of communication, such as text and email messages, feature some of the characteristics of both. Needless to say, this accounts for exactly the type of discourse encountered by the present application.

In general, the study of language with reference to Discourse will examine properties of the interaction that is taking place, by virtue of the exchange in which the language occurs. Any analysis of the language will therefore depend partly on the context of this exchange. Naturally, the type of language excerpts that are relevant to the present application are those that convey information, or assert propositions, as mentioned above. This comprises but a small area within the field of Discourse Analysis, which also looks, among other things, at the functional aspects of language use in social relationships and within society in general.

Given then, that email and text messages can be assumed to feature characteristics generally associated with both written and spoken discourse, a number of observations can be made about the sentences that the system is likely to encounter.

As with spoken language, the sentences are likely to feature syntax that is loosely structured and possibly even incomplete. In fact, this is one of the main reasons for the pattern matching approach, since grammatical analysis is likely to prove fruitless. Messages are liable to consist of short phrase-like excerpts, similar to spoken language, but also to exhibit such attributes of written language as pointers, conjunctions and other mechanisms whereby the reader is aided in the process of reading a piece of written discourse (for example with anaphora, which is explicitly used to resolve reference within the application). Acting as signposts indicating the

relations between the entities referred to in a sentence, these aspects can certainly aid the extraction process.

One aspect of spoken language that is problematic with regard to the system (as is borne out by the Evaluation for the present Project) is the presence of, often lengthy, sequences modifying a noun, for example:

```
The widely respected although overrated author Joe Bloggs
wrote the book.
```

(An exaggerated example, perhaps, but one that demonstrates the type of structures that are far more likely to occur in written discourse texts.) This type of structure will present the system with a real problem because the successful update process depends entirely on successful identification of the key value identifying the entity referred to. It seems fair to say that, in general, the messages are likely to feature extraneous information that will interfere with extraction.

Passive structures are more likely to be found in written language, and in fact some were encountered during the Evaluation process for this Project. A small number of passive structures have been accounted for as part of the present extension. This was expected given that the incorporation of varied verb structures was the primary objective, and one of the main ways in which the predicate in a sentence can vary is in terms of voice. The notion of voice, as well as the distinction between active and passive, is further explored in section 5.4. In this respect at least, the language handled features a common characteristic of written discourse.

Naturally, when participants are communicating in situations where they are not geographically close, the mechanisms used heavily within spoken language (in face-to-face interactions) to aid communication are not present. For this reason, written messages are likely to be more explicit, and to refer to entities using clearly named value terms, heightening the chances of a system such as this extracting the correct information.

Typically, spoken language will also feature additional words and phrases that fulfil primarily a social function rather than a linguistic one, for example to fill the gaps in a conversation. Luckily, this does not seem to be a common feature within the language captured by the present system, and so it does not interfere with the extraction process.

One aspect of both spoken and written discourse that is in the interests of social interaction, but not in the abilities of the system, is the practice of avoiding repetition, something users of a language do often. This is generally the reason for using mechanisms such as reference, and avoids sentences that are cumbersome. However,

for the purposes of the system, it requires the resolution of indirect references, achieved through the modelling of the discourse context and this features considerable complexity.

Discourse Analysis describes statements in more subtle terms than grammar, i.e. as being coherent (or not) within the discourse in which they occur. In this sense, valid sentences are considered to be acceptable rather than grammatical. This notion lends further credibility to the approach taken within the present system, avoiding as it does the classification of sentences in purely grammatical terms. The set of Templates used can then be viewed as a definition of those sentences that are acceptable within interaction with the application. In this sense, it defines a domain of its own, which is built on top of the Schema and other inputs to the program. Given that the application is designed for use within a known subject domain, it seems reasonable to have modelled the understanding of the messages in terms of some linguistic characterisation of that domain.

The resolution of pronouns in terms of context within the application (already in place prior to the present extension) uses some of the 'tried and tested' theories explored within the field of Discourse Analysis. The study of Narrative Analysis looks at how communication is achieved within longer texts (although not necessarily linguistic), using mechanisms such as reference. The way in which such structures are understood can also illuminate the reason for the ordering of the Templates in the matching process (the Template order is listed in Appendix B). Where explicit values representing entities are provided, there is naturally no need to attempt to evaluate what a pronoun stands for, and the explicit values are simply accepted as they are. However, when a pronoun occurs, a small amount of computation is necessarily performed on examination of the term (establishing, for example, gender, as an indication of what is being indirectly referred to), so that it can be made sense of. For this reason, it makes sense to eliminate those containing pronouns first, since their processing is more discriminating than that for values.

There are a vast range of subtleties of expression that are facilitated by natural language, and the extent to which these subtleties are accessible to speakers or hearers of the language varies according to many linguistic and contextual issues. As far as the system is concerned however, there are simply statements that it can process, and statements that it cannot. In this sense, the program defines a very strict grammar for its own discourse, which determines entirely what is acceptable and what is not. For this reason, the overall aim within the development must be to extend this 'grammar' to account for all those statements capturing valid updates to the data.

5.2 Linguistic Theory

While the present application has been described as generally avoiding syntactic parsing of incoming statements, it is not the case that it is entirely free from any notion of syntax. The set of Templates are essentially a structural description of statements that are acceptable for processing. In this sense, the application has its own limited description of a grammar for the domain that it models. Whereas with natural language, a grammar must cater for all sentences that are possible within the language, this system is content to dictate exactly that set of sentences in which it is interested, and to ignore the rest.

During the 20th century there was a general tendency within the study of English Language to define grammar as descriptive rather than prescriptive. That is, grammar that should describe the structures we use to communicate through language, as opposed to dictating how it should be used, as had previously been taught. From this point of view, a linguistic statement can be viewed, not in terms of whether it is grammatically 'correct' or 'incorrect', but perhaps instead in terms of whether something comprehensible has been communicated (in which case the grammar should be able to account for it). A natural parallel can be drawn here with the present system, in which the classification of sentences in grammatical terms is absent, with statements either being comprehensible to the system, or not. With this in mind, the system could be characterised as a participant having limited language abilities, for whom that set of linguistic excerpts that are understandable are described in the form of the featured Templates. Given the purpose for which the system has been put to work, its limited abilities within the wider field of language are not only unproblematic, but are actually an advantage, dispensing as they do with unnecessary processing activities. As far as this application is concerned, anything other than those sentences that it can process is irrelevant. As indicated in chapter 2, the recognition of subtlety can be a complicating factor in systems such as this, what is sought is simplification, even if it is at the expense of understanding.

Noam Chomsky, through his pioneering work in the field of linguistics, defined the grammar of a language as a finite set of rules that accounts for the infinite number of potential realisations in natural language sentences. (See Chomsky (1965) for an introduction to these theories.) *Competence* within this theory is the knowledge about the language held by a speaker, while *performance* is the actual use of the language. If the domain of discourse modelled within the system (for one Schema) is considered in these terms, *competence* would be the system of Templates, and *performance* the sentences processed. However, given the nature of the structural rules employed, the only way in which the sentences processed could be theoretically infinite, would be through the vocabulary recognised, since the number of structural variations for a single extraction process is necessarily finite (although the potential range of

Templates across executions is not finite). The possibility of an infinite number of terms being recognised is real, since the user is free to source these from different resources, and new terms are always being added to the language. In this way, the Template structures in the application can be thought of as defining the grammar for a very narrowly used variety of the language, although this analogy is purely for illustration purposes.

All communication that can validly occur in 'dialogue' with the system, is therefore described by the Template system employed for a given domain. The task then, in developing the application, is effectively defining and implementing that set of discourse that is valid. For linguists, the language behaviour of all of the people in the world can potentially have an impact on this, since their account must attend to all legitimate uses of language. Also, there is a sense in which a language 'belongs to' its speakers, in the sense that they can and do influence the continual evolution of the language, and that their use of it to communicate is necessarily valid, whether or not it fits in with any particular model of grammar. (If a sentence that is used to successfully communicate and the grammar does not account for it, the fault lies in the grammar and not in the language excerpt.) Unlike linguists though, developers of this system are free to conform to the 'prescriptive' approach, defining exactly those structures that are acceptable. (Although a future possible area for development is in equipping the application with the facility to 'learn' the Template structures through processing natural language excerpts, in which case the approach would definitely be 'descriptive').

The set of possible sentences relevant to the application might then contain all and only those sentences that indicate information that is expressible in terms of the database held for a given domain. In other words, in order to be complete, a 'grammar' (system of Templates) for the system would account for all expressions, in natural language, that describe an update (or insert etc) to the database. Whether or not it is possible to arrive at this description of the set of acceptable forms, corresponding to the various means of expressing information that can be held in a given database, is at best contentious. While the application is still within its development period, it seems at least possible in the context outlined, that a set forms accounting for the great majority of those most likely expressions of relevant information can feasibly be aimed at.

5.3 Semantics

As mentioned in the Survey chapter, the WordNet lexical database attempts to model lexical items in a structure that reflects the way that speakers of the language conceptualise their meanings. This notion is relevant also to any language processing system, such as the present Project, that models linguistic information in terms of a schema. Many recent Information Extraction systems have also relied on the related concept of Ontology. Indeed, the notion of *schema* has its own distinct sense within the fields of Cognitive Linguistics and Semantics, where a *schema* or specifically *image schema* is the description of a cognitive structure that facilitates understanding, one of the means by which we comprehend the world (see Taylor 1989). This notion has proved influential within the field of Semantics, which has close relation to Cognitive Psychology.

In addition to the fact that these linguistic notions generally reinforce the legitimacy of modelling information acquired through linguistic media in schema terms, there is an additional aspect of relevance to the present application, and indeed to other similar systems. Semantic theories have often framed the notion of meaning as being a property not only of the language, but of the context in which a language excerpt occurs. Meaning is defined in relation to a *domain*, whose structure and central concepts will be described in the form of a *schema*. In this sense, the *domain* referred to is a means of characterising meaning, of drawing boundaries around the various subsections within the greater world of meaning.

Although the notion of domain within the present application refers rather to a domain of discourse, i.e. language excerpts on a certain subject (which may be more or less general), the possibility of successful recognition of language input to the system must be surely fortified in light of these linguistic notions. As outlined in the preceding chapters, the application attempts to process language that falls within a known domain, and so the interpretation of this is likely to be more accurate than if the domain were either not known or more general. It has been accepted from the outset that inputs to the system will conform to a narrow range of discourse, which is naturally justifiable in any case given the intended deployment of the application, i.e. for use within a Web application with a specific remit of its own.

Further linguistic qualification regards, in general, the employment of a database within an IE system. Semantics offers the distinction between notions of *sense* and *reference* (originally from Philosophy of Language, especially Gottlob Frege). *Reference* denotes the mechanism whereby a linguistic term refers to/ denotes or identifies some abstract or concrete ‘thing’ (which can be a concept rather than a physical object) in the world. *Sense*, on the other hand, is abstract notion comprising

the speaker/hearer's understanding of the term, the mental mechanism whereby *reference* is delivered. For example, the words in bold in:

I attend **the University of Glasgow.**

refers to or denotes Glasgow University, whereas the hearer's understanding of the phrase, their mental processing of the language excerpt, can be defined in terms of *sense*. The *referent* of the phrase is the actual physical entity of the University in the world, whereas the *sense* associated with the phrase is a mental construct grasped by speakers/hearers of the expression. This implies also that the *referent* (the University) can potentially have a set of several *senses* via which it is *referred* to (for example earlier in this sentence where I used the term 'the University', to refer to the same thing!) Although not necessarily a mechanism that is expressed explicitly in the IE context, this concept is central to the way in which a database, or indeed ontology, is used to model the information captured from linguistic utterances.

With regard to the present application, the idea that the same entity can be referred to in more than one way, is essential to preserve the accuracy and integrity of the data. The main process by which this has been achieved within the system (which was implemented in advance of the present Project) is in the resolution of pronouns using the Context object. In general, recognition of the fact that terms in the sentence may be referencing some entity already listed within the data, which is delivered by the use of keys, is demonstration of the same linguistic phenomenon. (For example, when a named value, for an attribute functioning as a key, occurs in a message, the program will check the database to see if an entity having that value is already listed and, if so, update this rather than inserting a new one, which will only happen if an entity having the same key is not already present, a basic relational database principle.)

Saeed (1997) Chapter 2 provides an introduction to the above Semantic theories.

5.4 Grammar with Reference to the Present Project

A number of considerations on grammatical aspects of the structures modelled within the Project now follow.

Within the extension code, the verb and passive verb items within sentences that are recognised feature a number of assumptions. For active verbs, it is possible for a main verb accompanied by an auxiliary verb to be recognised, e.g:

```
She did write Persuasion.
```

In order to deliver this function, it was necessary to store a small set of auxiliary verb forms that it would in the interests of extraction to recognise. WordNet does not actually store the full set of auxiliary verbs within its dictionary so these forms cannot reliably be queried in this way. In any case, it was expedient to specify those exact auxiliary forms that were to be recognised for a match. This is due to the fact that auxiliaries can express various degrees of uncertainty and intension, whereas the only forms that should result in updates (within the present functionality) are those featuring relative certainty. As the algorithm stands at present, any of the auxiliaries:

```
has had did does have
```

followed by a main verb within the relevant Verb Set will constitute a match. The program simply ignores the auxiliary and uses the main verb as normal to carry out the matching function. Any other auxiliary forms encountered will not result in a match, for example:

```
He should have written the book.
```

It is also assumed that more than one auxiliary form preceding the main verb indicates a lack of certainty, or certainly some more complex form of expression, and therefore should not result in a match. Of the primary auxiliaries:

```
be have do
```

together with their various inflected forms, the only one that is not recognised within the algorithm is ‘be’. The reason for this is that the specific brief for the extension was to recognise exactly those sentences not having their predicate in the form of the verb ‘to be’. However, the word is used within the processing of passive structures, explained below. None of the set of modal auxiliaries:

```
can could may might must shall should will would
```

are recognised within the code, as these are used to form the tense, mood and voice of a more complex verb structure, and are beyond the scope of this Project.

The Pattern item corresponding to passive verbs within the extension actually occupies a larger portion of the passive sentence than just the verb phrase. The voice

of a clause is a function of the semantics of its elements. Voice describes the relationship between the predicate in a sentence and its subject (and object in this case). An example of a sentence having active voice could be:

```
She wrote Persuasion.
```

With its passive counterpart being:

```
Persuasion was written by her.
```

In the active sentence, the subject of the sentence ('she') is the actor of the verb 'write', and the object ('Persuasion') is the recipient. In the passive version, the subject of the sentence ('Persuasion') is the recipient, and the object ('her') is the actor. The relationship between predicate, subject and object in the second is the converse of the relationship in the first. As far as the extraction process is concerned though, these two sentences are providing exactly the same information, and should therefore result in the same update to the data. For this reason, the new set of Templates models pairs of structures expressing the same update but with alternate voice.

Those verbs that take the passive voice tend to be transitive verbs, i.e. verbs that take an object. Some verbs are exclusively transitive or intransitive, whereas some can behave as either. For example, in the following sentence:

```
She was writing a book.
```

the verb 'to write' has taken an object, and is therefore behaving as a transitive verb. However, in the following sentence:

```
She was busy writing.
```

the verb behaves as an intransitive verb, taking no object. There is an assumption within the extension that the verbs that are deemed relevant will be transitive, since they express the relationship of an attribute to its entity, where in grammatical terms the attribute is indicated by the predicate, and the object the entity (the actor of the verb in the above example being the author value). This means that it is justifiable to assume that passive structures will indeed be valid in this context.

In order to identify passive structures during the matching process, the program checks for the following: some form of the verb 'to be', followed by a main verb (that is within the relevant set), followed by the word 'by' (and then the next item in the pattern is checked as normal). The main verb in a passive sentence will normally be the past participle, but for the purposes of the extension, any form of a listed verb will constitute a match. For example:

```
It was written by Jane Austen.
```

Although it is not strictly part of the passive structure in grammatical terms, the checking for the word 'by' is essential to identify this as a passive structure. This is also part of the reason that active verb Templates do not recognise the verb 'to be' as a valid auxiliary, because it can interfere with the recognition of passive forms.

In purely grammatical terms, the above sentence can be described in the following way:

```
S (It) P (was written) C (by Jane Austen)
```

where S stands for Subject, P for Predicate and C for Complement (what verbs that are intransitive can take instead of an object –the sentence would make sense without the complement, but would have a quite different meaning, whereas an object is a necessary element that cannot legitimately be removed from the sentence). The word 'by' is therefore not part of the verb phrase, but for the purposes of the application, the word is sensibly part of the 'passive verb' element, with the final two words only, functioning as the object. Again, this is another way in which traditional grammatical parsing might let the system down, since it does not accurately represent the relation of the sentence to the data. An analysis more suited to the application would be:

```
S (It) P (was written by) O (Jane Austen)
```

which does, in fact, correspond to one of the new Templates introduced (O stands for Object).

Prior to the extension, the bulk of the sentences processed conformed roughly to the following kind of structures:

```
S (The author) P (is) C (Jane Austen).

S (Jane Austen) P (is) C (the author).
```

In these sentences, the structure is Subject – Predicate – Complement because the predicate takes the form of the verb 'to be'. The noun phrase at the end of both examples is a Subject Complement, which modifies the Subject of the sentence via the verb. Both sentences identify an attribute and provide a value for it.

For an active sentence corresponding to one of the Verb Templates, the structure is as follows:

```
S (Jane Austen) P (wrote) O (it).
```

where the noun phrase containing the object is naturally the recipient of the verb phrase comprising the predicate and having the subject as actor.

As has been indicated in chapter 4 on the Implementation of the structures, there is an assumption in the active sentences that the subject refers to the Attribute, and the

object to the Entity, whereas in passive forms, the subject refers to the Entity and the object/ complement to the Attribute. Clearly, this will only be legitimate when the Templates have been appropriately constructed and associated with Template Types accordingly.

The details of each new structure accounted for can be found in section 4.8 in the Implementation chapter.

6 Testing

6.1 Testing Methodology

A variety of mechanisms were used within the application to carry out a series of different tests. Frequent testing was part of the daily process of developing the extension, since the complexity of several parts of the program meant that a small change could have larger consequences. The impact of any amendments made therefore had to be continually monitored. Also, the functioning of the original part of the application had to be regularly verified, since any changes made may also have affected this.

As is outlined below, there is test code that is integrated within the program run, contributing to the System tests, and unit testing, intended to be carried out separately to the normal execution of the program. A number of output files indicate the results of these, for both unit and system tests. Additionally, some of these outputs have been utilised within the Evaluation process, as explained in chapter 7. An overview of these outputs is provided in Appendices D and E.

The extension was developed using the existing 'library' domain (together with its already created input files for Schema, Templates, Template Types, Context, Synonyms and Data). It was the tested with this domain, and then tested again and evaluated with a second, new domain. The reason for developing the extension using these existing inputs was that it was far easier to measure progress when results could be more readily compared to those for the 'tried and tested' domain. Also, given the nature of the extension (i.e. added functionality that is built on top of the existing code, and that is in fact generated from the results of the existing algorithms), it made sense for it to develop using an example for which the existing functionality already delivers results. The new Patterns, whose generation and matching essentially comprise the result of this extension, require Patterns generated according to the program as it stood prior to this project, in order to be instantiated. Given the complexity of the algorithms featured both in the original code and in the extension, it was naturally helpful to have structures already successfully incorporated as a reference point.

In terms of what exactly the tests are hoped to establish, the following serves as a summary:

- that the instantiation of the new Templates is carried out successfully

- that the verbs are generated successfully and attached to a Schema element

- that the new Patterns are created appropriately and linked to the relevant original Patterns

- that the matching of new Patterns is achieved accurately

- that updates based on successful matches are accurately defined

- that the context is maintained appropriately when new Patterns are processed

- that elsewhere the system continues to function as normal.

The complexity within much of the extension, and the application itself, means that a number of aspects of the procedure are easier to examine in isolation from the rest. For example, the Pattern generation process has its own dedicated output file that outlines information relating to how Patterns have been generated during a particular run of the program. This is particularly useful in the extension, because it sheds light on the building of the new Patterns of old Patterns, using the new Verb Templates as a blueprint. Similarly, the verb generation process has been afforded a dedicated output, summarising the verbs generated on particular attributes in the Schema. The file overviews the various processing activities that have been carried out, through the Verb Frame GUI, in order to arrive at this. Also, a summary of the instantiation of the new Template structures is output to file for testing purposes. These files, along with those representing the system tests explained below, also provide a detailed log of the events occurring during a single run of the program. The unit tests also have their own dedicated output, the details of which follow.

The secondary part of the Project (the Inflection generation) has not been subject to testing other than for demonstration purposes.

6.2 Unit Tests

The way in which the new code is built in with the original code made it problematic to carry out detailed unit tests on all classes and methods added. For this reason, unit tests were provided through class IEVerbTest for those classes that could be assessed in this way. Otherwise, testing has been integrated within the program itself. For this reason, the class contains test methods that examine only some of the new classes.

The application's main method was provided with code creating the IEVerbTest object that would execute a run of unit tests. The constructor is passed a PrintWriter reference to output the test results to the file TestingOut.txt, created in the IEFiles directory containing all of the program's input files. The IEVerbTest object holds a reference to both JWNL Dictionary and wn.jar WordNet objects for use within the tests. The main method calls the IEVerbTest testObjects method, which then creates instances of classes IELexTest and IEDictTest, whose methods are in turn called from within the callTests method. The classes IELexTest and IEDictTest share the Superclass IESuperTest, an abstract class declaring the methods runTests and overview, and taking a reference to the PrintWriter object for output. The Subclasses additionally feature methods for each of the classes that their tests are designed for. Class IELexTest tests the classes IELexicographer and IELexicalItem, while class IEDictTest tests classes IEDictAssistant, IETenseProcessor and IEVerbProcessor. The tests aim to exhaust the range of reasonable inputs on which to test those objects and methods featured, presenting valid, invalid and boundary values as appropriate. Once the tests have been run a short textual summary is additionally output to the file.

Results for these tests are outlined in section 6.4, while the documentation is listed in Appendix D. The files list the details of methods called, parameter values passed and expected results together with the actual results.

6.3 System Tests

As indicated above, the nature of the extension made full unit testing difficult. For this reason, and to provide a log of the activity during a program run, system tests were included in various locations. As the instantiation of some of the new classes depends on the successful execution of objects already defined, this seemed a necessary approach. The various output files are included in Appendix D.

The class IEGUI is equipped with a PrintWriter object for the output of general system test data to the file SystemTest.txt. The selection of any of the components in the main GUI is included in the output as it occurs, so that the file will contain a log of options selected by the user during a single extraction process. A reference to the PrintWriter is then passed to the IESchema once instantiated, so that it is accessible to other classes for output during the program run. When the IESynonymFrame GUI is loaded, the user's selection of the Generate Verbs button is written out to the file. The instantiation and use of the IEVerbFrame GUI is also included, writing out the occurrence of verb generation and the setting of verb sets within Attributes. Methods called during the generation of Patterns based on the Verb Templates are logged within the file. When the option to extract data is chosen, the details of methods executed within new classes or methods is additionally written out, including within the IEAttribute and IEVerbSet objects during the matching process, and the IEMatchSet, IEPattern and IETemplateType objects when a match is encountered. Finally, the text that is displayed in the IEGUI panels relating to feedback and updates is copied to the file, as is a summary of the extraction results.

The file PatternTest.txt, also created during the program run, contains details of the generation of the Patterns based on Verb Templates. For each new Template, the file lists each original Pattern with which it has been linked, and the resulting new Pattern realisations. In addition to assisting the testing process, this file also illuminates the process whereby the new Patterns are created.

File TemplateTest.txt contains details of the Verb Templates loaded, as well as an overview of the ordinary Templates, once these have all been instantiated.

The file VerbsOut.txt logs the finer details of the verb generation process, outlining each stage executed as well as the actual verbs terms generated and the Attributes with which they have been associated. The file also indicates such activity as the deletion of verb terms already generated.

6.4 Test Results

While the system test outputs outlined above were used primarily for informative purposes during the development process, providing insight into the details of the various algorithms employed, the unit testing was carried out, with amendments being made to the code, until the methods and classes behaved according to expectations. Given that the run of the program features a series of processing stages, it was extremely useful to have the details of the system tests to hand during the development process, although they now provide a concrete testament to what is actually happening within the application. While rigorous unit testing is naturally desirable throughout all areas of the application, it is, in reality, difficult to arrange given that so many aspects of the program are dependent on one another. It is hoped that the testing demonstrated provides adequate satisfaction as to the quality of the extended software delivered as part of the present Project.

In terms of analysing the success of the extension in light of testing, it is important to outline the criteria for assessment. Ideally, the system will cause appropriate updates on extracting data from valid structures. Also, and in many ways more problematic, it must not result in updates that are incorrect or invalid. In other words, it must not either match Patterns with invalid incoming structures, or instantiate updates incorrectly based on the results of the matching process. As implied above, the avoiding of spurious updates is in many ways more difficult to achieve than is creating correct updates. This process is aided largely by the appropriate ordering of the Patterns checked against during the matching process, as is outlined in chapter 4.

Messages used in the system tests were intentionally long because one of the more problematic aspects of the application is the maintaining of the context object, developed prior to this extension. The length was maintained to ensure that the new Patterns provided the already existing algorithms with the requisite information for this task. The messages also contain repetition of concepts and updates, and are generally contrived to demonstrate and stretch the abilities of the extension. Also, given that the new Patterns comprise but a small set of the whole number of expected structures, the likelihood of encountering more than one (or even one) of them within a given message is slight.

Further details are provided in the Evaluation for the Project, in chapter 7 of this report.

7 Evaluation

7.1 Evaluation Approach

The evaluation approach was to solicit a series of messages from contributors, on a given domain, that would be used to further test the system. The extension was developed using the 'library' domain that had been integrated into the application prior to the present Project. The decision was therefore taken to execute a twofold evaluation method, the second phase of which would use a new domain that was previously unseen by the system.

It was necessary then, to acquire two sets of messages, one corresponding to the original 'library' domain and one relating to a new 'song' domain (see Appendix F). To deliver this secondary domain, it was naturally necessary to devise an appropriate Schema, and to arrive at a set of input files expanding the details of the new domain. It should be stated at this point that the new Schema, although comprising a fresh test for the system, is, to a degree, necessarily contrived, not only to demonstrate the extended functionality, but also to integrate successfully with the system as it stands. This is explored further in section 7.3.

An email message was sent to contributors guiding them in constructing the types of message required, but hopefully not restricting the responses unnecessarily; the email is listed in Appendix E. Within the email, an example of the type of message sought was given, but the participants were encouraged to phrase their message in a way that seemed appropriate to them. While this encouragement has the obvious disadvantage of increasing the likelihood of messages that the system is unable to process, it would defeat the purposes of the exercise to dictate too dogmatically the structure of what was expected, and would naturally render the evaluation process somewhat redundant. However, a further qualification to this relates to the fact that this extension is a fractional component within a system that is subject to ongoing development, and so a certain degree of failure to successfully extract information is expected.

Appendix E details the Evaluation results.

7.2 Evaluation Results

On initial examination of the received messages, it was immediately clear that some of them would prove problematic. However, the system was run with each as it stood, the results documented and then the messages edited before being subject to the process again. All results as well as the details what changes were made during the edit process are listed in Appendix E.

The results were, at best, mixed. Conclusions drawn are outlined in section 7.3, as is an indication of what work would have been undertaken if more time were available. The most common problem encountered was in the incorrect matching of incoming structures, resulting in inaccurate updates. Also, structures that correctly resulted in matches were not followed through to the appropriate update because previous sentences were not extracted successfully (because they did not conform to any of the structures so far modelled). This was due to the inability to resolve contextual information, essential within texts featuring several sentences. While problematic, this latter problem is bound to recur throughout the development process for a system such as this, especially with longer messages, since there will likely be at least some sentences that cannot be processed, elements of which are referred to in subsequent sentences. For example, if a sentence, which does not conform to any of the structures modelled (and therefore not resulting in any update), introduces a reference to an entity, such as the name of a book, that entity may be referred to indirectly in later sentences. Where pronouns are used, the resolution depends on the context update having been successfully carried out on previous sentences. In this way, failed matches are likely to have a 'knock-on' effect within a single extraction process.

Nevertheless, the evaluation process did see the successful matching and update for a number of the new Patterns, and has surely informed any future progress in this area.

7.3 Evaluation Conclusions

The Evaluation process has caused a number of conclusions to be drawn, all of which are outlined below.

To begin with what has been achieved, the system can successfully extract information from several examples of sentences corresponding to the new structures. This has been achieved within the brief, i.e. without compromising or greatly complicating the present processes. The facility by which generation of appropriate verb terms is carried out has also been implemented successfully. The new structures accommodated are capable of being recognised and successfully processed also when occurring within longer messages, and are subject to the same functionality as the original set. In addition to the verb forms indicated at the start of the Project, the extension accounts for passive structures, over and above the main brief for this part of the extension.

Turning now to the various issues raised in the Evaluation process, an overview follows.

One clearly visible complication is that a noun can have an unpredictable set of verbs associated with it (retrieved by querying WordNet), this is because verbs are generally 'more polysemous' than the other parts of speech. For example, the verb 'write' used in the 'library' domain, is listed as being associated with the verbs 'get' and 'give', among others. However, the verb 'sing', used in the secondary domain, returned only a small set of verbs that all seem to be entirely appropriate. The implication of this is, of course, that inappropriate matches will result. Although the user is free to delete verbs generated on an attribute name so that these will not be used in the matching process, the algorithm does check input verbs for membership in the same Synset as a verb generated. In other words, even when the user has deleted inappropriate verbs, the matching process will still return a positive result if a verb is encountered that is in the same set as one of the verbs that remain. Given the nature of some of the Templates included, the problem is likely to occur, since some feature only verbs and placeholder terms, in which case everything other than the verb is considered feasible (not subject to discrimination). One option to eliminate this problem would be to tailor the match methods to return a positive match only when one of the verbs selected is encountered, however the risk is presented that some appropriate matches will not be made.

A general risk for the extension relates to the fact that the successful match process depends on whether an appropriate set of verbs has been generated and associated with the relevant attribute. This, in turn, depends on the quality of the user input, as well as the presence of an accurate set of Templates for the Schema. The

dependency on user input is thought to be inevitable, since there can be no automated process for verb generation that would reliably select appropriate verbs. In general, the user is required to be well acquainted with the processes and concepts through which the program is delivered, although this is possibly true for the system in general and not just the present extension.

As indicated in chapter 4, there exists within the extension, the facility to add a set of verbs to Entity Types, as well as Attributes in the Schema. Although only those that are attached to Attributes are employed within the present extension, an observation resulting from the Evaluation process suggests that this might be incomplete. It is possible to add sets of verbs to any number of attributes in the Schema, however it appears that an assumption has been made throughout the development process that the verbs were likely to be attached to only one Entity. The sentence:

```
It was written in 2001
```

illustrates a situation in which the verb is naturally associated with either the author Attribute or Entity Type, while the Attribute to be updated is actually the date. Also, the sentence:

```
It was produced by Joe Bloggs
```

could easily be applicable to the song domain if the entity Song also had a producer attribute, in which case two sets of verb Patterns will be necessary. This demonstrates the limitations in the approach taken, and points to, both the generation of verbs on more than one Attribute, and to a more complex definition of the relation of new Templates to the Schema. The structure of the Verb Templates is dependent on the relationship between an Attribute and its Entity Type, and also the Attribute for which a data value is indicated, as with the example above. Managing the relations between the database elements might also become more of an issue, for example, is the term 'written' in the above example referring to the attribute or entity 'author'? In terms of the actual referents of the language, these are the same thing, but the relation in database terms is defined in terms of foreign keys. In any case, the immediate conclusion that must be drawn is that the relation between the data and the language structures is subtler than that modelled in the extension. Perhaps some system in which attributes were associated with a subset of the Templates that accurately express their particular relation to the entity might be an option. However, this would run the risk of greatly complicating the process, potentially compromising the 'lightweight' aspect of the application.

The fact that the Patterns generated from Verb Templates are only checked for a match after the original set of Patterns have been exhausted is an area that might benefit from further examination. Presently, this is just a consequence of the way in

which the extension has been added to the code, however it may be beneficial to order these within the matching process differently. As mentioned, the order in which Patterns are presented during the match process has a huge impact on the success or failure of an extraction attempt.

Throughout the development and testing process, there has seemed a level of relative success in processing the desired structures, however there is a danger that the extension has been developed with the 'library' and 'song' domains/ schemas in mind. This could mean that the apparent success is deceptive, in that the method by which the extraction is achieved will not necessarily travel between different domains. It is, at the very least, true to say that certain domains will naturally lend themselves more easily to the extension logic, although, again, this is true for the application in general.

In retrospect, the Evaluation process was likely to be tricky given that the new Patterns constitute such a small segment of the total set of structures that the system can recognise. For this reason, the chances of encountering the new Patterns in the solicited messages were possibly quite slim.

The main issue that has arisen through the Evaluation for the extension, is that the approach taken was perhaps simplistic. While this is naturally problematic, it is reasonable to suggest that it is, in some ways, a necessary evil, in moving the development at all in this new direction. The work of the present Project involved incorporating whole new levels of linguistic complexity within the application. In light of this, it was probably inevitable that the exact nature of that complexity be discovered through this early attempt at accounting for these structures. Having said this, there has been a degree of success in the functioning provided, as is borne out in the Evaluation results. It seems therefore fair to describe the Project as a qualified success, although one that possibly identifies more issues than it solves.

There now follows an indication of further work that would have been attempted given unlimited time.

The extension of facilities such as vowel stripping, provided for synonyms, to the verb forms generated.

The structures that were originally intended for inclusion within the extension, but that were not possible within the constraints. Specifically, the structures:

```
spronoun verb the entityType

the entityType was verb by opronoun
```

in which the problem is presented by the `entityType` element. As indicated in chapter 4, the key to the linking of new Templates with Template Types is to relate them to types having the same parameters (except verbs, which are linked to attributes). The present set of Template Types in the system features only one having an `entityType` parameter, which also holds an attribute and a value parameter. Given the constraints and the way in which the incorporation of the new structures has been implemented, this means that only Verb Templates having those same parameters could be associated with this type. In other words, the incorporation of these structures was not possible within the constraints given. For this reason, their handling would need to accord to a more complex algorithm than that adopted for the present extension.

Exploration of verb frames in WordNet as a facility for assisting the verb generation process. (Verb frames are sentence structures that are listed as appropriate to verbs held in the WordNet dictionary.)

Inclusion of the facility to load verb forms in at the start of the program run, in the same way that synonyms are currently loaded, and for the user to enter the verbs manually through the Verb Frame GUI.

Evaluation results are listed in Appendix E.

8 Conclusion

The main aims for the present Project can reasonably be said to have been achieved, with some qualifications. A set of new verb structures has indeed been integrated, and these are subject to the same range of processing within the extraction process as those originally existing. While the inclusion of these structures has been generally successful, the impression is that they do not go quite far enough, as explained below. The application still achieves the full range of functionality as prior to the extension. Additionally, the requisite research on inflected forms has resulted in concrete facilities included within the submitted application.

As it stands, the extended application has been equipped with the ability to process a new set of structures, which are fundamentally different in type to those that were already processed. The way in which these structures differ to the original set lies in the linguistic aspects, rather than the informative. In fact, the new Templates capture the same set of information as those already existing. The structures have a different type of relationship with the data element in the program, which is facilitated by their 'translation' into those already recognised. Although the new Templates are small in number, they have laid the foundations for including many more types of language structure. The program extension includes, therefore, some form of 'interpretation' of the incoming structures that features an extra level of complexity.

The application includes two new packages, as well as new classes within its default package. 15 new classes have been added, as well as 4 test classes, and the existing classes feature considerable extension also. 5 new input files are identified, and several test output files. Also, a new domain/ schema has been developed to test and demonstrate the results of the Project.

The general conclusion reached, which is perhaps inevitable when dealing with natural language, is that there is greater complexity in the target messages than was anticipated. While those Template structures that have been devised and implemented are successfully integrated within the system, they account for a small fraction of the messages that might be encountered. In reality, the sentences incorporated indicate an update to an attribute having a very particular relation to its entity, and are therefore likely to be applicable only to a small range of the updates possible through varied verb forms. The ability of the extension to travel across domains also seems likely to be somewhat restricted.

However, the very fact that these Templates have been included successfully implies a measure of success. Previously, only terms from the Schema (metadata terms) were used to identify an attribute, whereas now verbs associated with these

terms can be used to arrive at updates also. This comprises a fundamentally different direction within the application, and one that should serve to facilitate future development, in ways that were previously out-with the linguistic scope of the system.

On the whole, the justification for the Pattern Matching approach to extraction is borne out by the extension. As is implied in chapter 7, the standard syntactic analysis of the target sentences conflicts with the algorithms used to deliver successful extraction within this application. That said, consideration of grammatical structures has helped to consider and formalise the structure of the Templates now included, and so it would be wrong to say that the approach is entirely free from grammatical considerations.

The secondary aim within the Project was subject to less well-defined requirements. However, the basic objective, to explore methods for generation of inflected verb forms, has been attained. The mechanism by which this has been demonstrated within the program submitted could certainly utilise a more efficient data structure (e.g. an external database accessible from within the application). However, this would be more sensibly designed in light of the work for which the facility is to be used, i.e. as part of some future extension. In any case, the ability to generate the inflected forms of a word on querying with the base form has been implemented. It must be stressed that the inclusion of the Inflection Repository within the submitted program is provided mainly for demonstration purposes, and is expected to require considerable further work before it could feasibly comprise a fully functioning component within the application; the function of the repository at present is primarily to illustrate the fruits of the inflection generation process.

Aside from further finishing the inclusion of the new structures (extending vowel stripping to verbs, loading verbs at the start of the program run etc), the main area for future development that would be attempted given more time regards the organisation of the new Templates. Devising some more complex and appropriate system whereby Verb Templates are related to elements in the Schema would extend the new functionality further, building on the foundations laid as part of this Project.

Appendix A

Project Proposal

The Project Proposal document, as submitted originally, is attached below.

Introduction

Information Extraction is the process whereby information that is significant to some purpose is derived from pieces of text, and contained in a structured format. In contrast with other language processing activities, it is not generally in the interests of Information Extraction systems to achieve a full syntactic analysis of the text, but rather to focus on the processes required to extract all and only that information that is required for some specified function. Relevant information has therefore to be defined in advance, and is later output in some, also predefined, regular way. IE facilities are often developed with a particular domain in mind, featuring some data repository to hold existing information regarding this domain. The domain data source is acquired during the development phase of a system and is then used for the specific text processing tasks that the system is responsible for. The extent to which IE systems attempt to grammatically parse input text documents varies greatly, as does their generality in application. Some of the many approaches to the IE task are outlined in the below sections, as is an overview of the system at hand, along with critical analyses of the differing perspectives on the design of IE systems.

The system under development is aimed at processing short messages, such as email or text messages. These messages are assumed to conform only to loose, unpredictable structure and to arise within a domain that is known, and within the broader context of the World Wide Web. The system approaches the Information Extraction task by modelling held domain information within a database structure. The repository comprises a database corresponding to metadata that are used to generate example linguistic structures in the form of templates, against which incoming data is pattern-matched. The end goal in this task is to update the data held within the system, as new information is received that is relevant to the domain; this, in turn, results in further possible structures to be generated next time around.

Text processed by the system takes the form of short, often variably structured, messages (email or text messages); for this reason a syntactically lightweight Information Extraction approach is justifiable, given the limited value in attempting rigorous linguistic analysis on language that has not been constructed according to the strict grammatical rules of Standard English. This information-focussed perspective is also beneficial given that the intended context for use of the system is the Web, an environment in which discourse takes place between people from radically varied environments, often in a second language, and where culture-specific references/ linguistic usage can give way to a focus on the exchange of raw information. The aim in this process is therefore to isolate the key informative parts in a message by matching against the most common message constructs that are believed likely to occur.

In terms of extension to the existing system, the aim is to account for variations in subject/ verb structure, particularly regarding nominalisation/ sentences using subject complements as opposed to direct objects and possibly also tense, time permitting. At present, sentences expressing information through the verb ‘to be’ are catered for, e.g.:

```
Otis Redding was the singer of These Arms of Mine.
```

However, often sentences will occur in which the verb encapsulates the bulk of the semantic and domain information, e.g.:

```
Otis Redding sang These Arms of Mine.
```

This constitutes a fundamentally different sentence structure, the relationship between the two being semantic rather than purely syntactic. The intention is therefore to explore the possible uses of Discourse Analysis, semantic and ontological approaches in Information Extraction in order to extend the system to accommodate such formations. A more detailed indication of the intended project follows after a review of the field and then a summary of the present system.

Information Extraction overview

Information Extraction began to comprise a field in its own right in the late 80s/ early 90s. Where Information Retrieval accomplishes the acquisition of documents relevant to some broad information need, Information Extraction selects information from documents that is, in turn, required for some specific need. The type and structure of the information sought by an IE system is, by definition, known in advance. To this end, IE aims to extract only information that is specific to the task, avoiding redundant processing activities such as full syntactic parsing that are generally labour-intensive and difficult to perform to high standards of precision. IE therefore aspires toward levels of efficiency and robustness that make the products of its research programs that are often feasible for commercial application. [6]

The field historically utilised Natural Language Processing (NLP) techniques to a greater or lesser degree depending on the task at hand. However, the most recent IE development has tended to focus on less stringent linguistic processing rules in order to deliver systems that are more robust. The reason for this is that NLP techniques are unable to cope with more than a small extent of actual natural language instances, and because most IE systems are tailored to a specific task and therefore domain. Extensive linguistic analysis is neither necessary nor desirable for the Information Extraction task and so more 'lightweight' approaches have been the focus of much recent research. The IE field is therefore characterised by projects whose application is somewhat specific, although recent directions in research have also aspired to increased portability and automation (such as machine-learning algorithms/ language independent systems).

IE systems additionally produce as an end result output that is variable in structure; often systems are developed as a component for some other application which takes the IE results as an input for further processing. A common scenario is one in which the IE activity results in the population of a database, such as the system currently under development. Several decisions need to be made in designing an IE system, since there are many different approaches to each of a set of relatively distinct processing stages that are combined to form a complete IE architecture. These stages can optionally feature differing approaches from one another (i.e. no one approach needs to be adhered to for the entire system). Decisions must therefore be informed by the overall purpose of any one IE system, and often feature certain 'trade-offs' in terms, for instance, of subtlety versus robustness or manual/ hand-crafting of elements versus automation; what is appropriate will likely vary greatly according to the context of any one project. [10]

Message Understanding Conferences

The Message Understanding Conferences have been largely responsible for shaping the direction of IE development. Sponsored by the US Defence Agency DARPA, the MUC conferences brought together projects having the same overall IE aims. The conferences provided the means to quantify and evaluate the success of participating systems, as well as serving as an impetus for advancement of the field and defining specific goals at which projects were able to direct their focus. Although having undoubtedly driven the field of IE forward, the MUC conferences have been criticised for encouraging considerable convergence in approaching the IE task. However, the benefit of the MUC process is that it produced (lots of) end results that were comparatively measurable and therefore ultimately of huge benefit to the progress of the discipline. Many systems developed as part of the MUC conferences have gone on to produce commercial applications. [5, 6]

Participating in the MUC conferences involved developing a system that would perform specific IE processes on a given set of texts issued immediately prior to the conference. Projects were able to base the design of their systems on training texts issued further in advance of the conference, indicating the type and domain of the unseen texts whose processing their applications would be tested with. The testing process required that functionality not be altered to improve accommodation, making the resulting analysis a fair reflection of performance across the board. This testing process was also developed formally as the conferences progressed, in some areas comprising automated assessment. Later conferences also introduced further specifications in terms of the format of the end results of processing, again making success easier to measure and the resulting applications closer to deliverable.

Message processing in a MUC type system results in the production of filled templates containing the required information gleaned from the messages (and formatted appropriately via the template structure). Template structures featuring increased complexity were introduced as the conferences progressed, incorporating hierarchical constructs such as template elements containing pointers to other templates. This increased complexity made possible far more diverse representations of the acquired information and thereby more authentic modelling of the subtleties of meaning implied by linguistic instances in reality.

In terms of the systematic evaluations developed throughout the MUC period, performance was measured according to defined criteria that were formally identified in advance. Recall and Precision were both standards adopted from the Information Retrieval field and given particular interpretations within the context of IE. Recall quantified the amount of information that could be said to have been ‘successfully’ retrieved from the text. Of this, the amount of accurate information extracted was measured to a factor of precision. Shades of precision for these types of functionality were also described: apart from information that is simply correct or incorrect, partial correctness, missing and superfluous information were also accounted for. Given the complexity of the task at hand, these variations within a range for indicating success are extremely useful, particularly as the field has developed with applications having different priorities and consequent definitions of success. In other words, there is no one perfect IE system –a successful one will be suited to the aims of a particular project, although during the MUC process success was necessarily strictly and unambiguously set out from the start.

MUC 6 and 7 (the final conference in 1998) saw the identification of specialised subtasks of which the IE undertaking is composed. These subtasks formed part of what were additionally laid out as goals for the conference itself, further refining the expectations for involvement. Five main tasks were identified: Named Entity Recognition, Coreference Resolution, Template Element Construction, Template Relation Construction, and Scenario Template Production.

Named Entity Recognition is the activity whereby names of known entities whose existence is evident within the text are identified. Entities can generally be thought of as objects (concrete or abstract) such as people, organisations or places. Finding and reporting entity type instances that are indicated by the text is the end goal in Named Entity Recognition. Initially, discovered entity information was translated into marked-up SGML documents.

Coreference Resolution was one linguistic aspect whose interpretation was also expected at this stage. Coreference occurs when more than one lexical item refers to the same entity. A common example is anaphora, for example, the word 'it' in: "I heard the song and I really liked it". Coreference occurs regularly in natural language and so it is in the interests of IE systems to manage these incidents effectively and thereby preserve the relevance of acquired information by linking together associated segments of data. This information was again described through SGML documents in which the elements had corresponding attributes indicating the relationships between linguistic instances. Other linguistic aspects that systems were expected to account for included sense disambiguation and predicate-argument structure.

Named Entity Recognition and Coreference Resolution also provided the basis for Template Element Production. The recognised need to accommodate varied message types and content led to the development of simpler template structures that were intended to have more universal application, leading to systems that should be more easily targeted at implementation areas (without too much manual adaptation necessary prior to deployment). Template Element production describes then an activity that results in the association of information complementing some recognised entity. This aspect of the IE process arrives at data whose generic format is essentially suitable for further use e.g. to populate a database.

Template Relation Production provides further associations between elements in the text, this time outlining potential relationships between the Template Elements already identified. This refining of the notion of relations between parts of the information extracted, through defining expected patterns, gives rise to more accurate framing of the data acquired.

Scenario Template Extraction facilitates IE tailored to specific applications. Entities and Relations already found in the text can here be broadly thought of as components in some set of possible situations or scenarios envisioned. These scenarios are best produced according to the particular needs of the application since the task is one of the more difficult/ labour intensive and domain-dependent.

Notions of standardisation continued to form part of the MUC outlook, creating systems whose worth could be evaluated. Further automation and the desire to perform to high standards of accuracy and speed of potential deployment were also encouraged for the IE discipline.

IE Architecture

Approaches to IE system architecture are varied but a generic structure that can be said loosely to have universal application has prevailed within the field. [7, 8, 11]

There are a number of basic stages that generally apply across any IE system, although the methods used within each may vary. Typically an IE system will conform to some cascading series of activities, with each stage informing the next. Firstly, any incoming text document has to be tokenised in some way, splitting it into sentences or clauses or whichever basic linguistic unit is appropriate to the task. The complexity of this phase naturally depends on the structures required to be isolated for the next stage of processing, and of course the format of the incoming language.

Typically the tokenisation stage is followed by some sort of tagging phase, during which segments of the units passed are marked-up, again appropriately to the task. This aspect is again dependent on the success of the training period, since the ability of the system to tag the text will relate directly to its past experience with similar texts. This activity is also far more reliably performed when a system is aimed at a particular domain because, the broader a linguistic repertoire catered for by a system, the more scope for ambiguity and subtlety that is difficult to resolve; generally there is a higher probability for success when a system is operating within a domain whose language it is familiar with to the exclusion of other domains. In this sense, it's better for the application's 'knowledge' or 'understanding' to be limited. The post-condition for this stage, as with the rest, is that elements in the text have been identified in a way that is useful to the next stage of processing.

As mentioned above, IE systems involve syntactic analysis to varying degrees of depth suited to the individual task. Only excerpts that are important to the extraction task are therefore subject to analysis; these are generally those parts of speech that are easier to identify and translate into some generic data structure (relating to the objects and relationships the application is interested in rather than the numerous shades of semantic expression). The notion of performing more extensive grammatical parsing was compromised fairly early in the life cycle of IE given the difficulty and expense of the task as well as the simple fact that it is often at best unhelpful and at worst actually a hindrance (and therefore inappropriate) to the success of the IE activity. Additionally, full linguistic parsing that accounts for real instances of language has not yet actually been achieved. Thus, this 'shallow' parsing has characterised most recent prominent IE applications.

At this point, it becomes necessary to resolve domain related issues in the text, such as Coreference. As always, the range and extent to which such linguistic phenomena are handled will be a matter for system design. Coreference arises through various structures in a language, in English examples feature referential pronouns and anaphora. The task can be particularly problematic given that a full syntactic analysis has not been carried out on the input text, and there is often the need to 'look back' at the information in the preceding text in resolving these issues as they are encountered. Some systems, generally those employing the greater levels of syntactic parsing, also attempt to deal with partially parsed excerpts by merging them with reference to contextual information at this stage.

Determining which information is to be extracted impacts not only on the extraction element of an IE system but on its output. Typically, a template structure will be designed in advance that will dictate the type of information that is to be

sought. The template structures to be output will often then be subject to further processing, depending on the context of the application. This task inevitably involves some predefined conceptual categorisation of the 'entities' and their relationships expected within the anticipated texts, that is translated into some sensible structure whose format may also be dependent on what the information is being extracted for. Often, this post-processing will take the form of populating some data repository.

As indicated above, these tasks can be approached in a number of different ways, with the focus always on the specific interests of any one application.

The success of systems depends on a number of factors such as the complexity of input text and indeed the required output, the domain/ how appropriately the structures have been chosen to accommodate the extracted information, and the quality (and naturally the quantity) of training data.

Systems employing hand-crafted structures tend to be labour-intensive but more robust, resulting in better performance, whereas learning systems involve less in the way of hands-on work in the early stages but are normally less reliable. Success is also dependent on other contributing factors when the systems are deployed such as input texts, domains, data structures etc. Hand-crafted systems also make for difficult adaptation especially if changes to requirements are identified some way into the development process. More obviously, they also depend to a considerable degree on the people who develop them, who may include domain experts. Learning systems depend for their success upon the training data, whose acquisition is not necessarily a trivial task. However, learning systems are more amenable to changes. Which type of design approach is appropriate also depends on the availability of training texts and other linguistic resources, as well as performance requirements.

IE as a field is all about making appropriate choices, appropriate not only in the sense that the composite parts complement one another but also appropriate to the task i.e. the information need (the type and structure of desired information). Success can be difficult to measure, not least because interpretation of a piece of language can be contentious even when the interpretation is performed by humans! Domain dependence often results from the need to tailor the methods whereby extraction is achieved, and because training texts are chosen that relate to the texts whose processing the eventual application is to be targeted at.

The Current System

The system under development accords to the more 'lightweight' IE approach, extracting data from short messages with the aim of maintaining a repository of domain information. Metadata corresponding to the database model are used to generate linguistic structures, against which incoming expressions are pattern matched; any new information identified is then translated into update statements. The system approaches the task at hand from the viewpoint that incoming messages will conform to simple although loose/ unpredictable structures and most probably contain terms that are recognisably from the domain. It is assumed that messages will not exemplify Standard English either in terms of syntax or spelling. [1, 2, 3, 4]

The system is designed for use as a composite part of a larger application for the construction of information bearing web pages for collaboratively developed sites. This generic design envisages a site where visitors submit contributions either via a form or through short messages, which are then processed in order to update the existing data repository. The facility is particularly useful in this context assuming that one of the aims of such a site would be to encourage the contributions of visitors (and that these may benefit from the more flexible nature of a freely formed text message, hopefully resulting in more information being submitted).

Initially, lexical items were categorised according to a three-way distinction that viewed words as either domain indicators, data values or providers of syntactic structure. However, recently this categorisation has been revised in favour of a more flexible approach that aims to account for a greater amount of incoming messages, as will be reflected in the current project. (One of the reasons for the project is that the system as it stands identifies the 'information category' of a message typically from noun phrases, whereas many common sentences will indicate this through the verb, as described in the below section.) The system detects terms from the metadata and performs pattern-matching techniques as outlined below.

Patterns to be matched against the incoming expressions take the form of template structures having placeholders where data values or words recognisable as domain terms might occur. If a sentence matches the structure, terms occupying the placeholder positions are deduced to be data that can then potentially form the basis for the update statements (unless the data is redundant i.e. is already there). One of the main benefits to the pattern-matching approach is that complex structures can be accounted for without the need for deep linguistic understanding.

The system reflects the recent focus on ontology-based approaches to IE, using a relational database structure to model an ontology for the relevant domain. The ontology describes the domain terms and their relationships, within the context being modelled, in a way that should be a sensible reflection of the meaningful relationships that the words themselves share in the language. In order to deliver the required application, the data structure has therefore been modelled carefully, and has resulted in a 'high-level' abstract structure. This database structure also provides the means to hold different examples of expressing the same information, to date mainly comprising set structures whose various possible forms of expression are realised by substituting value terms for their synonyms. Anything that seems to be unrelated to the domain is simply ignored by the process.

In the intended context of use for the system, this lightweight approach is justifiable given both the restricted set of messages that are likely to occur and the fact

that we know in advance the type of information that we are looking for. There are likely to be a finite set of structures in which we are interested, which removes both the need and benefit of full syntactic analysis. Also, part of the reason for a full syntactic analysis is often the need to resolve contextual ambiguity, which is not likely to be an issue for this application; the messages will occur within a context in which much of this information is known. These structures in which the application is interested are also less likely to change over time in the same way as some other forms of linguistic dialogue in which IE systems may be interested. The system is therefore given the terms and patterns at runtime, and it then compares these to the input text. It should be noted that, although the program does not engage in extensive parsing according to complex grammatical rules, the pattern-matching stage essentially performs a type of parse, but one with a simple defined syntax reflecting the ontology.

Problems that the system is necessarily required to resolve include alternative spellings (proper names are likely to feature often in the incoming texts), conflicting information and numerous structures describing the same information. The last of these is the focus of the current project, as described in the below section.

The nature of the data model will ultimately determine the success of the system in processing any set of statements, it must therefore correspond conceptually to the human model of understanding for the discourse information (if sensible information is to be extracted accurately). An abstracted description of the domain information must form the foundation for the data model, in some way that is realisable through actual data structures. The data model chosen is entity-based since this more realistically matches the postulated human perceptual model. This information framed according to human understanding therefore has to be translated into a form suitable for representation in a data repository classified by 'keys'.

The model in this case features entity types together with their corresponding properties, represented within a repository schema capable of accommodating multimedia data. Conventional relational database primary keys operate and are classed as *Dkeys*, while additional keys have been devised in order to perform the shift between natural language and database entity recognition. These functional *Hkeys* are present to indicate 'humanly intelligible keys', representing a language fragment that corresponds to a database key but in natural language terms, i.e. identifies one particular entity at the human level of discourse. Instances of the *Hkeys* are likely to occur within the text as explicitly identified values indicating an entity: 'named values'. This allows entities to be located within the repository when the input data mentions explicitly only the *Hkey* and not the *Dkey*, which may in fact be a value known only within the database (i.e. one that is artificially generated rather than representing some real world term). The *DKey* may also be the *HKey* for an entity but the facility for them to be different values helps to overcome the barrier in representing the discourse in this, necessarily contrived, way. The system resolves the *Dkey* for an entity given the *Hkey* as input within a message -not necessarily a trivial task, since the *Hkey* may not be unique to one entity and potentially relates to more than one database record.

Presently the data model is realised in a text file but the design allows for more varied implementation e.g. a relational database system/ XML documents.

Update and Insert commands to the data repository are expressed through a tailored query language having the following syntax:

update <Entity> set <Property> = <<Value>>
insert into <EntityType>(<property list>) values (<value list>)

These commands are the end result of the information extraction process in the system. Having established the entity type along with one of its properties and a corresponding value, an update statement is generated. When a new entity is encountered, the entity type and *Dkey*/ *Hkey* value must have been extracted; properties having no value indicated in the text are set to null in preparation for later updates, either from the same message or on other extraction occasions.

In architectural terms, the program functions first by generating sets of patterns that the incoming expressions are potentially expected to match, then generates query statements according to the patterns matched and the values contained in the matching messages. The entity whose data is to be entered/ updated may be explicitly referred to in the sentence at hand or may have been mentioned earlier in the text and retained or 'remembered' by the program. Pattern templates' placeholder elements are systematically replaced by any appropriate data found, resulting in fully or partially filled expressions (the structures generated with data values inserted in placeholder positions).

Templates are grouped into 'Template Types', accounting for different combinations of data elements. These types each correspond to an update statement appropriate to its particular arrangement of the information. It is naturally hoped that the patterns generated will exhaust the bulk of combinations that might occur. The program also allows for manual additions to those patterns automatically produced. This means that values for entities and properties can be instantiated by any of the set of synonyms that would sensibly express the same information. These synonyms are identified either by querying WordNet, through the use of vowel stripping or by manual input via the user interface, through which the synonyms generated can also be edited in the case that inappropriate examples have been selected.

Messages are therefore processed by comparing each sentence in turn against the set of patterns. A sentence is considered a candidate to be the source of an update if it corresponds to one of the patterns (and by implication features values at the placeholder positions). Updates are generated according to the template that has been matched and may also take into consideration information from the context to complete the update/ insert statement (see below).

As mentioned, patterns generated relate to the schema of the corresponding data repository. Whereas the patterns represent numerous possible realisations of information, and so different templates filled can result in the construction of the same update/ insert statements, giving rise to the different template types (each type corresponding to query commands of the same structure). Those template terms that are not occupied by placeholders are classed as 'constant terms', while placeholders exist for metadata terms and for data values. The placeholder names are accompanied by a number to accommodate sentences having more than one value for the type. Placeholder types are defined for: *entityType*, property, *ppronoun* (possessive pronoun), *spronoun* (subject pronoun), *opronoun* (object pronoun) and *qpronoun* (object possessive pronoun) for metadata values and: *Hkey* and *value* for the actual data indicating an Hkey property for an entity and the value for a property respectively.

To pattern match a message, each sentence is compared systematically with each generated pattern. First, the length of the sentence (the number of words) is compared to the length of the pattern; a sentence shorter than the pattern cannot be a match and so is disqualified immediately. Working through the pattern and sentence a term at a time in parallel, the sentence is examined by locating the non-placeholder terms and checking to see whether placeholder positions (relative to the non-placeholder terms) have values stated. The templates are also checked against in a predefined order, which is essential to preserve the relative accuracy of the updates.

The template types that correspond to the commands ultimately produced are parameterised updates, with the placeholder positions holding the parameter slots. Template types must therefore provide the information indicating the entity, property and value concerned. The parameter indicating the entity can come from the most recent entity identified (either explicitly mentioned or deduced through pattern matching) in the text, the entity corresponding to an *Hkey* mentioned in the sentence, or one implied by a possessive pronoun in the sentence. The property parameter is indicated by the property placeholder position in the matching stage. Value parameters similarly come from the value placeholders, but if the property is an entity type, the value will be an *Hkey* translated into a *Dkey* for the update. Update values can also be deduced through the use of subject pronouns or null for negative statements.

When the program discovers a new entity (i.e. one that is not already listed in the database), it must be able to create this in the repository together with any information present regarding the entity (i.e. values for any of its properties). The indication of a new entity might be an overt statement indicating its existence or it might be inferred during discourse about some other entity. The system must therefore be able to deduce the existence of such entities and to ascertain any values that can be attributed to them. When a new entity is explicitly referred to, the entity will occur in the placeholder position for an *Hkey* (or possibly also *Dkey*), whereas an implied entity will be realised in the text as the value of some property. The system, having established that the mentioned entity is not already present in the repository, has the information necessary to insert the data as a new record, setting any values not known to null for the moment. All that is necessary to insert a new entity is the entity type and the *HKey*, the *DKey* being defined if it is not the same as the *HKey*. In the event that further information regarding the new entity follows, this is simply used to generate an update statement as normal, now that the entity exists within the repository.

The fact that context is generally vital for the understanding of an utterance requires the system to provide some artificial ‘memory’ of the present discourse. This construct needs to be available throughout the extraction process if the program is to resolve any sentence featuring reference to something out-with itself. In order to interpret pronouns, it must have access to previously mentioned entities having the same gender. Definite reference requires that an explicit reference has already been made to an entity of the corresponding type (the type mentioned in the definite reference). Implicit references (i.e. not featuring a direct indication of what is being talked about) are resolved by looking at most recently mentioned entities. To make this processing possible, values are held throughout the IE processing task relating to: the most recently mentioned entity; most recently mentioned entities of each gender; most recent entity whose gender is unknown; most recently mentioned entities of each entity type; most recently mentioned entity type. Pronouns are logically characterised

as follows: possessive and object possessive pronouns refer to entities; subject and object pronouns refer to entity values.

The wider context in which a message occurs must also be managed. In particular, when the IE component is called on to do its work, it will require certain information in order to get started. This starting point must be constructed to provide all the contextual information necessary to the extraction process. The process at each step is therefore conditional on a number of factors: explicit entity references can be readily identified, as can directly stated values; entities referred to by means of possessive/ subject/ object pronouns can be resolved using the 'remembered' contextual information; entities referred to by type name only are identified as the entity held for that particular type and implicitly referred to entities are established using the 'most recent' information. On completion of processing for a sentence, the environment must then be prepared appropriately for the next, by updating the contextual information resource.

Synonym management is achieved in a variety of ways within the system, coping with alternative names for the same entity type or property. Sentences carrying the same information but having different synonymous terms are intended therefore to have the same overall effect. The facility also features an interface for managing synonyms within the system, making it possible to use such sources as WordNet, manual input or vowel stripping. Manual removal of inappropriate synonyms generated is supported, essential when dealing with polysemous words, or with genre-specific usage of a word.

The prototype application presently supports the IE process described above with interfaces facilitating manual input and user management for various aspects of the task. Input files to the application include the message, templates, context information and the repository itself. Firstly, the message is loaded, then the template types and templates (in that order), then the schema followed by the corresponding data, then the context, synonyms and patterns. The context structure relates to the already imported schema, and the synonyms to the context, which dictates what terms to generate synonyms for. Patterns are generated from the templates and synonyms already present. The user interface supports display of the message, template types, templates, context, schema, data, patterns, updates and synonyms, in addition to feedback messages/ information that is available as the program runs. The information is viewable in various ways selectable by the user, whose confirmation is also required before update/ insert commands can be issued.

The system has recently been extended in its accommodation of gender and temporal variation. It is also aimed at further resolving negation, uncertainty and inference.

Although the system currently models a percentage of the possible expressions that it may be interested in, it is likely that it will ultimately only be interested in a small number of structures. The data repository is a way of modelling descriptions of entities of interest to a domain, and so it is required to process only the forms of describing these that are likely to occur in short messages.

The Project

As indicated above, the main problem at hand is to extend the system to accommodate messages in which the predicate takes some form other than the verb 'to be'. Specifically, the aim is to handle statements framed in Subject-Verb-Object terms as opposed to those currently processed, which broadly correspond to the Subject-Verb-Complement(-Adverbial) structure.

For example in:

Otis Redding was the singer of These Arms of Mine
(subject) (verb) (complement) (adverbial)

domain term and entity type would be identified from the subject-complement 'the singer' via the verb 'to be', with the song title noun phrase identifying the property and value through an adverbial element.

However, many messages are likely to take the form:

Otis Redding sang These Arms of Mine
(subject) (verb) (object)

where entity type and domain term must be identified through the predicate, the intransitive verb 'to sing' in this case; the other terms present are either named entities or property values depending on how the sentence is analysed.

In order to handle these statements, it will therefore be necessary to extend the selection of patterns generated from the data repository, against which the input statements must be successfully matched. To accomplish this with the existing database schema, additional templates must be created to represent these structures but they must be capable of being generated from the data repository as it stands. A number of approaches will be explored with the aim of achieving this.

Given that the relationship between the types of sentence presently catered for and those aimed at is largely a semantic one, it is hoped that WordNet will successfully provide the means to incorporate the new sentences. The WordNet system describes the lexicographical relation between e.g. the verb 'to sing' and the noun 'singer' purely in terms of the words having common derivation. For this reason, substantial exploration of WordNet structure and functionality will be required in order to establish the usefulness and efficiency of using this feature of the database with the target application in mind (see below section on WordNet). [25]

Given that there is a substantial degree of uncertainty with regard to approaches that are likely to produce positive results for the project, other available facilities will be researched. For example, there are many applications existing online that build additional functionality on top of the WordNet architecture. One such system is HowNet, a resource that attempts to incorporate the concept of meaning into the lexical database. Should the WordNet system be found not to assist the system in the required way, such systems will also be explored. [27, 28]

Additionally, linguistic approaches such as Discourse Analysis and possibly also Transformational Grammar may offer helpful theories. Discourse Analysis is the study of natural language as used to communicate in different contexts. It attempts to account for the manner in which communication takes place through spoken or written language. The concerns of Discourse Analysis are therefore semantic,

grammatical and sometimes sociological/ psychological (and inevitably philosophical). Given that the aim of the system is arguably to simulate a kind of one-way communication process between the contributor of a message and the data resource (to replicate the communication process as it exists for humans in such a way that is appropriate to the data model), this field may inform the development of the system, although the specifics are undefined as yet. Discourse Representation Theory, a semantic theory that has been used in other IE systems, will also be explored.

Given that the system presently employs an ontological model of the domain information in the form of a database, it may also be of benefit to explore the various ways in which ontologies have been used in other IE systems such as that developed by KMi at the Open University. However, the fact that most of these systems are likely to engage in more extensive syntactic parsing processes, their approaches will likely require interpretation in order to be of use, if indeed their practices are transferable at all.

Although the manner in which it will (hopefully) be achieved remains to be decided, in general, the aim of the project will be to extend the templates generated according to the structure indicated above. This will mean that additional patterns will be generated, in the same way as those currently generated to account for synonymous expressions of the same information, since the sentences aimed at are also ways of describing the same data as those already accommodated. As with the generation of synonyms, it may also be useful to incorporate some user interaction with the system in order to validate the alternate sentence forms before they are actually generated, since erroneous forms are likely to appear. Naturally the aim will be to produce as few inappropriate patterns as possible, but the difficulty of this task is yet to be discovered.

Time permitting, the project will also aim to account for different verb tenses, a linguistic aspect that should be informed by exploration of the above problem, whose focus is also on the predicate part of the sentence. Which approaches turn out to be fruitful will depend both on their investigation, with the application in mind, and also on increased exploration of the functionality of the existing system. It should be stated at this point that all of the above techniques may fail to result in any real extension to the program functionality, but at the very least the project should furnish the future of the application with a clearer sense of legitimate directions in which to move.

WordNet

WordNet online lexical database assists many IE systems, including that under development. The database models lexical entries such as nouns, verbs, adjectives and adverbs in groups linked by semantic relations. The groups are defined as sets of synonyms, with members of different groups related through, not only synonymy, but also antonymy, hyponymy/ hypernymy, meronymy/ holonymy, troponymy and entailment, as well as derivational relations. The sets of synonyms, 'synsets' in WordNet terminology, form the basic relation on which the WordNet program is based. [25]

Given any lexical item listed in WordNet, it is possible to identify its synonyms (words having the same meaning or whose interchanging doesn't affect the overall meaning of the containing clause in its particular context), ordered by frequency or semantic similarity. Additionally for nouns it is possible to identify hypernyms/ hyponyms (super/subordinates) and also words having the same hypernyms, meronyms/ holonyms (part to whole and vice-versa) and for verbs: troponyms (words denoting the manner in which the action associated with the verb is carried out). Nouns, adverbs and adjectives can also be linked with their antonyms (opposites). Where applicable, adverbs can also be related to an adjective from which they are derived. Indications of domain and frequency for the use of each item can also be found, as can other words that share a derivational link –an aspect hoped to be useful for the present application.

The WordNet system is constructed from the perspective of assisting programs that attempt to handle natural language input and is therefore furnished with many useful facilities that will be explored as part of the current project.

The relation between the verb 'to sing' and the noun 'singer' can be found via the 'derivationally related forms' option for either item, although this feature naturally results in many other terms, and so the feasibility of using the function will be subject to experimentation.

Work Plan

The project is intended to be carried out according to the following loosely defined schedule:

An initial period of design informed by experimentation with the current system, increased familiarity with which should inform the production of a set of organised priorities for the project.

Timescale: roughly 2-3 weeks (end of May to mid-end June including 1 week holiday).

Gathering of training data for the extension together with the establishing of initial implementation details will then lead into a period of implementation punctuated by regular reviewing and subsequent changes in direction as appropriate. Decisions will be made at an early point in this phase as to what form the extension will comprise i.e. whether accommodating the new sentence structures will require augmentation of the existing data model or whether it can be achieved supplementary to it.

Iterative implementation period including design/ definition of according recognition algorithms and then testing/ evaluation: 6-8 weeks (end June to mid-end August).

Write-up: 2-4 weeks (mid August to hand-in in early September).

Naturally these stages will likely encroach on one another.

Literature

The Present System:

[1] Extracting Data From Personal Text Messages
Richard Cooper and Sajjad Ali
Department of Computing Science, University of Glasgow
-extensive overview of the current system including design and implementation details

[2] Extracting Information from Short Messages
Richard Cooper, Sajjid Ali, Chenlan Bi
Department of Computing Science, University of Glasgow
-introductory overview of the system approach

[3] A Strategy For Using More of the Language in Extracting Information From Short Messages
Richard Cooper
Department of Computing Science, University of Glasgow
-update on recent developments for the system

[4] Content Management For Declarative Web Site Design
Richard Cooper, Michael Davidson
Department of Computing Science, University of Glasgow
-overview of application context for the IE system under development

Information Extraction
-generally:

[5] Message Understanding Conference –6: A Brief History
Ralph Grishman and Beth Sundheim
Proceedings of the 16th International Conference on Computational Linguistics, June 1996 p466-471
-description of MUC-6 process and evaluations.

[6] Information Extraction: Beyond Document Retrieval
R. Gaizauskas and Y. Wilks *Journal of Documentation* 1997
-extensive overview of IE technology history/ context, development and application leading into case study of LasIE system; includes detailed summary of MUC history.

[7] Information Extraction, Automatic
Hamish Cunningham
Encyclopaedia of Language and Linguistics
Elsevier, 2nd Edition 2005
-introduction to IE and comparative review of deployed applications

[8] Information Extraction – A User Guide
Hamish Cunningham
Natural Language Processing Group, University of Sheffield

Research Memo CS-99-07 April 1999
-overview of general IE systems architecture/ processing phases.

[9] Information Extraction: Techniques and Challenges
Ralph Grishman
Lecture Notes in Computer Science, Vol 1299
Springer-Verlag 1997
-introduction to the common broad techniques and issues for IE systems.

[10] Introduction to Information Extraction Technology
Douglas E. Appelt, David J. Israel
http://www.ai.sri.com/~appelt/ie-tutorial/ -checked 20.04.07
-tutorial series providing detailed introduction and evaluation of alternative approaches to each stage in the IE task.

[11] The Generic Information Extraction System
Jerry R Hobbs
Artificial Intelligence Center, SRI International
Proceedings of the Fifth Message Understanding Conference
Morgan Kaufman 1993 p87-91
-introduction to the common components/ general architecture in information extraction systems.

-IE with reference to specific systems/ approaches:

[12] Ontology-Driven Discourse Analysis in GenIE
Philipp Cimiano
Data & Knowledge Engineering, Elsevier -Vol 55 Issue 1 2005
-overview of discourse resolution component of GenIE IE system; represents context through a domain-specific ontology model; Discourse Representation Theory informs analysis of the context.

[13] On-Demand Information Extraction
Satoshi Sekine Computer Science Department, New York University
http://nlp.cs.nyu.edu/sekine/papers/coling06.pdf -checked 20.04.07
-description of pattern-matching IE system whose patterns are generated in response to user query by employing paraphrasing techniques on the user input.

[14] Empirical Methods in Information Extraction
Claire Cardie, Department of Computer Science, Cornell University
AI Magazine 1997
-overview of IE systems with particular reference to machine learning techniques.

[15] Lightweight Natural Language Database Interfaces
In-Su Kang, Seung-Hoon Na, Jong-Hyeok Lee, Gijoo Yang
Natural Language Processing and Information Systems, Springer Berlin 2004
-description of linguistically lightweight approach to translation aspect of database interfaces that allow users to issue natural language questions.

[16] Template-Driven Information Extraction for Populating Ontologies

Maria Vargas-Vera, John Domingue, Yannis Kalfoglou, Enrico Motta, Simon Buckingham Shum
Knowledge Media Institute, The Open University
http://users.ecs.soton.ac.uk/~yk1/kmi-tr-105.pdf -checked 20.04.07
-overview of integration of ontology with IE system (ontology functioning both as an aid to the IE process and being populated with the IE results), uses lightweight linguistic processing techniques.

[17] Robust parsing based on discourse information: completing partial parses of ill-formed sentences on the basis of discourse information
Tetsuya Nasukawa
IBM Research, Tokyo Research Laboratory, Japan
Annual Meeting of the ACL, Proceedings of the 33rd annual meeting on Association for Computational Linguistics 1995
-approach to NLP using discourse information to complement instances of partial parsing.

[18] FASTUS: Extracting Information From Natural Language Texts
Jerry Hobbs, Douglas Appelt, John Bear, David Israel, Megumi Kameyama, Mark Stickel, Mabry Tyson
Artificial Intelligence Center, SRI International
http://www.ai.sri.com/~appelt/fastus.html -checked 20.04.07
-description of Finite State Automaton Text Understanding System developed as part of the MUC conferences (originally intended to complement TACITUS); cascaded linguistic processing stages together with pattern matching techniques produce iterative structures, end result is structured data suitable for populating a database or for further application.

[19] Ontology-Driven Question Answering in Aqualog
Vanessa Lopez, Enrico Motta
Knowledge Media Institute, The Open University –Tech Report kmi-04-5 April 2004
-description of question-answering approach in an IE system using an ontology, GATE NLP techniques and WordNet for use in the Semantic Web context.

[20] Automatic Template Creation For Information Extraction, an Overview
Robin Collier
Technical Report CS-96-07, University of Sheffield 1996
-description of approach to automating the template creation/ definition in an IE system (as opposed to hand-crafted techniques for producing template structures).

[21] Automatic Analysis of Descriptive Texts
James Cowie, University of Strathclyde
Proceedings of the ACL Conference on Applied Natural Language Processing 1983 p117-123
-description of IE system employing lightweight linguistic processing of descriptive texts to inform a database system, implemented in PROLOG.

[22] Automatic Extraction of Facts From Press Releases to Generate News Stories
P.M Andersen, P.J Hayes, A.K Huettner, L.M Schmandt, I.B Nirenburg, S.P Weinstein

Proceedings of the Third Conference on Applied Natural Language Processing 1992 p170-177
-description of the JASPER system (created by Carnegie group for Reuters); used a template driven partial parsing technique, pattern matcher uses tailor-made pattern language together with a set of extraction rules.

Associated Topics:

[23] Tapping the Power of Text Mining
Weiguo Fan, Linda Wallace, Stephanie Rich, Zhongju Zhang
Communications of the ACM; September 2006/ Vol. 49 No. 9
-introduction to concepts and techniques in Text Mining, a generic context in which IE systems can exist; text mining is similar to Data Mining but is more broadly aimed at handling different and more or less structured data formats.

[24] Text Mining: Finding Nuggets in Mountains of Textual Data
Jochen Dorre, Peter Gerstl, Roland Seiffert
SWSD, IBM Germany
http://maya.cs.depaul.edu/~classes/ect584/papers/dorre.pdf -checked 20.04.07
-overview of Text Mining principles, techniques and appplications.

[25] WordNet: A Lexical Database for English
George A Miller *Communications of the ACM*, November 1995
http://wordnet.princeton.edu/ -checked 20.04.07
-introduction to WordNet lexical database.

[26] NLP: An Information Extraction Perspective
Ralph Grishman
Department of Computer Science, New York University
http://nlp.cs.nyu.edu/publication/papers/RANLP05-GrishmanInvited.pdf -checked 20.04.07
-brief overview of pertinent issues in Natural Language Processing from the point of view of IE.

Online Resources

[27] HowNet semantic application built on top of WordNet
http://www.keenage.com/zhiwang/e_zhiwang.html
Checked 20.04.07

[28] Creative Language System Group –Analogical Thesaurus
(KNOW-BEST project uses HowNet and Wikipedia)
http://afflatus.ucd.ie/
Checked 20.04.07

[29] Information Extraction resource provided by NIST
http://www.itl.nist.gov/iaui/894.02/related_projects/muc/
Checked 20.04.07

[30] Sheffield University GATE (General Architecture for Text Engineering) homepage
http://gate.ac.uk/ie/
Checked 20.04.07

[31] Homepage of the Proteus Project, contains useful IE links
http://nlp.cs.nyu.edu/index.shtml
Checked 20.04.07

Language Textbooks:

[32] Discourse Analysis
Gillian Brown, George Yule
0521284759 Cambridge University Press 1983

[33] A Student's Grammar of the English Language
Sidney Greenbaum, Randolph Quirk
0582059712 Longman 1990

[34] Semantics
John I Saeed
0631200355 Blackwell 1997

[35] Introduction to the Grammar of English
Rodney Huddleston
0521297044 Cambridge University Press 1984

[36] Transformational Grammar
Andrew Radford
0521347505 Cambridge University Press 1988

Appendix B

Templates

The original set of Templates existing within the application are listed below:

7 The <attribute1> is <<value1>>, while the <attribute2> is <<value2>>.
1 The <attribute1> is <<value1>>.
1 <attribute1> : <<value1>>.
8 The <attribute1> of the <entityType1> is <<value1>>.
9 The <attribute1> of <<hkey1>> is not <<value1>>.
2 The <attribute1> of <<hkey1>> is <<value1>>.
2 The <attribute1> of the <<hkey1>> is <<value1>>.
3 <ppronoun1> <attribute1> is <<value1>>.
3 <ppronoun1> <attribute1> was <<value1>>.
4 <spronoun1> is the <attribute1>.
5 <spronoun1> is the <attribute1> of <<hkey1>>.
6 <spronoun1> is <ppronoun1> <attribute1>.
5 <spronoun1> is <attribute1> of <<hkey1>>.
4 <spronoun1> is <attribute1>.
5 <spronoun1> was <attribute1> of <<hkey1>>.
4 <spronoun1> was <attribute1>.
5 <spronoun1> was the <attribute1> of <<hkey1>>..
5 <spronoun1> was also the <attribute1> of <<hkey1>>.
4 <spronoun1> was the <attribute1>.
6 <spronoun1> was <ppronoun1> <attribute1>.

The new set of Verb Templates, listed in the order in which they are presented during Pattern matching is as follows:

6 <spronoun1> <<verb1>> <opronoun1>.
3 <<value1>> <<verb1>> <opronoun1>.
8 <<value1>> <<verb1>> the <entityType1>.
6 <spronoun1> <<passiveVerb1>> <opronoun1>.
5 <<hkey1>> <<passiveVerb1>> <opronoun1>.
3 <spronoun1> <<passiveVerb1>> <<value1>>.
8 the <entityType1> <<passiveVerb1>> <<value1>>.
2 <<hkey1>> <<passiveVerb1>> <<value1>>.
5 <spronoun1> <<verb1>> <<hkey1>>.
5 <spronoun1> also <<verb1>> <<hkey1>>.
2 <<value1>> <<verb1>> <<hkey1>>.

Appendix C

Patterns

The set of Patterns based on the new Verb Templates for the 'library' domain:

he (verb) it
she (verb) it
<author> (verb) it
<author> (verb) the Book
it (passiveVerb) him
it (passiveVerb) her
<Book> (passiveVerb) him
<Book> (passiveVerb) her
it (passiveVerb) <author>
the Book (passiveVerb) <author>
<Book> (passiveVerb) <author>
he (verb) <Book>
she (verb) <Book>
he also (verb) <Book>
she also (verb) <Book>
<author> (verb) <Book>

The set of Patterns based on the new Verb Templates for the ‘song’ domain:

he (verb) it
she (verb) it
<singer> (verb) it
<singer> (verb) the Song
it (passiveVerb) him
it (passiveVerb) her
<Song> (passiveVerb) him
<Song> (passiveVerb) her
it (passiveVerb) <singer>
the Song (passiveVerb) <singer>
<Song> (passiveVerb) <singer>
he (verb) <Song>
she (verb) <Song>
he also (verb) <Song>
she also (verb) <Song>
<singer> (verb) <Song>

Appendix D

Testing Output

Testing output files for several program runs are provided on the submitted CD.

The file entitled TestingOut.txt summarises the run of unit tests.

The directories entitled test1, test2 etc contain the following files:

- PatternTest.txt (summary of Pattern Generation)
- VerbsOut.txt (details of Verb Generation)
- TemplateTest.txt (Templates instantiated)
- SystemTest.txt (overview of entire program run)

The directories also contain the Messages processed as appropriate.

The Testing directory contains all of the above.

Appendix E

Evaluation Results

The following email message was sent to contributors:

Hello

Thanks for agreeing to help with my Masters Project, hopefully this shouldn't take up too much of your time.

What I'm looking for is for you to send me two short messages, each preferably consisting of several sentences, with the aim of providing me information about a subject. One of the messages should be about music, specifically about singers and songs. An example might be:

Love Me Tender was sung by Elvis Presley. He also sang Are You Lonesome Tonight. Otis Redding was the singer of These Arms of Mine.
-and so on

This is just one example of what would be appropriate, but you should phrase your own information in a way that suits you.

The other message should be about books, e.g:

The author of Persuasion is Jane Austen. She also wrote Pride and Prejudice. Douglas Coupland wrote Shampoo Planet. He is also the author of Generation X. Its ISBN was 0123456789. etc

The above qualification applying to this also.

The aim that the system will have on processing these messages is to extract the information within them and update its store of data on the subject, so the messages should be expressing information, rather than opinions about what you're talking about, whether you like it or not etc.

Thanks again

Susan

continued

As in the Testing Appendix (D) output files for several program runs are provided on the submitted CD.

The directories entitled eval1, eval2 etc each contain the evaluation results for a single message. Each directory contains a small text file listing the result in brief, and a sub-directory containing both the original message and the edited version, together with their output files:

- PatternTest.txt (summary of Pattern Generation)
- VerbsOut.txt (details of Verb Generation)
- TemplateTest.txt (Templates instantiated)
- SystemTest.txt (overview of entire program run)

The directories also contain the Messages processed as appropriate.

The Evaluation directory contains these for 10 messages.

Appendix F

Input Files

Library domain schema (created prior to this Project):

```
Library
2
        Book
        4
                ISBN    string  n       y       y       sv
                title   string  n       n       y       sv
                author  Author  y       n       n       mv
                year    string  n       n       n       sv
        Author
        4
                ID      number  n       y       n       sv
                name    string  n       n       y       sv
                dob     string  n       n       n       sv
                gender  gender  n       n       n       sv
```

Song schema (developed for the Evaluation of this Project):

```
Song
2
        Song
        4
                ref     string  n       y       y       sv
                title   string  n       n       y       sv
                singer  Singer  y       n       n       mv
                year    string  n       n       n       sv
        Singer
        4
                ID      number  n       y       n       sv
                name    string  n       n       y       sv
                dob     string  n       n       n       sv
                gender  gender  n       n       n       sv
```

Lexical Categories file:

```
<adjective-all>
<adjective-pertaining>
<adverb-all>
<noun-tops>
<noun-act>
<noun-animal>
<noun-artifact>
<noun-attribute>
<noun-body>
<noun-cognition>
<noun-communication>
<noun-event>
<noun-feeling>
<noun-food>
<noun-group>
<noun-location>
<noun-motive>
<noun-object>
<noun-person>
<noun-phenomenon>
<noun-plant>
<noun-possession>
<noun-process>
<noun-quantity>
<noun-relation>
<noun-shape>
<noun-state>
<noun-substance>
<noun-time>
<verb-body>
<verb-change>
<verb-cognition>
<verb-communication>
<verb-competition>
<verb-consumption>
<verb-contact>
<verb-creation>
<verb-emotion>
<verb-motion>
<verb-perception>
<verb-possession>
<verb-social>
<verb-stative>
<verb-weather>
<adjective-participial>
```

Lexical Descriptions file:

```
all adjective clusters
relational adjectives (pertainyms)
all adverbs
unique beginner for nouns
nouns denoting acts or actions
nouns denoting animals
nouns denoting man-made objects
nouns denoting attributes of people and objects
nouns denoting body parts
nouns denoting cognitive processes and contents
nouns denoting communicative processes and contents
nouns denoting natural events
nouns denoting feelings and emotions
nouns denoting foods and drinks
nouns denoting groupings of people or objects
nouns denoting spatial position
nouns denoting goals
nouns denoting natural objects (not man-made)
nouns denoting people
nouns denoting natural phenomena
nouns denoting plants
nouns denoting possession and transfer of possession
nouns denoting natural processes
nouns denoting quantities and units of measure
nouns denoting relations between people or things or ideas
nouns denoting two and three dimensional shapes
nouns denoting stable states of affairs
nouns denoting substances
nouns denoting time and temporal relations
verbs of grooming, dressing and bodily care
verbs of size, temperature change, intensifying, etc.
verbs of thinking, judging, analyzing, doubting
verbs of telling, asking, ordering, singing
verbs of fighting, athletic activities
verbs of eating and drinking
verbs of touching, hitting, tying, digging
verbs of sewing, baking, painting, performing
verbs of feeling
verbs of walking, flying, swimming
verbs of seeing, hearing, feeling
verbs of buying, selling, owning
verbs of political and social activities and events
verbs of being, having, spatial relations
verbs of raining, snowing, thawing, thundering
participial adjectives
```

Detachment Suffixes file:

```
NOUN s _
NOUN ses s
NOUN xes x
NOUN zes z
NOUN ches ch
NOUN shes sh
NOUN men man
NOUN ies y
VERB ies y
VERB s _
VERB es e
VERB es _
VERB ed e
VERB ed _
VERB ing e
VERB ing _
ADJ er _
ADJ est _
ADJ er e
ADJ est e
```

Files are all included on the submitted CD.

Appendix G

Exploratory Programs

A series of programs were devised to test the functionality of the various resources used within the Project.

The CD contains java files with these programs for information.

- IEwnjwnl.java explored both JWNL and wn.jar functionality.

- IEwndb explored wn.jar

- IEjwi explored MIT's JWI release

The files are contained within the Java Test Programs directory.

Appendix H

WordNet Reference

The Lexicographer File categories:

00 adj.all	all adjective clusters
01 adj.pert	relational adjectives (pertainyms)
02 adv.all	all adverbs
03 noun.Tops	unique beginner for nouns
04 noun.act	nouns denoting acts or actions
05 noun.animal	nouns denoting animals
06 noun.artifact	nouns denoting man-made objects
07 noun.Attribute	nouns denoting Attributes of people and objects
08 noun.body	nouns denoting body parts
09 noun.cognition	nouns denoting cognitive processes and contents
10 noun.communication	nouns denoting communicative processes and contents
11 noun.event	nouns denoting natural events
12 noun.feeling	nouns denoting feelings and emotions
13 noun.food	nouns denoting foods and drinks
14 noun.group	nouns denoting groupings of people or objects
15 noun.location	nouns denoting spatial position
16 noun.motive	nouns denoting goals
17 noun.object	nouns denoting natural objects (not man-made)
18 noun.person	nouns denoting people
19 noun.phenomenon	nouns denoting natural phenomena
20 noun.plant	nouns denoting plants
21 noun.possession	nouns denoting possession and transfer of possession
22 noun.process	nouns denoting natural processes
23 noun.quantity	nouns denoting quantities and units of measure
24 noun.relation	nouns denoting relations between people or things or ideas
25 noun.shape	nouns denoting two and three dimensional shapes
26 noun.state	nouns denoting stable states of affairs
27 noun.substance	nouns denoting substances
28 noun.time	nouns denoting time and temporal relations
29 verb.body	verbs of grooming, dressing and bodily care
30 verb.change	verbs of size, temperature change, intensifying, etc.
31 verb.cognition	verbs of thinking, judging, analyzing, doubting
32 verb.communication	verbs of telling, asking, ordering, singing
33 verb.competition	verbs of fighting, athletic activities
34 verb.consumption	verbs of eating and drinking
35 verb.contact	verbs of touching, hitting, tying, digging
36 verb.creation	verbs of sewing, baking, painting, performing
37 verb.emotion	verbs of feeling
38 verb.motion	verbs of walking, flying, swimming
39 verb.perception	verbs of seeing, hearing, feeling
40 verb.possession	verbs of buying, selling, owning

41	verb.social	verbs of political and social activities and events
42	verb.stative	verbs of being, having, spatial relations
43	verb.weather	verbs of raining, snowing, thawing, thundering
44	adj.ppl	participial adjectives

WordNet 2.1 has been included on the CD.

Please see the documentation within this for further information.

Bibliography

Please see the Project Proposal document in Appendix A for a longer list of useful references.

Brown, G., Yule, G. (1983) *Discourse Analysis* Cambridge University Press, ISBN 0521284759

Chomsky, N. (1965) *Aspects of the Theory of Syntax* MIT Press, ISBN 0262530074

Chomsky, N. (1998) *Language and Problems of Knowledge* MIT Press, ISBN 0262530708

Cooper, C. and Ali, S. (2006) *Extracting Data from Personal Text Messages* University of Glasgow

Cooper, C. and Manson, S. (2007) *Extracting Temporal Information from Short Messages* University of Glasgow

Coulthard, M., Montgomery, M. (1981) *Studies in Discourse Analysis* Routledge, ISBN 041504331x

Cunningham, H. (1999) *Information Extraction –A User Guide* Natural Language Processing Group, University of Sheffield, Research Memo CS-99-07

Gaizauskas, R., Wilks, Y. (1997) *Information Extraction: Beyond Document Retrieval* Journal of Documentation

Miller, G. *WordNet: A Lexical Database for English* Communications of the ACM, November 1995

Saeed, J. (1997) *Semantics* Blackwell, ISBN 0631200355

Sinclair, J., Coulthard, R.M. (1975) *Towards and Analysis of Discourse* Oxford University Press, ISBN 0194360113

Stubbs, M. (1983) *Discourse Analysis: The Sociolinguistic Analysis of Natural Language* Blackwell, ISBN 0631127631

Taylor, J. (1989) *Linguistic Categorization: Prototypes in Linguistic Theory* Oxford, ISBN 0198700121

Web Resources:

WordNet Lexical Database: http://wordnet.princeton.edu/

Five Papers on WordNet: http://wordnet.princeton.edu/5papers.pdf

JWNL WordNet Java API: http://sourceforge.net/projects/jwordnet

Dan Bikel wn.jar: http://www.cis.upenn.edu/~dbikel/software.html#wn

MIT's JWI: http://www.mit.edu/~markaf/projects/wordnet/

WNJN: http://wnjn.sourceforge.net/

AGID word list: http://wordlist.sourceforge.net/