Combining Data Integration and Information Extraction

Dean Williams

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy in the University of London.

Submitted July 2008

To my parents

Combining Data Integration and Information Extraction

Abstract

Improving the ability of computer systems to process text is a significant research challenge. Many applications are based on partially structured databases, where structured data conforming to a schema is combined with free text.

Information is stored as text in these applications because the queries required are not all known in advance – allowing for text is an attempt to capture information that could be relevant in the future but cannot be anticipated when the database schema is being designed. Text is also used due to the limitations of conventional databases, where the schema cannot easily be extended as new entity types and relationships arise in the future.

Information Extraction (IE) is the process of finding instances of pre-defined entity types within text, while Data Integration systems build a virtual global schema from available structured data sources. We argue that combining techniques from IE and data integration is a promising approach for supporting applications that access partially structured data: the virtual global schema and associated metadata can be used to partially configure an IE process, and the information extracted by the IE process can then be integrated into the virtual global database, supporting queries which could not otherwise be answered.

In this thesis we describe the design and implementation of the Experimental System To Extract Structure from Text (ESTEST) that investigates this approach. We

give examples of its use and experimental results from a number of application domains.

Declaration

This thesis is the result of my own work, except where explicitly acknowledged in the text.

Dean Williams, July 2008

Contents

Abstract 3

Declaration 5

Contents 6

List of Figures 10

List of Tables 11

Acknowledgments 12

1. Introduction 13

1.1 Problem Domain 13

1.1.1 Partially Structured Data 14

1.2 Our Approach 14

1.3 Structure of the Thesis 19

2. Databases and Data Integration 20

2.1 Conventional Database Provision for Text 20

2.2 Semi-Structured Data and Data on the Web 22

2.3 Text Mining 23

2.4 Graph Based Data Models 24

2.5 Data Integration 26

2.5.1 Schema Matching 27

2.6 Discussion 28

3. Information Extraction 30

3.1 Natural Language Processing 30

3.2 Information Extraction 33

3.3 The GATE Information Extraction System 37

3.4 Recent Developments in Information Extraction 39

3.5 Discussion 41

4. The AutoMed Data Integration System 43

4.1 Overview of AutoMed 44

4.2 Extending AutoMed: Ontologies 47

4.2.1 RDF.. 48

4.2.2 RDF Schema 50

4.2.3 Other Ontology Languages 52

4.2.4 Representing RDF and RDFS in the HDM 53

4.2.5 Wrapping RDF Data Sources 56

4.3 Extending AutoMed: the HDM Data Store.. 57

4.3.1 Implementation 59

4.3.2 HDM Data Store API & Wrapper..61

4.3.4 Command Parser.. 63

4.4 Discussion 65

5. The Design of the ESTEST System 67

5.1 The ESTEST Approach 68

5.2 Integrate Data Sources 70

5.2.1 Wrapping of Data Sources 71

5.2.2 ESTEST Metadata Repository 74

5.2.3 Initial Global Schema Creation 76

5.3 Semi-Automatically Configure IE 78

5.4 Information Extraction Process 80

5.5 Integrate Extracted Information 80

5.6 Remaining ESTEST Phases 83

5.7 Discussion 84

6. Evaluation of the ESTEST System... 87

6.1 The Road Traffic Accident Domain 87

6.2 An Example of ESTEST in use 89
6.2.1 Initial Configuration is Loaded 91
6.2.2 Integration of Data Sources 91
6.2.3 Querying the Global Schema 101
6.2.4 Configuration of Information Extraction Process 102
6.2.5 Parameters are Loaded 103
6.2.6 Information Extraction Configuration Generated 103
6.2.7 First Information Extraction Step 105
6.2.8 Second Query Step 108
6.2.9 Parameters Loaded to Expand Animal Word Forms 109
6.2.10 Configuration of Information Extraction Process 109
6.2.11 Second Information Extraction Step 111
6.2.12 Final Query Step 111
6.2.13 Limitations of the Example 112
6.3 Evaluation using Road Traffic Accident Data 112
6.3.1 Q1: Which accidents involved red traffic lights? 114
6.3.2 Q2: How many accidents took place 30-50m of a junction? 117
6.3.3 Q3: How many accidents involve drunken pedestrians? 119
6.3.4 Q4: Which accidents were caused by animals? 120
6.3.5 Q5: How many accidents resulted in a collision with a lamppost? 122
6.3.6 Summary of the Results 124
6.4 Discussion 125
7. Extending the ESTEST System 128
7.1 Crime Example 128
7.2 Information Extraction for Schema Matching 130
7.2.1 ESTEST Extensions for IE in Schema Matching 132
7.2.2 Schema Matching in the Crime Example 133
7.2.3 Making Use of WordNet in Schema Matching 136

7.3 Combining Duplicate Detection and Coreference Resolution 139

7.3.1 Detecting Duplicates in Databases 139

7.3.2 Coreference Annotation in NLP 141

7.3.3 Coreference Annotation in GATE........142

7.3.4 Combining the two approaches in ESTEST........143

7.3.4 Duplicate detection in the Crime Example........146

7.4 Text Matching Patterns in the EMR........153

7.5 Automatic Extraction of Values from Text 156

7.6 Discussion 160

8. Conclusions and Future Work........162

8.1 Summary and Principal Achievements 162

8.2 General Applicability of our Approach 167

8.2.1 Other Application domains........167

8.2.2 Using other Data Integration and Information Extraction Software.169

8.3 Future Work 170

Bibliography........174

A. ESTEST Implementation........189

A. ESTEST Output from the RTA Example 193

B. ESTEST Output from the Crime Example........205

C.1 Input Provided to Configure ESTEST 205

C.2 Automatically Created GATE Configuration Files........207

C.3 Output from ESTEST........211

List of Acronyms 234

Glossary 236

List of Figures

Figure 4.1 AutoMed Architecture 47

Figure 4.2 RDF Triple and its Reified Equivalent 49

Figure 4.3 RDF Containers 50

Figure 4.4 The Components of RDF Schema 52

Figure 4.5 Schema of the HDM Data Store. 60

Figure 5.1 ESTEST Phases 69

Figure 5.2 ESTEST Integration Component Architecture 71

Figure 5.3 Schema Network Generated by ESTEST 77

Figure 6.1 AutoMed representation of the AccDB data source. 93

Figure 6.2 The EDM representation of the AccDB data source. 95

Figure 6.3 The AccOnt data source and its EDM representation 97

Figure 6.4 Global Schema. 101

List of Tables

Table 4.1 RDFS Resources..........51

Table 4.2 RDFS Properties..........51

Table 4.3 RDF Constructs and their HDM Representation..........54

Table 4.4 RDF Statements and Containers..........55

Table 4.5 RDFS Constructs and their HDM Representation..........56

Table 6.1 Results of RTA Query Experiments..........124

Acknowledgments

I am especially grateful to my supervisors, Professor Alexandra Poulovassilis and Professor Peter King for their support and guidance.

Thanks are also due to the AutoMed and GATE development teams for producing such useful software, which assisted in my research.

Finally, I would like to thank the family and friends who have encouraged me through this research especially Barbara, Dave, Julia, Tony, Che, Nancy, Sue, Jane, Andrea, Sandra, Steve, my colleagues at work, the staff and students at the School of Computer Science & Information Systems, and my colleagues on the AutoMed development team especially Lucas, Edgar, Hao, Mike, Sasi, Nerissa, Nikos, and Peter. Luckily, I have maintained my usual good humour and unrelenting optimism throughout, so it won't have been too dull for them hearing about the PhD over the last seven years. Thanks also to Mat and Rachel, though really they should thank me, as sharing a college office they were able to benefit not only from my motivating insights into the PhD process, but also my robust views on whatever happened to be the latest story on the BBC website.

Chapter 1

Introduction

1.1 Problem Domain

The inability of current computer systems to adequately process text is a major research challenge. Despite the phenomenal growth in the use of databases in the last 30 years , 80% of the information stored by companies is believed to be unstructured text [Tan, A.H. 1999][1] and this includes documents such as contracts, research reports, specifications and email.

Beyond the workplace, the explosion of predominantly textual information made available on the web has led to the vision of a "machine tractable" *Semantic Web*, with database-like functionality replacing today's book-like web [Berners-Lee, T. 1999].

Research activity in the semantic web cuts across boundaries and includes researchers from a wide range of disciplines, including natural language processing, databases, data mining, AI and others. In industry, specialist vendors such as Autonomy [Autonomy] provide document management systems which attempt to make use of unstructured data. As well as these niche providers, general software developers regard this as a major area for the future; for example, IBM has over 200 people working on unstructured information management research [Ferrucci, D. and Lally, A. 2004].

[1] The figure of 80% (or sometimes 80-85%) of corporate information being text is widely quoted. References, where given, lead to [Tan 1999] or to a quote in a magazine article [Moore 2002]. While this statistic seems plausible, despite entering into correspondence with the sources we are unable to verify how this figure was obtained or what it means in detail.

In this chapter, we describe the characteristics of a class of application that rely on text data; this application class forms the problem domain addressed by this thesis. We discuss why combining two technologies, namely *Data Integration* and *Information Extraction*, offers potential for addressing the information management needs of these applications, and we outline the structure of the thesis.

1.1.1 Partially Structured Data

In this class of applications, the information to be stored consists partly of some structured data conforming to a schema and partly of information left as free text. This kind of data is termed *partially structured data* in [King, P. and Poulovassilis, A. 2000]. Partially structured data is distinct from semi-structured data, which is generally regarded as data that is self-describing: in semi-structured data there may not be a schema defined but the data itself contains some structural information, for example XML tags. In contrast, the text in partially structured data has no structure.

Examples of applications that generate and query partially structured data include: UK Road Traffic Accident reports, where data conforming to a standard format is combined with free text accounts written in a formalised subset of English; crime investigation operational intelligence gathering, where textual observations are associated with structured data about people and places; and Bioinformatics, where databases such as SWISS-PROT [Bairoch, A., Boeckmann, B. et al. 2004] include comment fields containing unstructured information related to the structured data.

1.2 Our Approach

Our methodology for undertaking the research reported in this thesis can be summarised as follows:

1) Consideration of the characteristics of partially structured data: We considered the characteristics of partially structured data and of current applications that are based on such data. As a result, we have identified two reasons for application

designers being forced to resort to storing information as text in partially structured data applications:

Firstly, it may not be possible in advance to know all of the queries that will be required in the future, and the text captured represents an attempt to provide all information that could possibly be relevant in the future. Road Traffic Accident reports are an example of this: the schema of the structured part of the data covers the currently known requirements while the text part is used when new reporting requirements arise.

The second reason for partially structured data arising is that data needs to be captured as text due to the limitations of supporting dynamically evolving schemas in conventional databases — simply adding a column to an existing table can be a major task in production systems. For example, in systems storing narrative police reports, when entity types or relationships are encountered for the first time it is not possible to dynamically expand the underlying database schema and the new information is only stored in text form [Chau, M., Xu, J.J. et al. 2002].

2) Identification of candidate technologies: With these reasons in mind we reviewed the literature for state-of-the-art technologies for dealing with both structured and textual data, in particular to identify areas where techniques from one area could benefit the other. Until recently, two main approaches existed that aimed to exploit textual data: *Information Retrieval,* which examines text documents and returns a set of potentially relevant documents with respect to a user query, and *Natural Language Processing* (NLP), which applies knowledge of language to construct a representation of what the text represents, for example as a syntax tree. NLP techniques can be characterised according to how ambitious the linguistic processing is: morphological and lexical processing are relatively straightforward compared to syntax, semantics and pragmatics (see for example the discussion in [Gazdar, G. and Mellish, C. 1989]).

Despite early optimism for NLP e.g. [Winograd, T. 1972], NLP techniques which aim to make use of semantics and pragmatics have shown disappointing results in real-world applications to date, and applications typically fail to scale when real-world knowledge bases are required. However, during the 1990's impressive results were obtained with *Information Extraction* (IE), a branch of NLP where pre-defined entities are extracted from text, for example people and company names from newswire reports.

Until now, IE systems have largely been developed as stand-alone components, configured by grammar rules and lists of entities developed for the particular application by a domain expert. The result of IE processing is either a set of annotations over the text, or a set of filled templates representing the instances of the target entitles found and possibly some pre-specified relations between them. In the literature on IE this tends to be the endpoint, with the precision and recall of the annotations found being the results measured in experiments. For practical applications, however, these annotations have need to be further processed, and the literature does not deal with this aspect often: there are some exceptions, for example [Nahm, U.Y. and Mooney, R. 2000] create a structured dataset for use in a data mining system.

Therefore, although IE is a promising technology for processing text, for the applications that we have outlined above there are several limitations: IE systems to date are configured from scratch and do not make use of any related structured data; and the results output by IE systems are produced as an independent dataset, with further processing being required in order to combine these results with any already existing structured data.

In *data integration* systems, a unified view of data is provided over a number of data sources each of which may be structured according to different data models. The ability to make use of structured data in a variety of formats is particularly important for partially structured data applications within large corporate environments. Data integration systems are available which can combine data modelled in most of the common structured and semi-structured data models e.g. relational, ER, XML, flat-file, and object-oriented — but, to our knowledge, none has attempted so far to provide support for unstructured text.

3) Research into combining techniques from data integration and IE: In this thesis, we explore how the combination of techniques drawn from data integration and IE can provide a basis for more effective support of partially structured data applications than either data integration or IE systems alone. We have identified a number of areas of synergy between these two types of system, which form the basis of our approach:

- IE is based on filling pre-defined templates, and data integration can provide a global schema to be used as such a template. This global schema / template can be created by combining the schema of the structured data together with ontologies and other available metadata sources.
- Metadata from the data sources can be used to assist the IE process by semi-automatically creating the required input to the IE process.
- As new entity types become known over time, data integration systems that use a graph-based common data model are able to extend their global schema without the limitations associated with conventional record-based databases [Kent, W. 1979].
- The templates filled by the IE process can be stored as a new data source that can be integrated into the global schema, supporting new queries which could not previously have been answered.

4) Development of a prototype system: We identified an existing state-of-the-art IE system (GATE) and a data integration system (AutoMed), and made use of their facilities to perform routine IE and data integration functionality, allowing us to focus our research into new techniques which exploit the synergies and mutual benefits of the two approaches. In particular, we have designed and implemented an

Experimental System To Extract Structure from Text (ESTEST), which has a number of novel features:

- This is the first time, to our knowledge, that a data integration system has been extended to include support for free text.
- One recent approach to extracting information from text is to make use of an ontology to assist in IE e.g. [Popov, B., Kiryakov, A. et al. 2003]. ESTEST uses data integration techniques to integrate available structured data into a global schema which is then used as a lightweight ontology (that is, an ontology with few axioms) for semantic annotation — this is a realistic application-specific alternative to the time consuming task of building ontologies from scratch.
- The global schema is used by ESTEST to semi-automatically configure the IE process, thereby reducing the configuration overhead of the IE process.
- ESTEST provides a domain-independent method of automatically extracting and integrating into pre-existing structured data the values that are found within the annotations output from IE, whereas up to now application-specific methods have been used for this purpose.
- We have developed a novel schema matching component which employs an IE pre-processor to make use of textual information associated with schema elements in order to generate mappings between schema elements in different data source schemas.
- In NLP, a core task is *co-reference detection,* which finds when there are multiple references to the same entity. For example, in the sentence "John sat on the chair. He fell off it" there are two references to the same person and two to the same chair. We have applied database identifier disambiguation techniques to the NLP co-reference problem in order to improve on the accuracy that can be obtained by using either approach in isolation.

5) Evaluation: ESTEST has been successfully applied to the Road Traffic Accident and Crime domains, and experimental results have been obtained which are discussed in the thesis. As well as demonstrating the system in use, we have also discussed with experts from a range of domains their use of partially structured data, in order to confirm our understanding of the reasons for information being stored in this way, and of the potential of our approach for improving upon the partly manual methods that are employed at present.

1.3 Structure of the Thesis

The rest of the thesis is structured as follows. Chapter 2 describes related work in managing structured data in databases, database systems and data integration, as it applies to the problem domain that we are addressing. In Chapter 3 we discuss Natural Language Processing and Information Extraction, and give an overview of the GATE IE system. The AutoMed data integration system is reviewed in Chapter 4, and we also give details of the extensions to AutoMed that we have made in order to support the development of ESTEST. Chapter 5 describes the design of the ESTEST system. In Chapter 6 we present an evaluation of ESTEST in the context of the Road Traffic Accident domain and we use this domain in an example that illustrates the system in use. A number of further innovations arose from this evaluation and these are described in Chapter 7. Finally, in Chapter 8 we give our conclusions and directions of possible future work.

Chapter 2

Databases and Data Integration

In this chapter we describe the limitations of current database management systems with regard to exploiting text. We summarise recent advances in processing textual data and we indicate why these approaches are more suited to text that has some, albeit loose, structure and not the free text that occurs in the class of application we outlined in Chapter 1.

We then describe work on graph-based data models and show how these are well-suited to our problem domain as they allow more flexibility in evolving a schema after its initial definition. Finally, we review data integration systems and indicate why some of the new approaches in this field are applicable to the applications that we are addressing.

2.1 Conventional Database Provision for Text

Database Management Systems (DBMS) organise data according to a schema in order to allow retrieval of data via a structured query language. The schema is defined in terms of the constructs of some data model, for example the tables and columns of the relational model in a relational DBMS. The data is formatted according to the schema prior to populating the database. A range of data types are provided, including support for strings, for example the common data types char and varchar. In some systems, strings can be constrained according to patterns e.g. for US ZIP codes. These data types are intended to support string values occurring within structured data rather than large quantities of free text.

DBMS providers argued throughout the 1980's that databases were suitable only for structured data and that file systems were the appropriate location for unstructured documents. However, market pressure led to data types for larger pieces of text e.g. the Postgres `text` data type. Limited facilities are available for making use of this text e.g. pattern searching with wildcards or, at best, regular expressions. Blobs[2] are now provided in most relational DBMS to deal with large unstructured objects. Blobs, in particular, were introduced against opposition from DBMS developers, who argued that documents should be broken down and normalised. However, new requirements to be able to store and retrieve images, video and audio in databases have led to their current widespread availability.

The relational model [Codd, E.F. 1970] has dominated the commercial database market since the late 1980's [Sciences, N.A.o. 1999]. More recent developments in data management have included Object-Oriented Databases and Object-Oriented extensions to relational databases. However, all of these are essentially record-based in that data is retrieved as a set of fields in a fixed order, for example the values of all the fields of a row from a table. The limitations of record-based approaches are summarised in [Kent, W. 1979] and include i) the assumption of problems of homogeneity of record formats making it difficult to represent general concepts with a varied collection of attributes over their populations, ii) difficulties in extending record formats, and iii) handling missing data.

For the class of applications that we consider in this thesis, record-based DBMS are too inflexible because as requirements for new types of entities and relationships arise it is hard to extend the database schema. XML databases have provided some more flexibility by allowing data to include tags which do not have to be predefined in an associated schema. However, to support the flexibility of schema evolution as part

[2] Blobs were first developed as part of the DEC Rdb database in the 1980's. Although the marketing departments of database vendors have attributed various acronyms to them e.g. Basic Large Object, Binary Large Object, in fact they were named after the 1958 Steve McQueen film "The Blob" as it was feared that these large objects would eat up storage [Harrison, A. 1997].

of normal use, we argue that graph-based data models (discussed further in Section 2.4) are more suitable for this class of application.

2.2 Semi-Structured Data and Data on the Web

The challenge of the huge amount of, mainly textual, data on the web has resulted in a number of new research directions from the database community, including the emergence of semi-structured data models such as XML [Beech, D., Malhotra, A. et al. 1999] and RDFS [McBride, B. and Hayes, P. 2002].

Database techniques applied directly to web data include web query languages, information integration and website restructuring – these are discussed in [Florescu, D., Levy, A. et al. 1998] for example. Systems aiming to make use of web data tend to make use of either a graph-based or a semi-structured data model. Semi-structured data is generally regarded as data that is “self-describing”, i.e. the schema may not be known in advance but schema information may accompany the data e.g. in the form of XML tags or RDFS statements.

In the graph-based approach, the web is regarded as a graph whose nodes are web pages and edges are the hyperlinks between pages. Simple queries can be posed on the contents and link structure connecting the pages, and return those pages that contain text matching the search arguments. However, such a query mechanism returning potentially useful web pages is not a great advance on a search engine. Many web sites are front-ends for displaying structured data e.g. the online yellow pages Yell.com provides the contact information for a list of companies matching some query. Extracting information from such pages is often achieved by developing site-specific wrappers – and the overhead of wrapper creation or induction [Freitag, D. and Kushmerick, N. 2000] is the main disadvantage to this approach.

NoDose [Adelberg, B. 1998] is able to semi-automatically derive structure from formatted text even where tags are not present, using methods such as detection of repeating strings e.g. “from” and ”cc” in files containing emails. A GUI also allows the user to highlight structural elements to be extracted from the rest of the file. The structure that NoDose derives is used to extract information from further instances of the text, which is assumed to be formatted in the same way. The tool is not, therefore, intended to be applied to free text.

A number of researchers have developed techniques that deploy machine-learning algorithms to handle information stored in web pages. For example, Snowball [Agichtein, E., Gravano, L. et al. 2001] and DIPRE [Brin, S. 1999] extract structured relations from the web. They do so by using a set of seed examples to find a larger set of occurrences of a relation (e.g. author – book) on the web. From this larger set of occurrences, they generate a pattern general enough to cover the results but specific enough not to return false matches. Rapier [Califf, M.E. and Mooney, R.J. 1999] similarly uses a set of examples and correctly filled templates to generate patterns for information extraction. These systems assume that relations described within web pages will usually occur in similar contexts. For example, for statements of the form "Dan Brown the author of the Da Vinci Code", a rule "<author> the author of <book>" would be generally useful.

2.3 Text Mining

Text Mining [Tan, A.H. 1999] is an extension of the area of data mining or knowledge discovery from databases. In data mining, previously unknown patterns are discovered within large datasets. In text mining, a technique such as information retrieval or summarisation typically transforms the text corpus to create a structured dataset for subsequent data mining.

[Mooney, R.J. and Nahm, U.Y. 2005] combine text mining with information extraction in the DiscoTEX system. This uses two systems to learn information extraction rules from a set of training data: Rapier [Califf, M.E. and Mooney, R.J. 1999] and Boosted Wrapper Induction [Freitag, D. and Kushmerick, N. 2000]. It then applies these rules to the text corpus in order to produce a structured dataset on which conventional data mining techniques are applied. IE rules can be produced from mined associations and reapplied to new instances of similar text.

For the partially structured data applications with which we are concerned, the text mining approach is unlikely to be effective as there are not very large static datasets to be mined (although there are some exceptions, for example the SWISS-PROT database). Rather, over time, new query requirements arise and extensions to the schema are required, that need to be populated by the structured data extracted from the free text.

2.4 Graph Based Data Models

In Section 2.1 we argued that the dominance of relational database systems in real-world applications has been achieved despite a number of limitations, especially in terms of flexibility in enhancing the schema after the initial database set-up. For example, in large-scale commercial environments with which we are familiar, changes to the existing schema are avoided wherever possible, comprehensive changes to the schema of core databases are very rare, and even small changes are rare, with new linked tables often being created rather than existing tables being modified.

The *binary-relational* data model [Frost, R.A. 1982] does not suffer from the schema evolution problems of record-based structures. In this model every real-world concept of interest is an entity type (c.f. a node in a graph) and associations between entity types are modelled by binary relationships (c.f. directed edges in graphs). A database is a set of these binary relationship “triples”, together with the extents of the entity types. The disadvantage of this approach is one of efficiency, in that retrieving related values will generally be slower than if they were stored together within a single record.

The Tristarp project [King, P., Derakhshan, M. et al. 1990; Tristarp] has developed a number of graph-based database prototypes, including triple stores for storing information in binary-relational storage structures, functional database languages such as FDL [Poulovassilis, A. 1992] which provides a functional approach to manipulating Tristarp's binary-relational data model, and an alternative, logic-based manipulation language [King, P.J.H. and Small, C. 1991] . The *hypergraph data model* (HDM) used in the AutoMed heterogeneous data integration system (discussed further in Chapter 4) is also graph-based, and its associated query language, IQL [Poulovassilis, A. 2001], is a functional language. Recent work in Tristarp [Smith, M.N. and King, P.J.H. 2004] has seen the implementation of the EDVC graphical interface for incrementally creating database views over complex graphs. This has been applied to the crime domain, in particular providing a visual query language for producing the *link charts* used in criminal intelligence analysis. A commercial database system, Sentences [Williams, S. 2002], supporting a variant of the binary relational model has also been developed and this is used as the triple store underlying the EDVC tool.

The commercial crime analysis system Xanalys produces link charts and has a related text extraction tool, the Xanalys Indexer [Xanalys]. This uses a preconfigured information extraction system to extract data which can then appear in Xanalys link charts[3]. However, the grammar rules used for extracting information are hard-coded and are produced afresh for each application domain, such as crime or finance. Changes to grammar rules must be made by developers and cannot be made by users.

Research on triple stores has also increased recently due to the development of the *Resource Description Framework* (RDF) for the semantic web and the need for repositories for RDF triples, for example [Broekstra, J., Kampman, A. et al. 2002; Harris, S. and Gibbins, N. 2003].

[3] Formerly Xanalys was known as Watson, and Xanalys Indexer known as both PowerIndexer and Quenza.

2.5 Data Integration

Data integration systems provide a unified view of data stored in a number of different sources. The data sources, each with an associated schema, are integrated to form a single virtual database, with an associated global schema. If the data sources conform to different data models, then these need to be transformed into a common data model as part of the integration process.

Two main approaches have been adopted to date [Lenzerini, M. 2002]: *global-as-view* (GAV) where the global schema is defined as a set of views over the data sources, and *local-as-view* (LAV) where the global schema is independently created and each source is defined as a view over the global schema. A number of systems have been developed which implement these approaches e.g. [Chawathe, S.S., Garcia-Molina, H. et al. 1994; Roth, M.T. and Schwarz, P. 1997] implement GAV and [Levy, A., Rajaraman, A. et al. 1996; Manolescu, I., Florescu, D. et al. 2001] implement LAV. Some integration tasks require the expressive power of both GAV and LAV and a variation on LAV has been proposed for this purpose — *global-local-as-view* (GLAV) [Friedman, M., Levy, A.Y. et al. 1999].

Recently a new approach, *both-as-view* (BAV) data integration [McBrien, P.J. and Poulovassilis, A. 2003] has been proposed, and the AutoMed data integration system implements this approach. BAV is based on the use of reversible sequences of primitive schema transformations, termed *pathways*. BAV combines the benefits of GAV and LAV (and indeed GLAV) in that from these pathways it is possible to extract both a definition of the global schema as a view over the local schemas (GAV) and definitions of the local schemas as views over the global schema (LAV and GLAV).

One of the main advantages of using a BAV rather than a LAV, GAV or GLAV-based data integration system in our context is that BAV readily supports the evolution of both source and integrated schemas by allowing transformation pathways to be extended: if a new data source is added, or if the schema of a source or integrated schema evolves, the entire integration process does not have to be repeated and instead the schemas and transformation pathways can be 'repaired' [Fan, H. and Poulovassilis., A. 2004; McBrien, P.J. and Poulovassilis, A. 2002]. As these situations are all characteristic of the class of application with which we are concerned in this thesis, this capability is beneficial for our approach and as a result we have made use of facilities provided by the AutoMed system for our research and for the implementation of the ETEST system.

2.5.1 Schema Matching

A central challenge in data integration is *schema matching* — that is, given a set of data sources we need to identify correspondences between pairs of elements in their schemas as a prerequisite to defining mappings. [Rahm, E. and Bernstein, P.A. 2001] gives a review of the state of the art in automatic or semi-automatic techniques for schema matching, and classifies the approaches using the following criteria: instance-based versus schema-based; element versus structure based; language versus constraint-based; those based on cardinality matching; and those making use of other auxiliary information, for example as provided by the user. The language-based approaches include matching the names of schema elements, and making use of synonyms. While the possibility of making use of textual metadata beyond element names (for example, comments or descriptions) is considered in [Rahm, E. and Bernstein, P.A. 2001], no systems are cited that use this approach and we are aware of none other than our own ESTEST system that does this (as discussed in Section 7.2).

The LSD system [Doan, A., Domingos, P. et al. 2000] learns correspondences between schema elements, deploying a number of machine learning approaches to try to identify correspondences: these include a Naïve Bayesian learner [Domingos, P. and Pazzani, M. 1997], some domain specific resources such as recognition of the names of US counties, and WHIRL [Cohen, W.W. 1998] which assumes that the names of schema elements will be given in natural language and therefore that the textual similarity between these names can be used to find mappings. In particular, WHIRL combines cosine similarity from information retrieval with the widely used *tf.idf* weighting scheme [Salton, G. and McGill, M. 1983] – the two strings to compared are split into words and compared, and a weighting system prefers matches between words which occur infrequently across the whole corpus. For example, “Mr George Bush” and “Bush, George” would have a high similarity using this approach despite the different order of “George” and “Bush” and because “Mr” will receive a lower weight due to its relative frequency in the corpus.

These systems are targeted at textual data in the form of web pages and, like the semi-structured work discussed in Section 2.2, they assume the existence of some structure within the text to be extracted. While a number of systems have used textual information, especially element names, as part of schema matching, we are aware of no previous system that makes use of the all available text metadata (including schema element descriptions) as ESTEST does.

2.6 Discussion

We are seeking to improve support for applications that generate, query and manipulate a combination of free text and structured data. In such applications, unstructured text may have been used either because it is not fully known in advance what queries will be required and therefore a complete schema cannot be constructed, or because new entity types and relationships become apparent over time and it is difficult to extend the schema of record-based DBMS.

Commercially available DBMS provide very few facilities for exploiting free text. Recent advances in processing textual data on the web apply to semi-structured data rather than to free text with related structured data. Text mining finds previously unknown patterns within text and can usefully be combined with information extraction; however, this approach is not generally applicable to our setting which is characterised by new information requirements arising over time.

Graph-based data models do offer the flexibility that we require with respect to schema evolution, while data integration systems allow available structured data to be combined into an integrated schema. The BAV data integration approach supports incremental evolution of the integrated schema, and the implementation of the BAV approach within the AutoMed system uses a graph-based metamodel. Therefore, we have made use of, and have extended, the facilities offered by AutoMed in order to combine structured data sources in our ESTEST system.

Schema matching is a central problem in data integration, and recent work has investigated integrating semi-structured text on the web, for example by making use of textual similarities to identify correspondences. However, we are aware of no previous system that has made use of the full range of available text metadata for schema matching, which is a feature of our ESTEST system.

Chapter 3

Information Extraction

In this chapter we discuss research in *natural language processing* (NLP)[4], and we argue that *information extraction* (IE) is the most suitable branch of NLP for use in the class of applications that we have identified. We review both IE research and the area of *language engineering*, which aims to develop NLP applications that reuse common components. We discuss why the GATE system, which is designed for language engineering, and focuses on information extraction, is suitable for use in our ESTEST system. Finally, we outline recent related work that has taken place in parallel with our research.

3.1 Natural Language Processing

NLP research began in the 1960's amid much optimism, and was funded by US military and government-sponsored programmes to develop *machine translation* applications. These failed when it became clear that word-for-word substitution was not capable of scaling up to the volumes required by real-world applications. A common theme in NLP research since then has been optimism in demonstration systems being followed by disappointing results when attempting to scale from knowledge-bases capable of supporting the demonstration to those required for real-world problems. NLP research is categorised in [Gazdar, G. 1996] into the following areas: *theory of linguistic computation*, *computational psycholinguistics*, and

[4] Both Natural Language Processing (NLP) and Computational Linguistics (CL) are terms used to describe the processing of text by computers. There is considerable overlap between the terms but, in general, NLP is a computer science term for processing text and CL is a linguistics term for using computers. Throughout we use NLP to refer to the domain covered by both terms.

applied NLP. The first two are not concerned with developing solutions to real-world problems but concentrate instead on using computer software to gain insights into language and human psychology. Across the three areas there are also three kinds of task: *language understanding*, *language generation*, and *language acquisition*. According to this classification, making use of text in databases falls within applied NLP and is to be achieved through the language understanding task. Therefore, this is the area of NLP that we focus on in this chapter.

A milestone for language understanding was SHRDLU [Winograd, T. 1972], a system that modelled a world of blocks and supported natural language queries such as “is anything on top of the small red block?”. Instructions to move objects, such as “put the blue triangle on the big box”, could be submitted to the system and the results were displayed in a graphical representation of the world. However, approaches such as SHRDLU, which were based on a complete representation of the world to be modelled, proved difficult to scale up to real-world applications and during the 1970s were increasingly replaced by knowledge-based approaches based instead on heuristics for reasoning about the world.

Two main approaches to natural language understanding were pursued throughout the 1980s: *symbolic,* where the focus was on parsing the text in order to build a structure that can be transformed into a representation of the meaning of the text; and *probabilistic,* where frequencies of words in the text are calculated and these statistics are used to predict the occurrence of words based on the words that precede or follow. This approach is more successful in narrow domains, or where patterns occur in the way the text is used, for example the parliamentary language used in Hansard [McEnery, T. and Wilson, A. 1996]. However, again, applications developed from either approach have not proved scalable to real-world problems.

Whichever the approach adopted, eventually the problem becomes what is described as *AI-complete* [Rich, E. and Knight, K. 1991] – that is, it depends on a solution to the general AI problem of having wide-ranging knowledge about the world and the ability to be able to reason on that knowledge. This is also loosely described as “common sense” or “general knowledge”. Syntactic ambiguity (e.g. the four possible interpretations of the well-known example “Flying planes made her duck”) in natural language is an example of an AI-complete problem [Hobbs, J.R., Appelt, D. et al. 1996].

During the 1990’s, more focus in research was given to the evaluation of the results of systems, due in part to the importance placed on empirical evaluation in projects funded by the US government agency DARPA [DARPA].

Information Extraction (IE) [Appelt, D.E. 1999; Cowie, J. and Lehnert, W. 1996] emerged as a technology for retrieving structured data from within text documents. IE is often defined by contrasting it to *information retrieval* (IR) – IR returns a set of relevant documents in a corpus, whereas IE returns facts from within the documents about previously identified entities and relationships. In contrast to their predecessors, IE systems are more pragmatic and use only more straightforward techniques, which we describe below.

The description of the IE system FAUSTUS in [Hobbs, J.R., Appelt, D. et al. 1996] goes so far as to argue that IE is not text understanding at all in the established sense, as in IE systems only a fraction of the text is of interest; information is mapped onto a pre-defined target representation rather than parsing text and transforming the parse tree into a representation of knowledge; and there is only a very limited attempt to deal with complex features of language such as semantics.

The results of IE systems have been impressive: levels of accuracy achieved have been close to those achieved by human experts manually marking-up newswire reports [Marsh, E. 1998], while its limitations are not problematic for our problem domain: as new entity types are required or new query requirements arise, only the part of the text relating to these is of interest to us; the integrated schema built over the data sources can be used to create templates for the IE extraction and the extracted information can be integrated into this virtual global schema. It is for these reasons that we selected IE for our ESTEST system.

3.2 Information Extraction

Current IE research has been influenced and developed by the Message Understanding Conference (MUC) series [Grishman, R. and Sundheim, B.]. These MUC conferences, which ran from 1987 to 1998, were part of the TIPSTER text programme [TIPSTER] funded by DARPA. This programme had three threads: document detection, information extraction, and summarisation. As well as MUC, the Text Retrieval series of conferences were also run as part of TIPSTER.

At each MUC conference, an application was defined and researchers developed systems which competed to achieve the highest levels of recall and precision for a number of IE tasks performed on the same previously unseen dataset. Early conferences focused on extracting information from military messages, and in later conferences this theme was developed to cover newswire reports. The components found in IE systems today largely reflect the tasks set in these conferences. The tasks for the last conference, MUC-7, in 1998 (the most challenging in the series) were as follows:

1) Named Entity Recognition: at its simplest this involves identifying proper names in text by matching against lists of known entity values, although patterns can also be defined to recognise entities. For the MUC tests, the entities to be identified were known in advance and so targeted recognition rules could be developed specifically to identify text patterns associated with these entities.

2) Coreference detection: resolution of multiple references to the same entity within the text. This is a relatively hard NLP task and was introduced because of the benefit even poor results in this task can bring to the other tasks, rather than just as an objective in itself.

3) Template Element: this is a task for identifying references to entities as well as their names. It is harder to detect references e.g. 'the manufacturer of the DB2 database' as opposed to 'IBM', and results for this task are less accurate.

4) Template Relation: instances of a small number of pre-specified relations are to be found e.g. people who are employees of companies.

5) Scenario Template: scenarios link template instances describing an event, for example in a newswire corporate merger scenario, the companies, executives and their positions would be extracted into a scenario template.

The Named Entity Recognition task has been the focus of much of the research to date; it has achieved the best results, and is the least domain specific. The Template Relation and Scenario Template define relations between annotations, but the literature contains little about techniques for performing these tasks. For MUC-6 and MUC-7 the specifications of these tasks were given out a month before the competition — this timing was intended to show that the systems could be configured for specific domains in reasonable time and is therefore also an indication that generic systems did not yet exist for these tasks; this continues to be the case.

While IE systems have encouraged the reuse of generic components such as sentence splitters, tokenisers and configurable Named Entity Recognition components, this has not extended to techniques for creating structured data from extracted annotations, and the processing of the annotation is implemented from scratch for each system (whereas annotations are in fact processed further, for researchers into IE producing the annotations is often the final goal).

Most systems in MUC-7 achieved higher than 80% success for named entity recognition, with around half achieving over 90%. Even for the Scenario Template task results were credible, ranging from 40% to 70% [Marsh, E. 1998].

For a fair comparison of the systems participating in the conferences, they needed to be compliant with the TIPSTER architecture [Grishman, R. 1998], which describes the system components comprising IE systems and, more specifically, the definition of the text annotation structure. Systems implemented in this way are described as being TIPSTER compliant.

In the literature, the performance of IE systems is usually measured in terms of *recall* (the number of correctly identified references of an entity type as a percentage of the actual number occurring in the text) and *precision* (the number of correctly identified instances as a percentage of all the identified instances). Depending on the application, recall and precision can be of different value and are often combined in an *F-measure*, a geometric mean where recall and precision are weighted to indicate their relative importance [Appelt, D.E. 1999].

An alternative to basing IE systems on grammar rules is to use a machine learning approach to induce rules by mining a training dataset [Basili, R., Pazienza, M. et al. 2002; Califf, M.E. and Mooney, R.J. 1999; Nahm, U.Y. and Mooney, R. 2000]. For the class of applications that we are concerned with, new entity types and query requirements arise over time. Thus, making use of IE based on grammar rules is more readily applicable, although we do not discount the possibility of using pattern rule mining in some applications where large static datasets exist, and is this a possible area of future work.

Even though IE systems in the 1990's were constructed independently and from scratch, most of them share some common features. Generally they are constructed as a number of components each performing a discrete task. These tasks run in a sequence, commonly referred to as a *pipeline* [Graca, J., Mamede, N.J. et al. 2004]. Many of the individual components of an IE system are more generally applicable e.g. any English IE system will need an English Tokeniser. IE systems also typically use some kind of annotation graph as output, and components use this to pass information between components in the pipeline.

A number of initiatives encourage reuse and standardisation of IE components and *language engineering* [Boguraev, B., Garigliano, R. et al. 1995] is emerging as an accepted term for attempts to apply a software engineering approach to NLP systems.

Many systems developed their own annotation format, because the TIPSTER model used in MUC proved unable to meet their requirements. The ATLAS architecture and API [Laprun, C., Fiscus, J. et al. 1999] provides a formal and extensible model for annotations. ATLAS aims to become the standard model for annotation representation and is supported by the US government National Institute of Standards and Technology (NIST) agency [NIST]. IBM has developed its Unstructured Information Management Architecture (UIMA) [Ferrucci, D. and Lally, A. 2004], a framework for handling unstructured information in the workplace. This is designed to be an industrial strength language engineering framework which will allow analysis components, both commercially developed and those from universities, to be combined and to process unstructured text, video and audio information.

For our requirements, namely to utilise IE in our class of applications, the GATE (General Architecture for Text Engineering) system [Cunningham, H., Maynard, D. et al. 2002] was selected as the most suitable system for a number of reasons. GATE is the most widely used open-source IE system and in its current version has been re-implemented according to language engineering principles. It provides a complete, general IE system and a wide range of third-party developed components are also available. Its language engineering architecture allows the development of bespoke applications and components. GATE conforms to the TIPSTER annotation model and offers near compatibility with the ATLAS model. Work is under way to allow GATE and IBM's UIMA unstructured information initiative to be integrated. Finally, it is unusual for UK academic software to be able to adequately support non-research aspects of software such as ensuring stability, versioning, providing easy installers and providing support to end users – and the GATE system is able to do so.

3.3 The GATE Information Extraction System

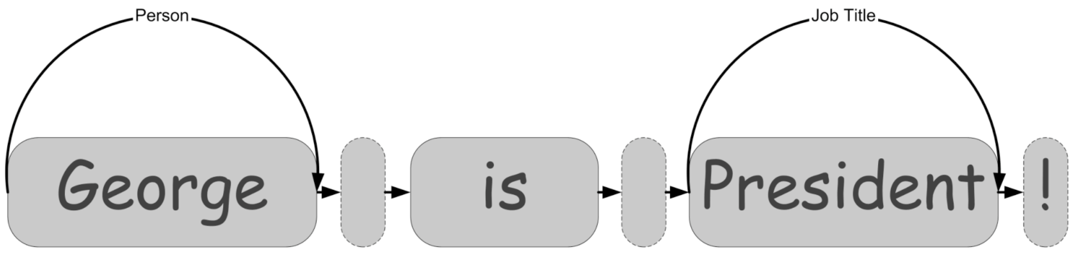
GATE was originally developed at the University of Sheffield as an implementation of the TIPSTER architecture specification. A number of IE systems have been built using GATE, including the LaSIE-II system which competed in MUC-7 [Humphreys, K., Gaizauskas, R. et al. 1998].

GATE 2 was developed as an implementation of the *software architecture for language engineering* (SALE) [Cunningham, H. 2000] which offers a framework for developing IE and other more general text processing components; for example IR and Google components have been implemented. Three types of component exist in GATE 2: *(i) language resources* — such as documents or corpora, *(ii) processing resources* — which apply an algorithm to some language resources, and *(iii) visual resources* — used in the GUI to represent either language resources or to configure or view the results of processing resources. While GATE 2 remains focused on IE tasks, the importance placed on an extensible language engineering architecture means that GATE 2 is now also suitable for more general NLP application development; examples of non-IE components currently available include machine learning components such as the WEKA wrapper [WEKA], and ontology resources such as the Protégé editor [Noy, N.F., Sintek, M. et al. 2001].

According to the TIPSTER architecture, the result of running a processing resource is a set of *annotations* over the text. Each annotation has an associated start and end position within the text. The annotation has a type and may have features. An annotation can be thought of as an arc in a directed acyclic graph whose nodes are tokens within a string and whose edges indicate the type of annotation represented by the tokens between the start and end of the edge. For example, for the string "George is President!" some of the annotations that might be created are as follows:

Annotation	**Start**	**Finish**	**Features**
FirstPerson	0	6	{gender=male, rule=FirstName}
Lookup	0	6	{majorType=person_first, minorType=male}
Person	0	6	{gender=male, rule=PersonFinal, rule1=GazPersonFirst}
Sentence	0	20	{}
Token	0	6	{category=NNP, kind=word, length=6, orth=upperInitial, string=George}
SpaceToken	6	7	{kind=space, length=1, string= }
Token	7	9	{category=VBZ, kind=word, length=2, orth=lowercase, string=is}
SpaceToken	9	10	{kind=space, length=1, string= }
Lookup	10	19	{majorType=jobtitle}
Title	10	19	{rule=Title}
JobTitle	10	19	{rule=JobTitle1}
Token	10	19	{category=NNP, kind=word, length=9, orth=upperInitial, string=President}
Lookup	10	19	{majorType=title, minorType=civilian}
Token	19	20	{category=., kind=punctuation, length=1, string=!}
Split	19	20	{kind=internal}

A graph-based representation of the key information, with tokens as nodes, is as follows:



IE applications are built by constructing a chain of processing resources each targeted at a collection of documents. The set of annotations produced by a processing resource is accessible to those further down the chain.

GATE 2 is implemented in Java. The distribution includes a collection of standard IE components, including the ones we have used in ESTEST: the *document reset* component, which ensures the document is reset to its original state with any annotations removed; the *English tokeniser,* which splits text into tokens, such as strings or punctuation; and the *sentence splitter,* which divides text into sentences.

Gazetteers associate lists of values with entity types e.g. to recognise male first names. A pattern matching language, JAPE [Cunningham, H., Maynard, D. et al. 2002], is supported which allows rules to be defined for matching patterns in text. The *Jape Transducer* component takes text as its input and returns a set of annotations over the text. When a Jape rule fires an annotation is created and, optionally, bespoke Java code can be executed. Developers can develop new components as Java Beans implementing the relevant component interface.

Using the GATE component architecture, it is possible to assemble IE applications either using the GATE GUI or by using the GATE API to develop a stand-alone Java program. Third parties have now developed a range of GATE processing resources and these are included in the GATE distribution; many of these are not strictly for IE applications, for example traditional NLP parsers, Information Retrieval and Machine Learning components.

3.4 Recent Developments in Information Extraction

We describe now the developments in IE that have taken place since 2000, in parallel with the research in this thesis.

The emergence of the semantic web [Berners-Lee, T. 1999] as a research goal to enable machine processing of the content of the web has been a major focus for IE researchers e.g. [Amardeilh, F. and Francart, T. 2004; Bontcheva, K. and Cunningham, H. 2003; Popov, B., Kiryakov, A. et al. 2003] and we first drew attention to the potential application of our approach to this domain in [Williams, D. and Poulovassilis, A. 2003].

Since the end of the MUC tests, the focus on template and scenario extraction has diminished: for example, there are no template extraction components in GATE. The development of technologies for implementing and using *ontologies* is leading to ontologies replacing templates as the structure that IE systems use for describing relationships between references to named entities. An ontology specifies a common definition of concepts and the relationships between them for some domain [Gruber, T. 1993]. The term is taken from philosophy where it means a set of categories to describe a particular view of the world. There is now wide research interest in using ontologies in information systems e.g. [Guarino, N. 1998], where ontologies are seen as enabling knowledge sharing.

The limitations of the conventional named entity recognition approach of applying to an annotation an entity type selected from a list of entity types has been recognised as a weakness of IE systems in the context of the semantic web, and researchers have recently sought to provide facilities for *semantic annotation* which assigns to an annotation a reference to a concept within an ontology rather than an entity type [Uren, V., Cimiano, P. et al. 2006]. This recent work is the closest to our approach in ESTEST in which annotations reference elements in a global schema.

The ACE programme [Doddington, G., Mitchell, A. et al. 2004] is, like MUC, administered by US government agencies, in this case the National Security Agency (NSA), National Institute of Standards and Technology (NIST) and the Central Intelligence Agency (CIA). Although the tasks in ACE appear to mirror those in the MUC programme (its *entity detection and tracking* is the equivalent of named entity recognition, while its *relation detection and characterisation* is equivalent to the template element task), this series of tests is more challenging than MUC because of the focus on a deeper semantic analysis of the text. Another extension of this programme is the requirement to be able to handle text from more varied domains and of varying quality e.g. text obtained from speech recognition and optical character recognition. A significant limitation of the ACE programme compared to its predecessors is that the results of the regular evaluations are closed and may not be published in the open literature. The MACE system was developed using GATE and has participated in the ACE evaluations [Maynard, D., Bontcheva, K. et al. 2003].

In GATE, support for ontologies is being developed [Bontcheva, K., Tablan, V. et al. 2004] and an ontology-aware gazetteer is now provided which is able to provide a link to a concept in an ontology as a feature of an annotation. KIM [Popov, B., Kiryakov, A. et al. 2003; Popov, B., Kiryakov, A. et al. 2004] is a system for semantic annotation which is built over GATE. Its main parts are the KIM Ontology (KIMO), a knowledge base, and an API for accessing the KIM functionality. The KIMO is an 'ontology of everything', designed to be domain independent. Its top level divides entity types into objects, events and abstract concepts. The KIM knowledge base stores instances of the types in the KIMO, and similarly aims to cover instances of the most important entity types in the real world. The current version of the KIM knowledge base includes 80,000 entities. While aiming for domain independence, the KIM researchers acknowledge the difficulties and concentrate on providing good coverage of entities mentioned in the news, that is, those of the core IE newswire mark-up task.

Semantic annotation has now replaced Template Relations and Scenario Templates as the active area of research for creating structured data from the annotations created by IE. However, it is less ambitious in the complexity of the structured data produced, and is an extension of named entity recognition rather than a replacement for the complex data structures envisaged during the MUC conferences. The ontologies produced to date by systems such as KIM are essentially taxonomies apart from the limited set of entity specific and hard-coded relationship rules.

3.5 Discussion

IE is distinguished in NLP research for the success it has achieved in real-world applications with respect to achieving near human success rates for tasks such as marking up newswire reports. While there are restrictions to the NLP tasks to which

IE can be applied, these are not restrictive for the class of applications that we have identified as the focus of this thesis.

By making use of data integration in our ESTEST system, it has been possible to apply IE in a novel way: rather than depending on hand-coded templates and rules, we can semi-automatically configure the IE process from a virtual global schema. The information extracted can be stored and is available for use by the query processing facilities of the data integration system. GATE is the most widely used system for IE and has been designed according to language engineering principles, making it well-suited to supporting the IE process integrated within our ESTEST system.

Recent research in the IE community, conducted in parallel with our own, has resulted in the emergence of a number of related directions of research: semantic annotation has similarities to the process that we have designed for ESTEST (described in Section 5.4), while the use in KIM of an ontology-of-everything has some parallels with our own use of a virtual global schema (described in Section 5.2). When considering the limited progress on generically extracting structured data from text both in the MUC Template Relation and the Scenario Template task, and also more recently with semantic annotation, we believe that just as databases have so far provided only limited support for free text, similarly IE has provided limited support for the extraction of structured data. This thesis seeks to demonstrate an approach which does provide the basis for a more generally applicable method of extracting structured data from the annotations produced by IE.

Chapter 4

The AutoMed Data Integration System

In our discussion of data integration in Chapter 2, we described how the AutoMed system is a suitable candidate for providing the data integration facilities to be used by our ESTEST system, and this chapter begins with an overview of AutoMed in Section 4.1. Two extensions were prerequisites for AutoMed's use in ESTEST: support for ontologies and a native HDM data repository. These features are required by ESTEST, but are also more generally applicable and so we developed them as extensions to the core AutoMed toolkit rather than as components of ESTEST – for this reason, they are described in this chapter rather than in the description of ESTEST itself in Chapter 5.

Ontologies have been a source of much recent research activity in both the database and the IE communities, mainly in relation to the semantic web. To build its virtual global schema, ESTEST combines the available structured data sources, and ontologies are likely to be amongst these. Therefore, we show in Section 4.2 how the constructs of RDF and RDFS can be mapped onto AutoMed's HDM data model. We also describe our implementation of an AutoMed RDFS wrapper.

A repository was also required for the information extracted by the ESTEST IE process, and a native HDM data store has been developed for this purpose. The HDM is the data model of this data source, and an appropriate wrapper for HDM sources has also been developed. The HDM data store and its wrapper are described in Section 4.3.

4.1 Overview of AutoMed

Data integration systems provide a unified view of data stored in a number of different sources. The data sources, each with an associated schema, are integrated to form a single virtual database with an associated global schema. If the data sources conform to different data models, then these need to be transformed into a common data model as part of the integration process.

AutoMed is able to support a variety of common data models by providing a graph-based metamodel, the Hypergraph Data Model (HDM) [Poulovassilis, A. and McBrien, P.J. 1998]. AutoMed provides facilities for specifying higher-level modeling languages in terms of this HDM e.g. the relational, entity-relational, XML models have been defined [McBrien, P.J. and Poulovassilis, A. 1999; McBrien, P.J. and Poulovassilis, A. 2001; McBrien, P.J. and Poulovassilis, A. 2002].

HDM schemas consist of nodes, edges and constraints, where a constraint is a boolean-valued query over the nodes and edges of the schema.

In order for a modelling language to be supported by AutoMed, each of its modelling constructs needs to be defined in terms of some combination of HDM nodes, edges and constraints. In AutoMed, instances of modelling constructs are uniquely identified by their *scheme*, enclosed within double chevrons, `<<...>>`. Each construct of a modelling language may be of one of the following four types:

1. *Nodal* constructs may exist independently of any other constructs in the model and are identified by their name. For example, an entity `e1` in the ER model is represented by a nodal construct with scheme `<<e1>>` and is defined in the HDM as a node.

2. *Link* constructs associate other constructs and can only exist when these other constructs exist. The extent of a link is a subset of the cartesian product of the extents of the constructs it depends on. Link constructs are defined by an

HDM edge and their scheme includes the schemes of the constructs that the link depends on. For example, a relationship r1 in the ER model between two entities e1 and e2 is represented by a scheme <<r1,e1,e2>>.

3. *Link-Nodal* constructs are nodal constructs that can only exist when one or more other constructs exist. They are represented by an HDM node and an HDM edge and are identified by a scheme including the scheme of the node and edge. For example, in the ER model, attributes are link-nodal and an ER attribute a of an entity e is identified by a scheme <<e,a>>.

4. *Constraint* constructs, unlike the other construct types, have no extent but instead define restrictions on the extents of other constructs. For example, in the relational model, primary keys are represented by specifying a constraint that states that for a particular column, or set of columns, duplicates are not allowed.

For any modelling language specified in this way (via the API of AutoMed's *Model Definitions Repository* [Boyd, M., McBrien, P.J. et al. 2002]), AutoMed automatically provides a set of primitive schema transformations that can be applied to schemas expressed in the modelling language. In particular, for every modelling construct there is an add and a delete primitive transformation which respectively add to, or delete from, a schema an instance of the construct. For those constructs which have textual names, there is also a rename primitive transformation.

AutoMed schemas are incrementally transformed and integrated by sequences of such transformations, termed *pathways*, each transformation affecting just one schema construct. All source, intermediate, and integrated schemas, and the pathways between them, are stored in AutoMed's *Schemas & Transformations Repository* (STR).

add and delete transformations are accompanied by a query (expressed in a functional query language, IQL [Poulovassilis, A. 2001]) which specifies the extent of the added or deleted construct in terms of the rest of the constructs in the schema.

Thus, these transformations do not change the *information capacity* [Milo, T. and Zohar, S. 1998] of the schema.

Also available are `extend` and `contract` primitive transformations which behave in the same way as `add` and `delete` except that they state that the extent of the new/removed construct cannot be precisely derived from the other constructs present in the schema. More specifically, each `extend` and `contract` transformation takes a pair of queries that specify a lower and an upper bound on the extent of the construct. The lower bound may be `Void` and the upper bound may be `Any`, which respectively indicate no known information about the lower or upper bound of the extent of the new construct.

The queries accompanying transformations can be used to translate queries or data along a transformation pathway. In particular, queries expressed in IQL can be posed on a virtual integrated schema, are reformulated by AutoMed's *Query Processor* into relevant sub-queries for each data source, and are sent to the data source *Wrappers* for evaluation. For each of the modelling languages defined in AutoMed, a Wrapper is available to extract details of any data source schema represented in that modelling language and for building a representation of this schema in AutoMed's Schemas and Transformations Repository. The Wrapper is also used during query processing, interacting with its data source for the evaluation of sub-queries, and with the Query Processor for returning sub-query results.

The queries supplied with transformations also provide the necessary information for these transformations to be *automatically reversible*: an `add / extend` transformation is reversed by a `delete / contract` transformation with the same arguments, while a `rename` transformation is reversed by another `rename` with the opposite ordering of arguments. This means that AutoMed is a *both-as-view* (BAV) data integration system: the `add / extend` steps correspond to Global-As-View (GAV) rules and the `delete / contract` steps correspond to Local-As-View (LAV) rules.

A *Graphical User Interface* (GUI) is provided by AutoMed from which schemas and transformation pathways can be displayed. Selecting a schema in the GUI gives the user access to the Query Processor and the ability to pose queries. It is also possible to wrap new data sources from the GUI by specifying the wrapper to be used and the connection details.

The overall architecture of AutoMed is illustrated in Figure 4.1.

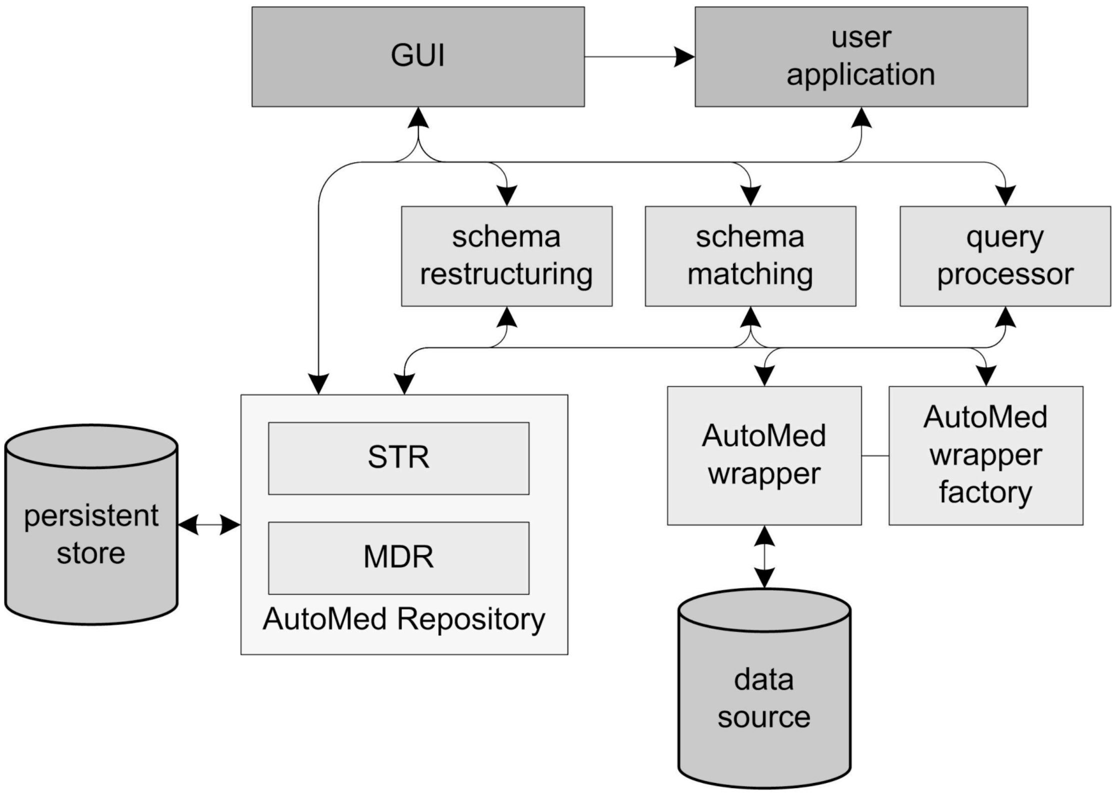


Figure 4.1 AutoMed Architecture

4.2 Extending AutoMed: Ontologies

In the semantic web, ontologies can describe the semantics of web data. In particular, *RDF* [Lassila, O. and Swick, R.R. 1999; McBride, B. and Hayes, P. 2002] has emerged as a standard for describing resources on the web, while *RDF Schema* [Brickley, D. and Guha, R.V. 2004] defines a type system for RDF. In order to make use of ontologies in ESTEST, we now describe how RDF and RDF Schema can be modelled in AutoMed.

4.2.1 RDF

We first summarise the characteristics of RDF which are relevant to its use in the ESTEST system. Resources on the web are identified by *Uniform Resource Identifiers (URI)* [Berners-Lee, T., Fielding, R. et al. 1998]. Properties of these resources are expressed as statements which take the form of `(subject, predicate, object)` triples. For example, the statement "Tony Blair is the Prime Minster of the United Kingdom" might be represented by the triple `(http://idcard.gov.uk/tblair346789,http://www.my-ont.com/ jobs/pm, http://www.my-ont.com/countries/uk)`.

Values in RDF can be URIs, literals or unlabelled nodes, known as "blank" nodes. Blank nodes can be used to structure other values e.g. by linking each line of an address. They can also represent concepts to which properties apply. Arbitrary identifiers are assigned to these blank nodes, and so they are analogous to object identifiers in object-oriented systems. There are also restrictions in the RDF data model concerning the kind of value each part of the triple can have, namely that 1) the subject can be a URI or a blank node, 2) the predicate must be a URI, and 3) the object can be a URI, blank node or literal.

It is possible in RDF to assert statements about statements e.g. "Reuters says that Tony Blair is Prime Minister of the UK". This is a fact about something that Reuters has said, not about Tony Blair's current job. In order to make a statement about this statement, it is necessary to remodel. This process of remodelling is known as *reification*. Reification in RDF happens in the following way. A blank node is created which represents the statement. This node is the subject of a triple with predicate `rdf:type` and object `rdf:statement`. The subject, property and object of the statement are now each the object of a new triple that has the new blank node as the subject and `rdf:subject`, `rdf:property` and `rdf:object` respectively as its predicate.

It is possible to illustrate sets of RDF statements as graphs, where by convention nodes are drawn as ovals and will be either labelled or blank, literals are drawn as rectangles, and edges as single-headed arrows. Figure 4.2 shows graphically a) the original RDF triple about Tony Blair's job and b) the reified form of this information.

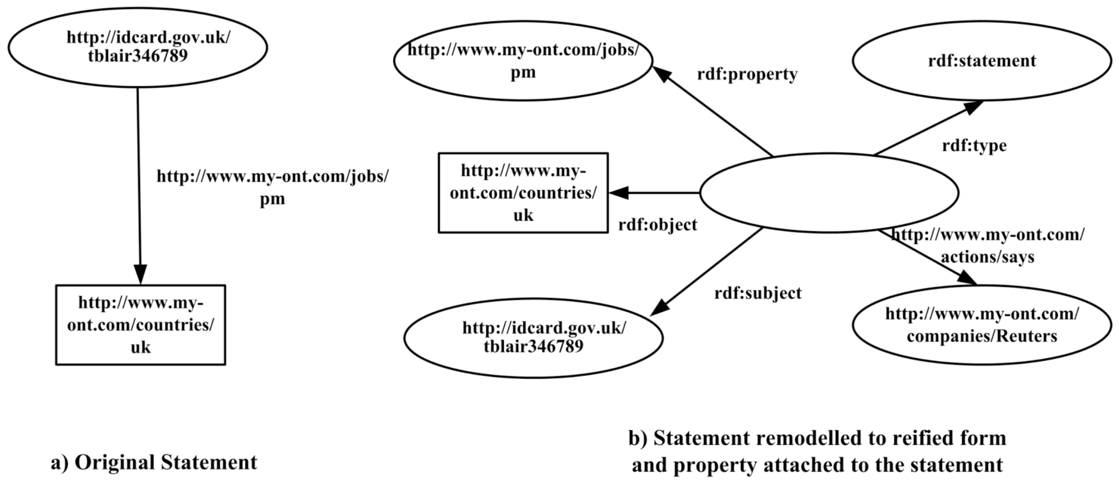

http://idcard.gov.uk/ tblair346789
http://www.my-ont.com/jobs/ pm
http://www.my-ont.com/countries/ uk
a) Original Statement
http://www.my-ont.com/jobs/ pm
rdf:property
rdf:statement
rdf:type
http://www.my-ont.com/countries/ uk
rdf:object
http://www.my-ont.com/ actions/says
rdf:subject
http://idcard.gov.uk/ tblair346789
http://www.my-ont.com/ companies/Reuters
b) Statement remodelled to reified form and property attached to the statement

Figure 4.2 RDF Triple and its Reified Equivalent

RDF has provision for three types of *container,* which are implemented in a similar way to reification. The container types are *Bag* — an unordered list with duplicates allowed, *Sequence* — an ordered list with duplicates allowed, and *Alternative* — a list of resources that represent alternatives for the single value of a property. A blank node represents the container itself, with the container type being indicated by a triple with the blank node as the subject, `rdf:type` as the predicate, and one of `rdf:Bag`, `rdf:Sequence` or `rdf:Alternative` as the object . Figure 4.3 shows the Tony Blair RDF triple with three associated containers: alternative telephone numbers, a bag with the names of his children, and a sequence showing his previous job history.

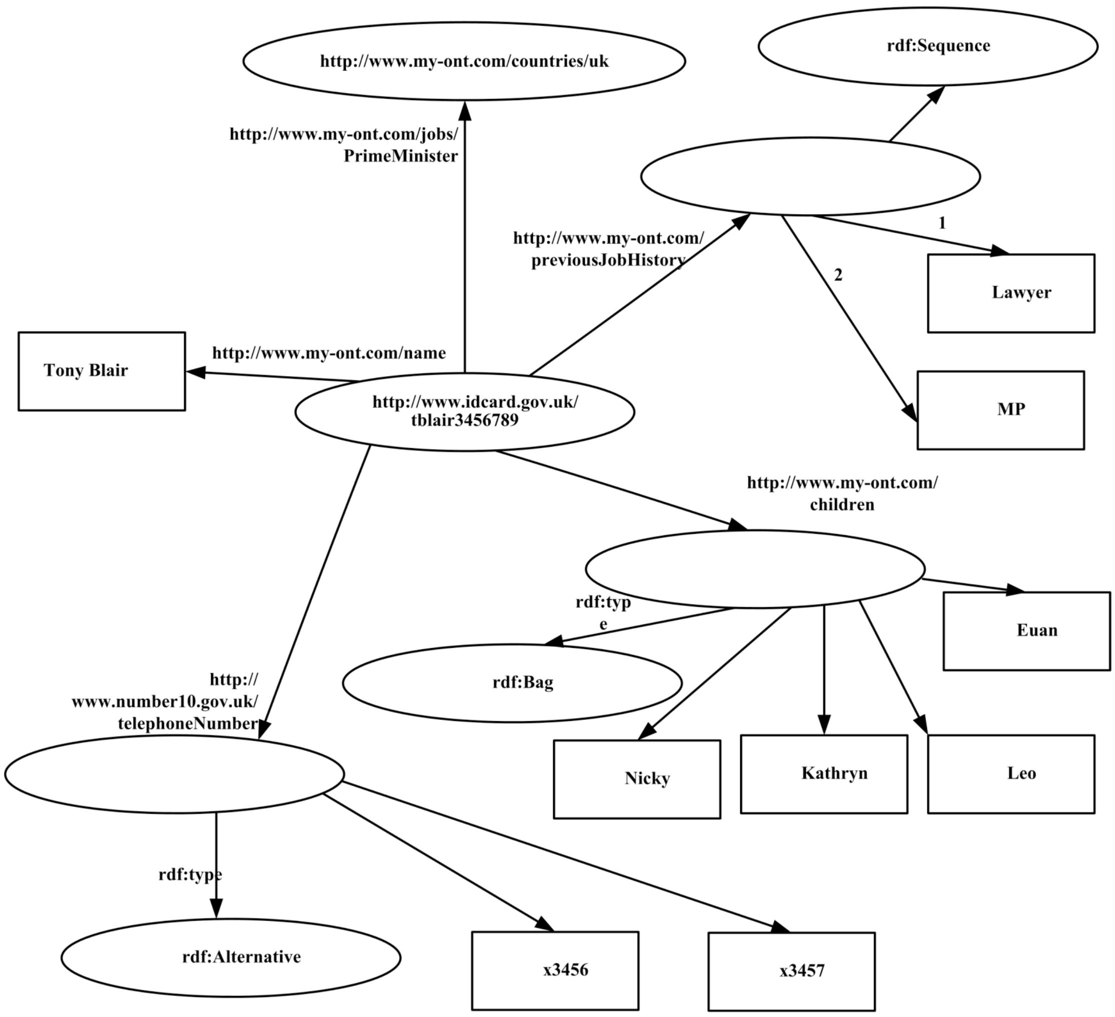

http://www.my-ont.com/countries/uk
rdf:Sequence
http://www.my-ont.com/jobs/PrimeMinister
http://www.my-ont.com/previousJobHistory
1
2
Lawyer
MP
Tony Blair
http://www.my-ont.com/name
http://www.idcard.gov.uk/tblair3456789
http://www.my-ont.com/children
rdf:type
Euan
rdf:Bag
http://www.number10.gov.uk/telephoneNumber
Nicky
Kathryn
Leo
rdf:type
rdf:Alternative
x3456
x3457

Figure 4.3 RDF Containers

4.2.2 RDF Schema

RDF is a general language for describing resources on the web. In order to define a domain in terms of a set of specific classes and properties, *RDF Schema (RDFS)* [Brickley, D. and Guha, R.V. 2004] can be used. RDFS provides a type system for RDF, comprising of classes and properties. These are similar to the classes and properties of object-orientated languages, with one key difference: every object in RDF is described as a resource and each resource has a type which defines it as being either a class or a property. Properties in RDFS are therefore defined independently of classes, unlike in object-oriented languages – that is, properties are not properties of any single class.

RDFS provides the resources listed in table 4.1 and the properties listed in table 4.2:

RDFS Resource	Description
`rdfs:Resource`	This is the base class for RDFS. Every object described by RDF is a resource and is an instance of the class `rdfs:Resource`.
`rdfs:Class`	This is used by the `rdf:type` property to specify that a resource is a class.
`rdfs:Property`	This is used by the `rdf:type` property to specify that a resource is a property.

Table 4.1 RDFS Resources

RDFS Property	Description
`rdf:type`	A property of a resource. If it has value `rdfs:Class` the resource denotes a class; if it has value `rdfs:Property` the resource is a property.
`rdfs:subClassOf`	Denotes that a class is a subclass of another. Each class can have many parent classes.
`rdfs:range`	Specifies a class whose instances contain the possible values that a property may take.
`rdfs:domain`	Specifies classes to which a property belongs. If no `rdfs:domain` is given, the property is assumed to apply to all classes.
`rdfs:subPropertyOf`	Specifies that a property is a sub-property of another property. Each property can have many parent properties.
`rdfs:seeAlso`	Additional information about the resource, referring to another resource.
`rdfs:isDefinedBy`	Sub-property of `rdfs:seeAlso` and indicates the resource defining the subject resource.

Table 4.2 RDFS Properties

The resources and properties of RDFS and the relationships between them are illustrated in Figure 4.4.

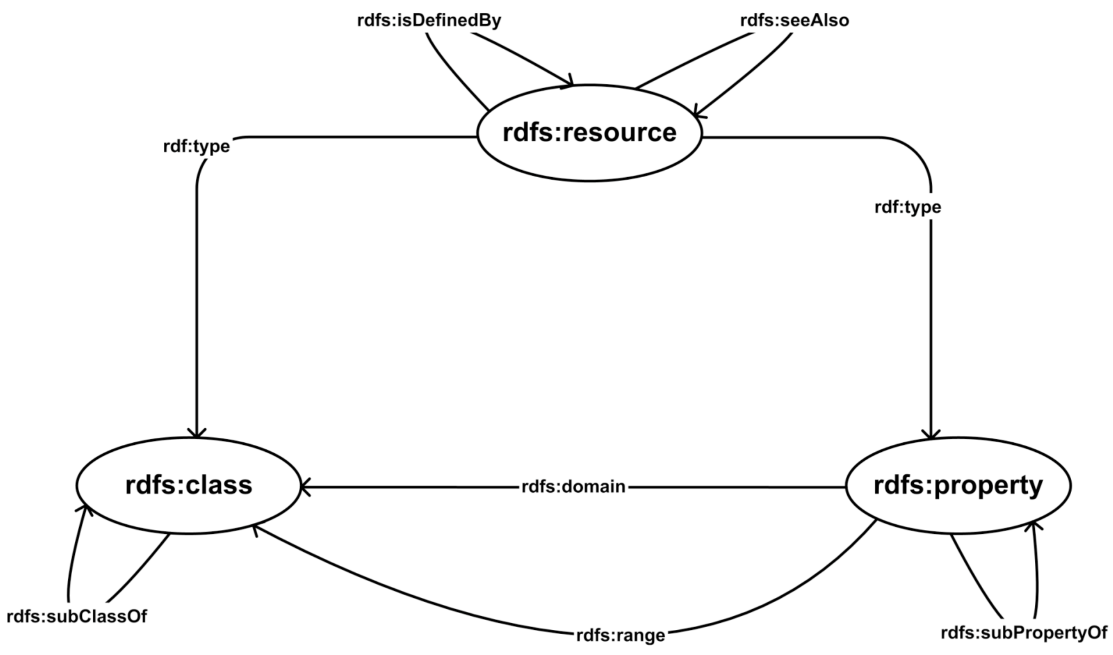


Figure 4.4 The Components of RDF Schema

4.2.3 Other Ontology Languages

RDF / RDFS has a number of limitations, for example, inability to represent disjoint classes or cardinality constraints. Proposals for more expressive formalisms have emerged, including DAML-ONT [McGuinness, D.L., Fikes, R. et al. 2003] and OIL [Fensel, D., Harmelen, F.v. et al. 2001], which were merged to produce DAML+OIL [Horrocks, I., Patel-Schneider, P.F. et al. 2002]. DAML+OIL itself is the basis for the subsequent OWL language [Horrocks, I. 2005]. OWL subsumes RDFS and comes in three versions, each at a different point along the trade-off between expressiveness and ease of use:

- *OWL Full* permits the use of all the RDF, RDFS and OWL primitives. The syntactic freedom provided includes the ability to alter the basic RDF or OWL primitives by applying them to each other. The power of the language means that there are no computational guarantees on processing an OWL Full document, and computations may fail to terminate.

- *OWL DL* is *decidable* and guarantees conclusions will be computed in finite time by placing restrictions on the use of some OWL language constructs. However, the time taken may be worse than exponential in relation to the number of triples.
- *OWL Lite* provides only a classification hierarchy and support for cardinality constraints, and will process in exponential time in relation to the number of triples.

OWL is emerging as a new standard for ontology development. Although in the current version of ESTEST we have only provided support for RDF/RDFS ontologies, it would be possible to extend ESTEST to support OWL as future work, and we have developed the system with this extensibility in mind.

Natural Language ontologies aim to represent one or more human languages. The most well known natural language ontology is WordNet [Fellbaum, C.E. 1998] which comprises over 80,000 nouns organised into a semantic net with 60,000 concepts. Each concept has a list of word forms that represent it (synonyms). Concepts are connected by hypernym (is-a), meronym (part-of) and antonym (opposite-of) relationships. Modifiers and verbs are also captured with appropriate relations.

4.2.4 Representing RDF and RDFS in the HDM

There was no requirement for ontology data sources in AutoMed before the development of ESTEST, so it was necessary to define the RDF and RDFS modelling languages in terms of AutoMed's HDM metamodel.

Every object in RDF is a *Resource* and is represented in the HDM by a nodal construct. Similarly, the RDF constructs `URI`, `Literal` and `Blank` are all nodal constructs, each with an additional constraint that their members must also be members of `<<Resource>>`. The RDF construct `Triple` is represented by a combination of three HDM nodes, an edge, and three constraints. This specification of RDF in the HDM is illustrated in table 4.3. For any given RDF description, the HDM nodes `<<URI>>`, `<<Literal>>` and `<<Blank>>` have as their extents the set of URIs, literals and blank nodes appearing in the description, respectively, while the HDM node `<<Triple>>` has as its extent the set of triples.

RDF Construct	HDM Representation
Construct: `RDFResource` class: `nodal` scheme: `<<Resource>>`	node: `<<Resource>>`
Construct: `RDFNode` class: `nodal, constraint` scheme: `<<URI>>`	node: `<<URI>>` constraint: `<<URI>>` ⊆ `<<Resource>>`
Construct: `RDFNode` class: `nodal, constraint` scheme: `<<Literal>>`	node: `<<Literal>>` constraint: `<<Literal>>` ⊆ `<<Resource>>`
Construct: `RDFNode` class: `nodal, constraint` scheme: `<<Blank>>`	node: `<<Blank>>` constraint: `<<Blank>>` ⊆ `<<Resource>>`

Construct: `RDFEdge`
class: `nodal, linking and constraint`
scheme: `<<Triple>>`
	node:
	`<<subject>>, <<predicate>>, <<object>>`
	
	edge:
	`<<Triple,subject,predicate,object>>`
	
	and three constraints:
	
	`<<subject>>` ⊆ (`<<URI>>` ∪ `<<Blank>>`)
	`<<predicate>>` ⊆ `<<URI>>`
	`<<object>>` ⊆
	(`<<URI>>` ∪ `<<Blank>>` ∪ `<<Literal>>`)

Table 4.3 RDF Constructs and their HDM Representation

To support reification, RDF Statements are also supported in our specification of RDF. A blank node is used to represent the statement and this is linked to the components of the statement by a (hyper)edge `<<Statement,Blank,subject, predicate,object>>`.

Containers are also supported: there is one HDM edge `<<bag,Blank,Resource>>` whose extent is all the container / member associations for bag containers, one HDM edge `<<sequence,Blank,Resource>>` whose extent is all the container/member associations for sequence containers and one HDM edge `<<alternative,Blank,Resource>>` whose extent is all the container/member associations for alternative containers.

Bag and sequence containers use an additional HDM node `<<Number>>` to model members' cardinality and ordering respectively. In particular, we use an additional HDM edge from `<<bag,Blank,Resource>>` to `<<Number>>` and an additional HDM edge from `<<sequence,Blank,Resource>>` to `<<Number>>`. For all instances of the edge `<<bag,Blank,Resource>>` there is an edge to an instance of `<<Number>>` indicating the cardinality of that member in that bag. Similarly, for all instances of the edge `<<sequence,Blank,Resource>>` there are one or more edges to an instance of `<<Number>>` indicating the positions of that member within that sequence. The specification of RDF containers and statements in terms of the HDM is shown in Table 4.4.

RDFS Construct	HDM Representation
construct: RDFStatement class: nodal and edge scheme: <<Statement>>	edge: `<<Statement,Blank,subject,predicate,object>>`
construct: `RDFContainer` class: `nodal, linking, edge` scheme: `<<t>>` where t = `bag, sequence` or `alternative`	node: `<<Number>>` edge: `<<t,Blank,Resource>>` if `t=bag` or `t=sequence` then edge: `<<_,<<t,Blank,Resource>>,<<Number>>>>`

Table 4.4 RDF Statements and Containers

Table 4.5 below specifies RDFS in the HDM. We see that there are two different modelling constructs: `RDFSNode` and `RDFSEdge`. Also given are the instances of these constructs i.e. the RDFS schema. A parser can make use of the RDFS schema to constrain the triples that can be stored.

RDFS Construct	HDM Representation
Construct: `RDFSNode` class: `nodal` scheme: `<<s>>` where s is `rdfs:Resource, rdfs:Property` or `rdfs:Class`	node: `<<s>>`

Construct: `RDFSEdge`
class: `linking` and `constraint`
scheme: `<<e>>`
where e is one of:
`rdfs:domain,rdfs:Property,rdfs:Class`
`rdfs:range,rdfs:Property,rdfs:Class`
`rdfs:subClass,rdfs:Class,rdfs:Class`
`rdfs:subProperty,rdfs:Property,rdfs:Property`
`rdfs:seeAlso,rdfs:Resource,rdfs:Resource,`
`rdfs:isDefinedBy,rdfs:Resource, rdfs:Resource`
	edge:`<<e>>`

Table 4.5 RDFS Constructs and their HDM Representation

4.2.5 Wrapping RDF Data Sources

A wrapper for RDF data sources has been developed, conforming to the AutoMed wrapper architecture. This RDF wrapper, and also the HDM wrapper described in Section 4.3.2 below, were among the first to be developed for AutoMed and contributed to the development of the generic AutoMed wrapper architecture.

In this architecture, for each data model represented in AutoMed, an implementation of the `AutoMedWrapperFactory` class exists. This contains details of how to represent the data model in terms of the HDM, and is able to construct `AutoMedWrapper` objects supporting open connections to data sources.

The `AutoMedWrapper` provides methods for accessing the data source. The Wrapper can be passed IQL queries either by programs using the Wrapper to access the data source directly or by the AutoMed Query Processor, and will return query results in the Abstract Syntax Graph (ASG) format of the Query Processor [Jasper, E. 2002].

The `RDFWrapperFactory` is therefore able to create a representation of the RDF and RDFS modelling languages in the AutoMed Model Definition Repository (MDR) the first time that an RDF / RDFS data source is accessed.

To identify an RDF / RDFS data source to AutoMed, the URLs for the location of the RDF and RDFS specifications are passed to the `RDFWrapperFactory`, which returns an `RDFWrapper` object for that data source. If no RDF / RDFS description exists in the AutoMed Schema & Transformations Repository for that data source, the `RDFWrapper` connects to and extracts the RDF / RDFS metadata in order to create the representation in the MDR.

The JENA API [McBride, B. 2002] is used in the `RDFWrapper` to access RDFS XML files. Using JENA in this way means that adding support for new ontology languages such as OWL in the future should be straightforward.

Programs to demonstrate the RDF/ RDFS wrapper in use are now part of the AutoMed standard distribution.

4.3 Extending AutoMed: the HDM Data Store

The ESTEST system requires a store for the data extracted by its Information Extraction process. At the time that this requirement arose, development of the AutoMed system was in its initial phases and the Wrappers and Query Processing API had not yet been developed. A native HDM data store was therefore developed to meet this requirement of storing instance data. The development of the HDM store highlighted a number of issues relating to instance data in the HDM, such as the need for a clear and concise notation for defining non-trivial schemas; this notation is used in a *command parser* to allow HDM database definition scripts to be processed and the corresponding schema to be built in the HDM data store. The wrapper developed for this HDM data store was the first Wrapper implemented in AutoMed toolkit and I was a member of the Wrapper Architecture design team.

The HDM data model has been outlined above in Section 4.1. We highlight the following characteristics of the model which influenced the design of the HDM data store.

- HDM schemas consist of nodes and edges e.g. `<<person>>` is a node and `<<worksIn,person,room>>` is an edge.
- Edges can link any number of other nodes and edges e.g. `<<address,houseNumber,road,town,postCode>>`
- Edges can be named or unnamed e.g. `<<_,person,room>>` or `<<worksIn,person,room>>`.
- Each component of an edge can be either a node or another edge e.g. `<<livesAt,person,<<address,houseNumber,road,postCode>>>>`
- Edge names are only unique for a given sequence of components e.g. it is possible to have both `<<worksIn,person,project>>` and `<<worksIn,person,room>`.
- Nodes have an associated data type e.g. `integer`, `string`.

To illustrate the use of instance data in the HDM, we consider a simple personnel database with the following HDM schema:

Nodes:

```
<<person>>,   <<project>>,   <<room>>,   <<houseNumber>>,
<<road>>, <<town>>, <<postCode>>
```

Edges :

```
<<worksIn,person,project>>, <<worksIn,person,room>>,
<<address,houseNumber,road,town,postCode>>
<<livesAt,person,
      <<address,houseNumber,road,town,postCode>>>>
```

This database is required to store the following instance data:

- Dean, Mat, Hao and Edgar are people
- Hao, Edgar and Dean work on the AutoMed project

- Dean and Mat work on the Tristarp project
- Mat and Dean sit in room BE
- Edgar and Hao sit in room BG
- Dean lives at 64 Northdown Street, London N1 9BS

We note that there are two edges named `worksIn` and that the second component in the edge `livesAt` is itself an edge. For the purposes of this example, it is assumed that the data type of all nodes is `string` except for `<<houseNumber>>` which is an integer. As double chevrons are used to indicate a node or edge, to distinguish an instance data tuple we enclose it in square brackets.

This database is used in the following sections as an example to illustrate the different methods available for interacting with the HDM store.

4.3.1 Implementation

The HDM data store is implemented in the same way as other AutoMed metadata repositories, such as the *Model Definitions Repository* (MDR) and the *Schema and Transformations Repository* (STR), that is, it is listed in AutoMed's configuration file as a DSR (Data Source Repository) implemented in Postgres. The schema of the HDM DSR is shown in Figure 4.5.

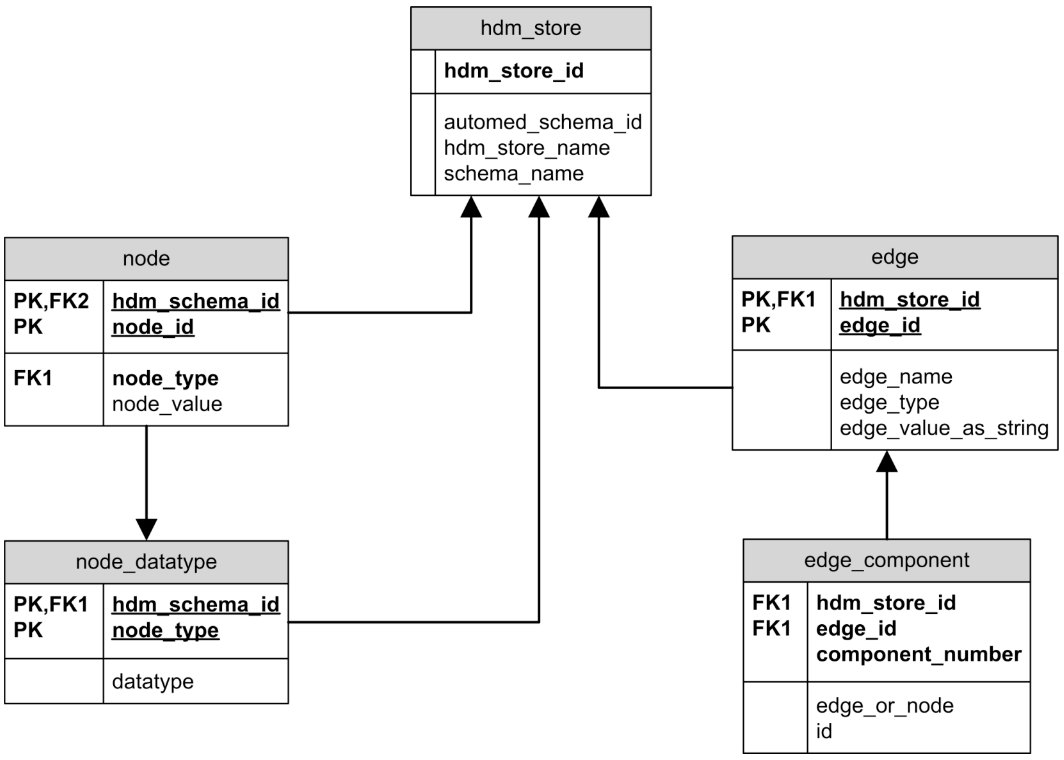


Figure 4.5 Schema of the HDM Data Store.

The `hdm_store` table allows multiple HDM data stores to be created, each linked to a schema in the STR. These data stores are identified by the `hdm_store_id`. The other tables are identified by a composition of this attribute and the logical identifiers of entities within the data store. This AutoMed schema is the only schema for HDM data stores and it is used to validate HDM instance data[5].

Instances of nodes are stored in the table `node`. The table `node_datatype` records the types of nodes with a data type other than the default string. The `edge` table stores instances of edges, each identified by a unique edge id. Edge instances have an associated `edge_type` (e.g. `<<worksIn,person,room>>`). The `edge_component` table stores each component of each edge. Each component can be either a node or another edge, and the `edge_or_node` attribute indicates if the `id` attribute references the `edge` or `node` table.

[5] This is unlike the general situation in AutoMed where data sources, such as relational databases, have some external schema and the corresponding AutoMed schema in the STR is a representation built for the purpose of accessing a specific data source.

As components of edges can themselves be edges, in order to fully retrieve an edge it may be necessary to recursively access the `edge` and `node` tables. Therefore, to assist debugging and for processing efficiency, a redundant string representation of an edge's components and the name of the edge are also recorded in the `edge` table, within the `edge_value_as_string` attribute.

4.3.2 HDM Data Store API & Wrapper

An API is available which allows HDM data stores to be created and updated through the `HdmStore class`. This class can create new, empty, data stores via the `hdmStore.createHdmStore()` method and make use of exsting data stores via the `hdmStore.use()` method. Data types can be assigned to nodes using the `hdmStore.setDatatype()` method. Nodes and edges can be added, retrieved and deleted. Instance data can be created through this API and should the node or edge not exist in the schema it will be added. As an illustration, the code for generating the example database described above is as follows:

```
HdmStore hdmStore = new HdmStore();
hdmStore.createHdmStore("personnel","bbkdata");
hdmStore.use("bbkdata");
hdmStore.setDatatype("HouseNumber","integer");
hdmStore.addNode("person","dean");
hdmStore.addNode("person","hao");
hdmStore.addNode("person","mat");
hdmStore.addNode("person","edgar");
hdmStore.addNode("room","N26");
hdmStore.addNode("room","B34E");
hdmStore.addNode("project","autoMed");
hdmStore.addNode("project","tristarp");
hdmStore.addNode("houseNumber","64");
hdmStore.addNode("road","Northdown Street");
hdmStore.addNode("town","London");
hdmStore.addNode("postcode","N1 9BS");

Edge edge = new Edge("<<worksin,person,project>>", new String[]
  {"dean","tristarp"});
hdmStore.addEdge(edge);

edge = new Edge("<<worksin,person,project>>", new String[]
  {"mat","tristarp"});
hdmStore.addEdge(edge);

edge = new Edge("<<worksin,person,project>>", new String[]
  {"hao","automed"});
hdmStore.addEdge(edge);
```

```
edge = new Edge("<<worksin,person,project>>", new String[]
  {"edgar","automed"});
hdmStore.addEdge(edge);

edge = new Edge("<<worksin,person,project>>", new String[]
  {"dean","automed"});
hdmStore.addEdge(edge);

edge = new Edge("<<worksin,person,room>>", new String[]
  {"mat","B34E"});
hdmStore.addEdge(edge);

edge = new Edge("<<worksin,person,room>>", new String[]
  {"dean","B34E"});
hdmStore.addEdge(edge);

edge = new Edge("<<worksin,person,room>>", new String[]
  {"hao","NG26"});
hdmStore.addEdge(edge);

edge = new Edge("<<worksin,person,room>>", new String[]
  {"edgar","NG26"});
hdmStore.addEdge(edge);

edge = new Edge(
 "<<address,houseNumber,road,town,postcode>>",
 new String[]{"64","Northdown Street","London", "N1 9BS"});
hdmStore.addEdge(edge);

edge = new Edge(
 "<<livesAt,person,<<address,houseNumber,road,town,postcode>>>>",
 new String[]{"dean","[64 Northdown Street, London, N1 9BS]"});
hdmStore.addEdge(edge);
```

This syntax is typical of AutoMed APIs, in that classes exist to represent the constructs of the data model used e.g. node and edge schema elements are created by passing a set of strings to a constructor. However, it differs from other AutoMed code in that instance data, as well as metadata, is created. In practice, for other data models the schemas will usually be imported automatically; while it is possible to hand code the definition of the schema of a relational database, it is much more usual to ask the wrapper to import the metadata via JDBC and create the AutoMed schema in that way. As no such native schema exists for HDM data sources, a more straightforward method of defining schemas and populating instance data is required and for this reason we developed the command parser described in Section 4.3.4 below.

An HDM wrapper is available with the same functionality as for the other AutoMed wrappers e.g. the RDF wrapper described above. This enables HDM data stores to be treated in the same way as any other data source by AutoMed. In particular, instance data can be created or amended by issuing an IQL statement through the Wrapper.

4.3.4 Command Parser

As an alternative to the HDM data store API, a command parser has been developed which takes a text file containing a sequence of commands and executes each command in turn. This removes the necessity to write new code each time an HDM data store is created or amended. If the text file of commands is being hand-crafted, then the syntax as described above would lead to lengthy files, even for reasonably small databases. For example, the commands for the personnel database described earlier would be:

```
createdb;
newstore personnel birkbeck;
use birkbeck;
settype houseNumber integer;
add <<person>>      [dean];
add <<person>>      [hao];
add <<person>>      [mat];
add <<person>>      [edgar];
add <<room>>        [NG26];
add <<room>>        [B34E];
add <<project>>     [automed];
add <<project>>     [tristarp];
add <<houseNumber>> [64];
add <<road>>        [Northdown Street];
add <<town>>        [London];
add <<postCode>>    [N1 9BS];
add <<worksIn,person,project>> [dean,tristarp];
add <<worksIn,person,project>> [mat,tristarp];
add <<worksIn,person,project>> [hao,automed];
add <<worksIn,person,project>> [edgar,automed];
add <<worksIn,person,project>> [dean,automed];
add <<worksIn,person,room>> [mat,B34E];
add <<worksIn,person,room>> [dean,B34E];
add <<worksIn,person,room>> [hao,NG26];
add <<worksIn,person,room>> [edgar,NG26];
add <<address,houseNumber,road,town,postCode>>
    [64,Northdown Street,London,N1 9BS];
add <<livesAt,person,
            <<address,houseNumber,road,town,postCode>>>>
    [dean,[64,Northdown Street,London,N1 9BS]];
```

Despite being a small database, this is quite unwieldy because the syntax is quite verbose especially where edges contain components that are themselves edges, although it is an improvement to writing code through the API. To make such definitions more manageable, a number of syntax shortcuts are provided which the parser converts into the full commands before execution:

1) Multiple instances of the same node can be specified in one command e.g. `add <<person>> [dean] [mat] [hao] [edgar].`

2) An optional additional parser command `addmissingnodes` instructs the parser to insert into the database any nodes mentioned in an edge definition which do not themselves currently exist in the database e.g. `add <<worksIn,person,project>> [dean,tristarp]` will add the `<<person>>` node `[dean]` and the `<<project>>` node `[tristarp]` if they are not already in the database.

3) As mentioned above, edge names do not have to be unique in a schema if the edges link different sequences of components. However, in practice they often will be and the parser allows for the edge name to be used and not the full type description in cases where there is not more than one edge with the same name e.g `add <<livesAt>> [dean,[64,Northdown Street,London,N1 9BS]]`instead of

```
add <<livesAt,person,
     <<address,houseNumber,road,town,postcode>>>>
     [dean,[64,Northdown Street,London,N1 9BS]].
```

4) Macros are available so that, once defined, later references to entries in the database can be referenced by a shorthand tag e.g. &1.

Using the above syntax shortcuts, the schema and instance data for the personnel database can be defined using just the following commands:

```
createdb;
newstore personnel birkbeck2;
```

```
use birkbeck2;
addmissingnodes;
settype houseNumber integer;
add <<worksIn,person,project>> [dean,tristarp] [mat,tristarp]
[hao,automed] [edgar,automed] [dean,automed];
add <<worksIn,person,room>>
[mat,B34E] [dean,B34E] [hao,NG26] [edgar,NG26];
add <<address>> [64,Northdown Street,London,N1 9BS] &1;
add <<address>> [12,Malet Street,London,WC1E 7HX] &2;
add <<livesAt>> [dean,&1];
add <<livesAt>> [edgar,&2];
```

4.4 Discussion

In this chapter we have summarised the facilities of the AutoMed data integration toolkit. We have also described two enhancements made to AutoMed in order to support the development of ESTEST, but which are more generally applicable.

As RDF/RDFS sources provide structural information, it was required for ESTEST to be able to accesses such data sources. More generally, RDF/RDFS is an area of focus for semantic web research, and so adding RDF and RDFS to the models supported by AutoMed is also a useful general extension to the toolkit. We have described a method for representing RDF and RDFS in AutoMed's HDM, including statements and containers. This has been implemented in the AutoMed toolkit, so that RDF and RDFS data sources can now be used in the same way as any other data sources, using an RDF wrapper that allows querying of the RDF triples that conform to the associated RDFS description.

For the purposes of ESTEST, such RDF triples describe an ontology and are of use as schema information, not instance data. In our discussion of the ESTEST integration step in Section 5.2 of Chapter 5, we will describe how the AutoMed-oriented representation of RDF data sources is transformed into an ESTEST-oriented schema.

ESTEST also requires a repository for the data extracted by its IE component, and a native HDM data store was developed to meet this need. This development led to the issues discussed in this chapter being considered, particularly the suitability of the HDM syntax for describing real data. The development of the HDM data store was driven by the fact that the ESTEST requirement for storing instance data arose before the development of the Wrapper and Query Processing functionality which now exists in AutoMed. However, even without this necessity, the development of a native HDM data store provides insights into the HDM data model as well as highlighting practical issues such as the verbose HDM syntax, especially for nested edges. The HDM data store has been developed as a stand-alone component in order for it to be reusable independently of ESTEST. As well as the code for the HDM data store itself, the AutoMed distribution now includes a number of demonstration programs for setting up HDM stores, and creating schemas and instance data, making use of both the API and the command parser.

Chapter 5

The Design of the ESTEST System

We have discussed in Chapters 1, 2 and 3, why, for the class of application addressed by this thesis, there are likely to be benefits in combining techniques from data integration and information extraction (IE), relating to integrating the available structured data and using it to partially configure and assist an IE process, and subsequently integrating the data and metadata extracted from the text with the existing structured data so as to support queries that would not be possible without this extension.

There are a number of interesting areas for research to focus on in combining these techniques, for example: automatic integration of extracted information into a structured database; how far the configuration time for an IE system can be reduced by the use of a schema to specify the entities to extract; and how such an approach compares to the manual methods presently employed in a particular domain. However, it would be difficult to investigate in depth any specific area without first having available a prototype end-to-end implementation of the proposed approach, and for this purpose we have designed and implemented an Experimental System To Extract Structure from Text (ESTEST), the design of which is described in this chapter.

In Chapter 6 we discuss the use of ESTEST in a specific application domain and present the results of experiments carried out. As a result of our experience of using this system in practice, a number of limitations were revealed in the area of automatic result integration, and further innovations to overcome these limitations are discussed in Chapter 7, along with a detailed investigation into two specific areas:

(i) the use of IE in schema matching, and (ii) the combination of NLP and database de-duplication techniques in order to resolve references to the same entity within both text and structured data.

5.1 The ESTEST Approach

In general, IE is used as a step in a sequence, normally to produce a structured dataset for further analysis. Our goal, in contrast, is to make the information extracted from text available to the query processing facility of a DBMS. As we have also argued, over time requirements for new queries may arise and new structured data resources may become available. Therefore, for this class of applications, a system must be able to handle incremental growth of its integrated schema and repeated application of its IE and integration functionality.

Our ESTEST system thus supports an evolutionary approach, allowing the user to iterate through a series of steps as new information sources and new query requirements arise. Each step may need to be repeated following amendment of the system configuration by the user. The overall process may also need to be restarted from any point. As a result, the components are built as independent modules which can be chained together to achieve the desired result and each can be re-run.

ESTEST makes use of the facilities of AutoMed for data integration, and of GATE for IE. Both of these systems are ongoing research projects, with research and development occurring in parallel to our own work. Extensions were required to both systems, and the design approach adopted has been to develop any enhancements likely to be more generally useful as components for those systems while any functionality specific to ESTEST has been implemented in separate ESTEST modules so as not to add complexity to the AutoMed and GATE API's. We discuss in Section 8.1 the challenges entailed in this choice of data integration and IE systems, and in Section 8.2.2 the possibility of using other data integration and IE systems within ESTEST.

We envisage that the ESTEST approach will ultimately be used by end-users within an integration and extraction workbench, making use of a graphical user interface (see Chapter 8). To facilitate this future work, we have designed ESTEST as a set of components that can be controlled either by scripts or through the envisaged GUI. The script functionality allows the specification of steps simulating user changes to the configuration, for example to change the schema elements for which word forms are automatically expanded. Scripts also make straightforward the re-running of experiments many times.

A functional overview of the main phases of the overall ESTEST processes is shown in Figure 5.1. In this chapter we describe each of these phases in detail, including the architecture of the corresponding ESTEST components and their interaction with AutoMed, GATE and other third-party software. The phases are Data Source Integration (described in Section 5.2), Semi-Automatically Configure IE (Section 5.3), IE Process (Section 5.4), Integrate Extracted Information (Section 5.6), and Query Global Schema and Enhance Schema (Section 5.7).

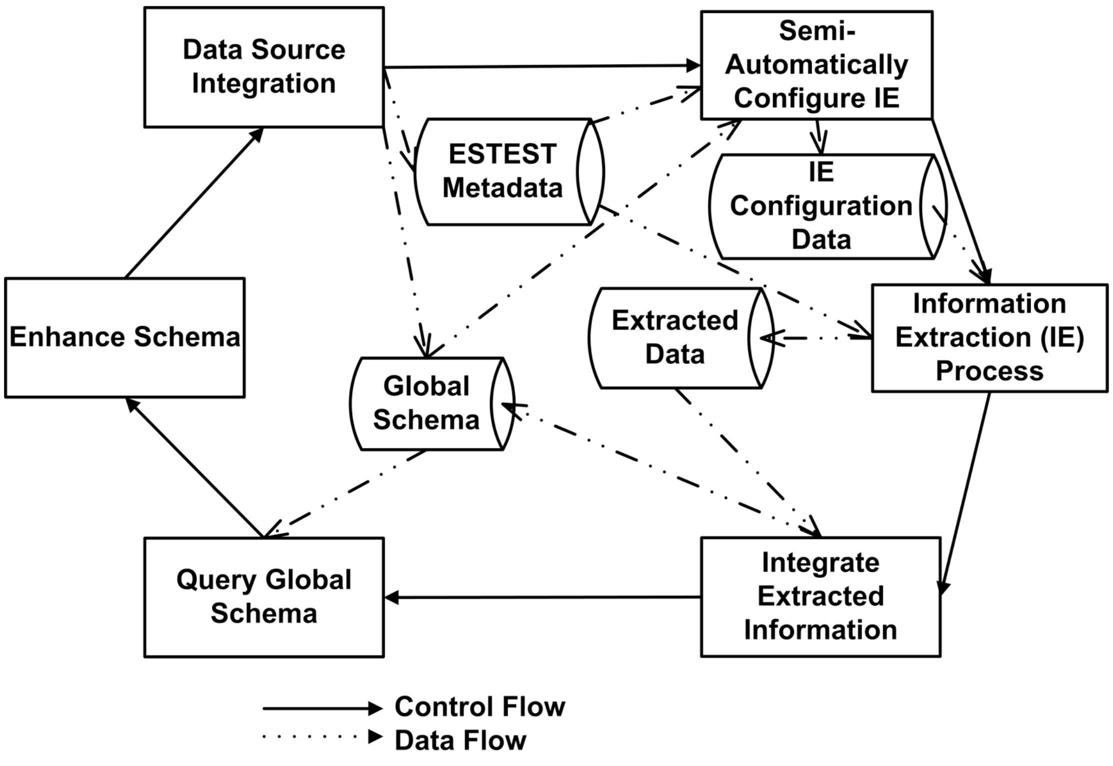


Figure 5.1 ESTEST Phases

5.2 Integrate Data Sources

In this section, we first give an overview of the integration process and then describe each of the steps in more detail. An integrated schema is first built from the variety of data sources available to the user: these may include structured databases, semi-structured XML data files, ontologies and sources of text for subsequent IE processing. ESTEST achieves this by first creating an AutoMed representation of each data source, via the appropriate AutoMed wrappers. Each of these schemas is then transformed into a schema expressed in the *ESTEST data model (EDM)*, which preserves the structural information needed by ESTEST. Transforming the data sources into a single data model makes it easier to reason about all data sources during the integration and for the ESTEST phases that follow. Metadata about schema constructs in each of the data sources is captured for use in merging schemas and also later in the IE step, for example textual descriptions and type information. Possible correspondences between constructs are suggested by a set of heuristics and the correspondences are confirmed or amended by the user. ESTEST uses these correspondences to merge the schemas and create a Global Schema. It is also one of our goals to automate as far as possible each part of the system, including the integrations, and we describe below steps where input is suggested for confirmation by the user, such as possible matches between schema elements from different data sources. ESTEST then makes use of the Global Schema to configure the IE process. New data may be found during the IE phase and this will be stored in a native HDM repository which has been integrated with the Global Schema.

Figure 5.2 shows the architecture of the ESTEST Integration Component and we now describe each step performed by this component in greater detail.

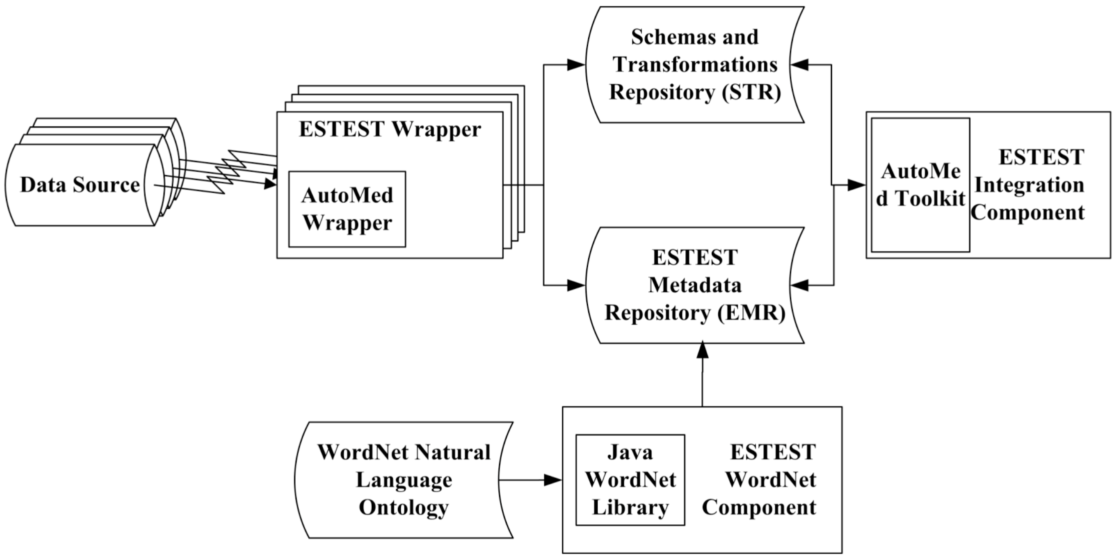

Schemas and Transformations Repository (STR)
Data Source
ESTEST Wrapper
AutoMed Wrapper
AutoMed Toolkit
ESTEST Integration Component
ESTEST Metadata Repository (EMR)
WordNet Natural Language Ontology
Java WordNet Library
ESTEST WordNet Component

Figure 5.2 ESTEST Integration Component Architecture

5.2.1 Wrapping of Data Sources

An ESTEST Wrapper exists for each data model that makes use of the corresponding AutoMed wrapper where possible and implements ESTEST-specific functionality where necessary e.g. collecting additional metadata such as textual descriptions and type information.

The data sources for an application may include structured databases, semi-structured data files, and domain ontologies. These can be organised according to any data model supported by an ESTEST Wrapper. Currently, relational databases, XML data files, ontologies (represented in RDF/RDFS) and HDM data stores are supported by ESTEST. Each such data source is made known to ESTEST with a name, the data model it conforms to and the relevant connection information e.g. the details required by JDBC for a relational data source.

For each such data source, the ESTEST Wrapper first calls on the corresponding AutoMed wrapper to create an initial representation of the schema in the AutoMed STR.

Each schema is then converted into the ESTEST data model (EDM). The table below shows the constructs of the EDM and their representation in the HDM. The EDM provides *concepts* which are used to represent any construct that has an extent i.e. instance data. Concepts are represented by HDM nodes e.g. `<<fox>>`, `<<animal>>`, and are structured into an isA hierarchy e.g. `<<isA,fox,animal>>`. Concepts can have *attributes* which are represented by an HDM node and an unnamed edge in the HDM e.g. an attribute to represent the number of legs an animal had, would be represented by a node `<<num_legs>>` and an edge `<<_,animal,num_legs>>`.

ESTEST Construct	HDM Representation
Construct: `Concept` class: `nodal` scheme: `<<c>>`	node: `<<c>>`
Construct: `Attribute` class: `nodal, linking` scheme: `<<c,a>>`	node: `<<a>>` edge:`<<_,c,a>>`
Construct: `isA` class: `constraint` scheme: `<<isA,c1,c2>>`	constraint: `<<c1>>` ⊆ `<<c2>>`

As the constraint functionality for AutoMed was not implemented at the time we were developing ESTEST, we have implemented this functionality instead by materialising the `isA` relationships using a sequence of `add`, `contract` and `rename` AutoMed transformations (the method by which this is achieved is described in Section 5.2.3).

The ESTEST wrapper for any data model takes the schema created by the AutoMed wrapper and transforms this, still within the AutoMed STR, into its equivalent EDM representation. For example, the ESTEST Relational Wrapper transforms an AutoMed relational schema into an EDM schema as shown below:

AutoMed Construct	ESTEST Representation
Model: `Relational` Construct: `Table` class: `nodal` scheme: `<<t>>`	concept: `<<t>>`
Model: `Relational` Construct: `Column` class: `nodal, linking` scheme: `<<t,a>>`	concept: `<<a>>` attribute: `<<t,a>>`
Model: `Relational` Construct: `Foreign Key` class: `constraint` scheme: `<<fky,a,t>>`	isA: `<<a,t>>`

From the point of view of a conventional data integration system, an RDFS schema and a set of RDF triples would be treated as a schema and its extent respectively – and this is how the AutoMed RDF/RDFS wrapper we have developed works. However, in the case of ESTEST this would mean that only the RDFS constructs would exist in the global schema, with the RDF triples being the extent of the corresponding RDFS constructs and accessible as data only via queries submitted to the AutoMed Wrapper. Therefore, the ESTEST RDF/RDFS Wrapper takes the representation created by the AutoMed RDF / RDFS Wrapper and converts the RDF triples into EDM schema information so the whole ontology is used for schema matching.

The ETEST RDF/RDFS Wrapper uses queries on the AutoMed RDF Wrapper to create representations of RDF/RDFS triples as EDM schema information:

- Each RDFS class is represented by an EDM concept e.g. `<<c>>`
- All RDFS sub-class relationships are represented by an `isA` edge between the two classes e.g. `<<isA,c1,c2>>`
- Properties in RDFS exist independently of classes. Therefore the following algorithm is used to represent RDFS properties as attributes in the EDM.

```
for each property:
    find the class which is its range
    for each class which is a domain for this property:
        create an EDM edge : <<attribute,domain,range>>
    endfor
endfor
```

- RDF triples are represented as an instance of the EDM `isA` relationship.

5.2.2 ESTEST Metadata Repository

The ESTEST wrappers collect metadata for use in the later phases of ESTEST and store this in the *ESTEST Metadata Repository (EMR)*. The metadata collected currently consists of word forms and type information. A *word form* is a word or phrase representing a concept. Word forms associated with concepts are of importance in ESTEST because of their use in the IE process where they will match strings contained in the text. The ambiguity of natural language means that word forms can be associated with many concepts.

ESTEST is able to collect word forms from a number of alternative sources: those manually entered by the user, from schema element names, from related concepts in the global schema's `isA` hierarchy, or from the WordNet natural language ontology [Fellbaum, C.E. 1998]. Each source has an associated confidence level so that, for example, manually entered word forms are given greater weight than words extracted from a schema element name.

Initially, word forms are collected from the data source metadata and any user input. On subsequent iterations of ESTEST, the user can expand the number of word forms associated with a concept should the IE process fail to find sufficient matches in the text. Word forms from these less precise sources are likely to increase recall but reduce precision.

The *ESTEST WordNet component* includes useful algorithms over the WordNet data, making use of the third-party Java WordNet Library [JWNL] to interact with the WordNet database. ESTEST uses WordNet to expand the available word forms by linking an ESTEST schema concept to a WordNet concept (user confirmation for this mapping may be required if there are multiple concepts matching the concept name) and expanding the word forms available by traversing the WordNet concept network and obtaining word forms from nearby concepts. As a default, the ESTEST WordNet component will keep searching for word forms until either 200 have been returned or 20 levels of hyponym relationship have been traversed. These default values have been arrived at during use of ESTEST in a number of domains; it is likely that the optimum default values will vary across domains and these can therefore be set as a parameter when ESTEST is loaded.

In naming database objects, abbreviations are important and they frequently occur in database schemas. These may be explicitly defined in a standards document or they may emerge through use e.g. “acc” is a frequently used abbreviation for “account” in financial systems and for “accident” in the Road Traffic Accident domain.

In addition to word forms, the ESTEST wrapper also gathers type information for subsequent use in schema matching, to suggest sources of named entities for IE, and to suggest sources of text to be processed by IE. The current AutoMed release collects some type information (though this was not the case when ESTEST was designed) but does not, on its own, meet the requirements of ESTEST. For example, the AutoMed relational wrapper will assign the same type information to both a short fixed-length character attribute and an unlimited length text attribute.

Textual descriptions from the source metadata (such as the metadata remarks supported by the JDBC database API) are also collected by the ESTEST Wrappers. In our experience, while it is rare for this feature of relational databases to be used in academic applications, in industry these are sometimes mandated to be completed and can also be populated by CASE tools and data dictionaries.

5.2.3 Initial Global Schema Creation

ESTEST uses the metadata in the EMR to suggest to the user correspondences between elements in the EDM representations of the data source schemas. This is achieved by comparing the word form and type information of each element in a schema with each of the elements in every other schema. ESTEST assigns a confidence measure to word forms depending on their source: word forms manually entered by the user have a higher confidence score than those mined from data source metadata, which in turn are preferred to word forms from WordNet. Using these confidence levels, ESTEST suggests the best match, providing the evidence crosses a threshold level. The user can accept or reject these suggestions as well as adding their own correspondences.

Using these correspondences, each of the EDM schemas is incrementally transformed into a *union schema* by means of a series of AutoMed primitive schema transformations. All the union schemas are syntactically identical and this is asserted by a series of `id` transformations between each pair of union schemas: `id` is a primitive AutoMed transformation that asserts the semantic equivalence of two syntactically identical constructs in two different schemas. The transformation pathway containing these `id` transformations is automatically generated by the AutoMed software. An arbitrary one of the union schemas is designated finally as the *global schema*.

ESTEST requires the ability to store the results found from its IE process and an additional data source is created for this purpose after the global schema has been created. This data source is stored in the native HDM repository that we have developed, and its schema is the HDM representation of the global schema. This new data source is integrated into the global schema by the automatic generation of the necessary AutoMed transformation pathways.

Figure 5.3 shows on overview of the network of schemas generated by ESTEST: schemas representing the data sources are created by the ESTEST wrappers, each of which itself wraps the equivalent standard AutoMed wrapper, and extends the functionality provided by directly connecting to the data source to obtain additional metadata. These schemas are converted by ESTEST into the EDM, and the schema matching process then finds correspondences between schema elements from across the different data sources. Once the corresponding schema elements have been renamed to show they are in fact the same, each data source schema is extended to contain the elements of all the schemas to be merged. These extended “union schemas” are shown to be equivalent by the id transformations that are asserted between them and any one can be chosen to be the global schema.

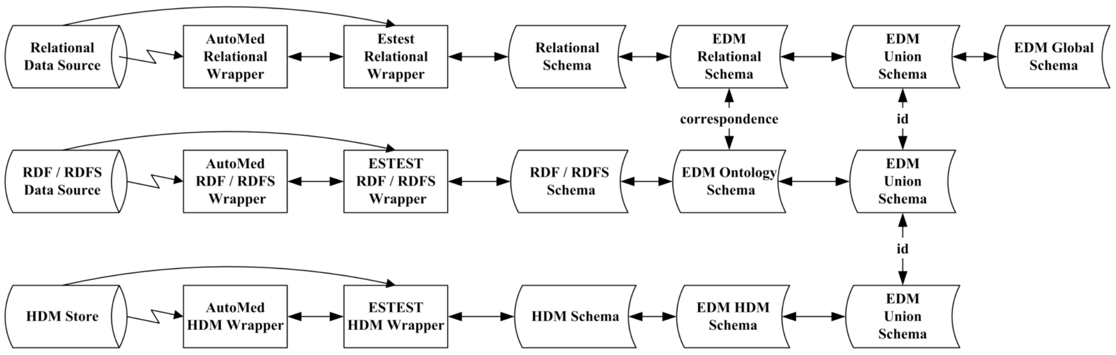


Figure 5.3 Schema Network Generated by ESTEST

Once the global schema is created, ESTEST now materialises the `isA` relationships. As the ESTEST wrappers create the EDM representation of each data source they maintain an array containing the `isA` edges created. This array is now used to examine the edges and produce expanded queries used in AutoMed to represent the extents associated with each schema element. For example if the schema contained elements `<<fox>>` and `<<animal>>`, and there is also an edge `<<isA,fox,animal>>`, then `<<animal>>` is replaced by a new schema element of the same name whose extent is defined to be the union of schema element `<<fox>>` and the original `<<animal>>` schema element. Our general method for undertaking these expanded extents is as follows:

```
for each schema element e in the global schema:
    if e is the 3rd component of an isA edge then:
        create a new schema object temp using an add transformation,
        passing as the query parameter the expanded extent query for e;
        for each edge edge in the global schema which has
        e as one of its components:
            add a new edge with component temp replacing e;
            apply a contract transformation to remove
            orig_edge from the global schema;
        apply a contract transformation to remove e from
        the global schema;
        apply a rename transformation to rename temp to e;
```

Querying the global schema for any schema element in the isA hierarchy now returns the expanded extent.

5.3 Semi-Automatically Configure IE

ESTEST's IE Configuration Component can now use the global schema and the additional metadata in the ESTEST metadata repository (EMR) to create configuration data for the IE process. This includes using the extent of concepts in the schema for named entity recognition, and using the schema and metadata in the EMR to suggest basic information extraction rules to the user and to create *templates* to be filled based on concepts in the schema which have missing attributes in the data.

Named entity recognition is central to IE, and the ESTEST configuration component suggests schema elements whose extents may be a set of entity names for use in IE by the SchemaGazetteer component (described in Section 5.4 below). These suggested schema elements can be amended by the user, who can also specify if the word forms associated with the named entity sources should be found from the schema metadata or if the word forms are defined by the extent of the schema element.

In Chapter 3 we described the limited generic support given by IE systems, to date, for producing structured data from the annotations resulting from the IE process. In Section 3.2 we gave an overview of the IE systems influenced by MUC and including Template Relation and Scenario Template tasks — however, these are hand coded for each specific relationship and scenario. More recently, semantic annotation has been used to populate ontologies [Popov, B., Kiryakov, A. et al. 2004] but other than limited support for predefined relationships, this is merely extending named entity recognition to move from a single annotation type to an annotation hierarchy.

To clarify the terminology in use, we observe that in GATE an *annotation schema* defines the annotation features that are valid for a particular annotation, e.g. a gender feature might be valid for a person annotation, but not for a location annotation. Annotation schemas are solely used to validate the manual entry of annotations through the GUI and not, for example, for validating annotations generated by JAPE rules. GATE annotation schemas are distinct from schemas in the database sense used by ESTEST.

In ESTEST, templates are automatically constructed from the global schema and consist of concepts and their attributes. Of particular interest are annotations in the text referring to an attribute in the template that have no value for a given instance of the template. If there are multiple possible annotations for a fragment of text, unfilled slots in a template that are known to be related to the text are preferred. For each concept and attribute, a stub GATE Jape pattern matching rule (see section 3.3) is defined. The user will expand these rule stubs to specify the text patterns that identify instances of the entity in the text.

5.4 Information Extraction Process

For this, a GATE pipeline is constructed consisting of 1) a standard GATE English tokeniser, 2) a GATE sentence splitter, 3) our `SchemaGazetteer` component configured to perform named entity recognition using the identified concepts in the schema, and 4) a GATE Jape transducer configured to use the Jape rules generated by ESTEST and enhanced by the user.

Named entity recognition for a specific entity type involves looking up tokens in the text being processed against lists of instances. In the standard IE approach, the entity types are not part of any type hierarchy. In contrast, our `SchemaGazetteer` component links the named entity annotations it generates over the text to elements in the global schema. It obtains the set of known instances of the entity from either a query to retrieve the extent of the global schema concept, or alternatively to obtain from the EMR the list of word forms associated with the concept.

5.5 Integrate Extracted Information

The final set of annotations over the text, produced by GATE, is now examined by ESTEST, and annotations that refer to new instances of schema concepts are extracted. GATE annotations have an associated "map" of features, each being a pair of the form `<attribute-name,attribute-value>` e.g. `{kind=employee}`. The Jape rules and the `SchemaGazetteer` configuration file which ESTEST generates make use of the standard GATE `kind` feature to identify annotations of interest by setting as the attribute value of this feature, the name of the schema element to which the annotation refers.

These annotations of interest are now stored automatically by ESTEST in the HDM store. As these annotations refer to a concept in the EDM global schema, they are represented by a node in the HDM. Therefore, for each such annotation, a corresponding instance of a node will be stored in the HDM data store e.g. `<<student>> [Dean Williams]`.

In the EDM, the concept of the instance being added may participate in a chain of `isA` relationships. These will result in a new node and a set of `isA` edges being added, moving recursively up the `isA` hierarchy e.g. `<<person>> [Dean Williams]` and associated edge `<<isA,person,student>> [Dean Williams, Dean Williams]`.

We have used the term *template* to refer to the situation where a concept has one or more related attributes — this is borrowed from IE where the template extraction task has the greatest similarity to ESTEST's generic approach to storing the results of IE; instances of attributes are said to be filling slots in an instance of a template.

A characteristic of partially structured data is that text from which the information has been extracted is itself an instance of a concept in the global schema which will itself be an attribute of some other concept e.g. if the document representing the minutes of a meeting in a university departments is processed by IE, these minutes may be represented in the global schema by an instance of the `<<minutes>>` concept, and there may also be an instance of the edge `<<attribute,department,minutes>>` representing the specific department in which these meeting minutes were taken.

Therefore should there be, in addition to the `<<attribute,department, minutes>>` edge, an edge `<<attribute,department,student>>`, then, when through IE an instance of `<<student>>` is extracted, it is assumed by ESTEST that this instance fills a slot in the same template as the earlier instance of `<<minutes>>`.

For example, if the document processed is: `<<attribute,department, minutes>> [Computer Science, Minutes of Computer Science staff / student meeting 1 April 2007 Dean Williams reported that……]`, then ESTEST will decide that the extracted student is from the Computer Science department and will store an edge `<<attribute,department,student>> [Computer Science, Dean Williams]`.

The text is not a separate and independent information source, unrelated to the schema of the structured data, but is itself an object in the global schema. This characteristic can be used to deduce relationships for structured data extracted from the text. The concept representing the overall text will appear in a template and will be an attribute of some other concept (for example, the column of a relational table which holds the text values being processed will be represented in the EDM as a concept which is an attribute of the concept representing the table). If the structured data extracted is for a concept that is an attribute in the same template as the whole text (for example, the extracted data is an instance of a different column in the same table), then the instance of the template which the extracted data refers to can be assumed to be the same template instance as that of the overall text being processed. For example, `<<attribute,department, student>> [Computer Science, Dean Williams]`.

Where this is not the case, that is, the extracted concept is not an attribute of the same concept as the overall text from which it was extracted, then ESTEST assumes that the extracted instance is new and creates unique identifiers (e.g. `ESTESTINSTANCE1),` as well as edges for attributes in templates.

This approach makes use of the global schema to automatically store the extracted information found by the IE process. This contrasts previous work on template extraction, reviewed in Section 3.2, and we are aware of no system which provides a generic approach to storing extracted data (note that in the case of the MUC and ACE competitions, details of the domain were released some time before the event and systems were tailored using sample data that was also provided in advance). Similarly, while IE researchers have made available a large number of IE components both in their own right, and distributed as part of a framework such as GATE and UIMA, we are not aware of any which provide a similar facility. The system with the closest goal to ESTEST in this regard is the KIM system described in Section 3.4 which assumes that all information extracted is an instance of a class in its pre-defined "ontology of everything".

While our approach described here is generic and original, when used in practice as described in Chapter 6, certain limitations become apparent when the relationships between the concepts extracted from the text are more complicated than those described above. In Chapter 7 we describe an extension made which uses database duplicate removal techniques to make use of the existing structured instance data in addition to the global schema metadata in order to provide a more flexible facility for dealing with integrating the extracted information.

5.6 Remaining ESTEST Phases

The user can now pose queries to the global schema the results of which will include the new data extracted from the text.

The global schema may subsequently be extended if a new data source is added, or if new schema constructs are identified and added to it, for example as new query requirements arise with respect to the global schema. We will see examples of this in the next chapter.

The user may also now choose to expand the number of word forms associated with schema concepts in order to increase the recall of the IE process. This can be done by entering word forms manually, or by obtaining more word forms from either the schema or WordNet as described in Section 5.2.2.

Following any such changes, the process described in Sections 5.2 – 5.6 is then repeated and new data is possibly extracted from the text. Because of this incremental approach to schema evolution and data extraction, we expect that a graphical workbench will ultimately be required for end-user use of ESTEST and the requirements of such a workbench are considered in Chapter 8.

5.7 Discussion

In this chapter we have described the design of the first version of the ESTEST system. This system demonstrates our approach end-to-end. Available structured data is integrated into a virtual global schema which is then used to assist in configuring an IE process. The extracted information is then automatically integrated into the global schema and is available to queries posed against this schema. Our approach and the ESTEST system have a number of novel features:

This is the first time, to our knowledge, that a heterogeneous data integration system has been extended to include support for data extracted from unstructured text.

IE systems require considerable effort to configure for each new domain. ESTEST makes use of the metadata extracted from the available domain-specific structured data sources, and from the WordNet natural language ontology, to semi-automatically configure the IE process.

In classic IE systems, no general purpose facilities were provided for subsequent processing of extracted annotations. Recent research has extended IE by using ontologies to provide a hierarchy of annotation types that are used in named entity recognition in the KIM system [Popov, B., Kiryakov, A. et al. 2004] discussed in Section 3.4. This has similarities with our own approach of using a virtual global schema and linking IE named entities to schema concepts. In both, named entity recognition is extended to go beyond linking an instance to one of a set of entity types, and instead to map the instance into a richer metadata structure: in our approach, the mapping is to a concept in the global schema constructed by the ESTEST system while in the KIM approach, it is to a entity in an ontology.

However, the KIM approach has as a prerequisite the existence of an 'ontology of everything'. In contrast, ESTEST adopts a pragmatic approach, developing the global schema from the available structured data sources specific to the application and seeking to expand the instance data and schema incrementally. ESTEST achieves this by adding to the previously known data and schema over time, by using the system to extract structured data from text and also by integrating new structured data sources into the virtual global schema as they become available. This approach is more readily applicable to the class of applications this thesis addresses and avoids the time consuming task of manual ontology creation by experts.

The ESTEST method of using the characteristics of partially structured data to automatically relate extracted instance data to the appropriate object in the global schema is analogous to the *template relation* and *scenario template* MUC tasks. However, whereas systems in the MUC competitions were hand-coded to fill templates representing the entities of interest, ESTEST's `SchemaGazetteer` component and the JAPE grammars it creates ensure that the annotations created by IE link back to concepts in the global schema and they are stored automatically.

In addition to describing partially structured data for the first time, [King, P. and Poulovassilis, A. 2000] suggests an approach for progressing research into this area consisting of i) developing graph-based representations capable of representing semantic and grammatical information in text, ii) developing a functional database programming language able to encode the data structures and extraction methods which would integrate the information from the text with the structured data, and iii) developing a workbench for processing textual fragments in order to generate semantic and grammatical information for the user to integrate into the structured data.

Our approach differs from this in focussing on combining existing structured and unstructured information management techniques in order to make use of the structured data to assist in the extraction of data from the text. In contrast, in [King, P. and Poulovassilis, A. 2000] the emphasis is on using NLP techniques in order to process the text and then integrating the extracted information with other previously known information; such an approach of incrementally processing the text and merging the results into the structured data has recently been demonstrated in [Chu, E., Baid, A. et al. 2007].

Our approach is similar to both [King, P. and Poulovassilis, A. 2000] and [Chu, E., Baid, A. et al. 2007] in that there is an emphasis on incrementally processing text. Like [King, P. and Poulovassilis, A. 2000], we see graph-based data models as being more suited to evolutionary expansion whereas [Chu, E., Baid, A. et al. 2007] argue that relational data models are sufficient – wrongly, in our view, given the limitations of dynamic schema evolution in relational databases. We differ from these two approaches in that we make use of the previously known structured information to assist in the extraction of new information from the text.

Chapter 6

Evaluation of the ESTEST System

Having described the design of the ESTEST system in Chapter 5, in this chapter we demonstrate its use in a specific application domain – UK Road Traffic Accident Reports. In Section 6.1 we outline the characteristics of this domain, focussing on the data collected and current limitations on its use. In Section 6.2 we demonstrate the use of ESTEST on this kind of data. In Section 6.3 we discuss experiments and present results performed on a collection of real road traffic accident reports. We give our concluding remarks in Section 6.4.

ESTEST is the first system that aims to support the requirements of partially structured data, and there is no other competing system to use in a comparison (as there would be if, for example, we were improving the recall and precision of named entity recognition). Therefore, our evaluation approach has been i) to demonstrate how the system would be of use in a real-world application domain that requires partially structured data, which we describe in Section 6.2, and ii) to show that the system is capable of producing similar results to a "vanilla" IE system while also being able to support queries that could not be supported by such an alternative.

For a complete evaluation of our approach, the development of a full end-user workbench would be required and we discuss such a possibility in Chapter 8.

6.1 The Road Traffic Accident Domain

In the UK, road traffic accidents (RTA) are reported using a format known as *STATS-19* which is defined in a specification called "STATS-20: Instructions for the Completion of Road Accident Report Form STATS19" [STATS20]. STATS-19 is a flat file format with multiple records per accident. In this file format, one header record exists for each accident, followed by one or more records for each person and vehicle involved in the accident. Such accident reports are collected by each of the UK's local police forces and are periodically sent to the UK Government Department of Transport for producing national statistics.

The bulk of the RTA schema consists of over seventy coded attributes. For example, the attribute 'road surface condition' has value 1 for 'Dry', 2 for 'Wet or Damp', and so on. For each attribute, the STATS-19 specification gives detailed guidance on the circumstances in which each of the codes should be used.

A textual description of the accident is also recorded in the accident report, expressed in a stylised form of English. An example textual description might be "FOX RAN INTO ROAD CAUSING V1 TO SWERVE VIOLENTLY AND LEAVE ROAD". Here, "V1" stands for "vehicle 1" which, by convention, is understood to be the vehicle which is thought to have caused the accident; the other vehicles involved are termed "V2", "V3" etc.

The schema of the structured part of the STATS-19 data is very comprehensive and there have been a number of revisions to it during its several decades of use. There are currently quinquennial reviews of the format to include new requirements. The format is also regularly the subject of questions and discussions by committees of the UK parliaments: for example, recently road safety campaigners lobbied for the inclusion of a code indicating when mobile phone use while driving may have been a cause of the accident.

However, there are still queries that cannot be answered via this schema alone and the textual descriptions need to be consulted in such cases. Also, anecdotal evidence given by police officers to road safety researchers indicates that the complexity of the forms to be filled in for each accident means that the quality of the information recorded varies greatly [Heydecker, B. 2005]. Indeed, when there is a discrepancy between the coded entries and the textual description, or the circumstances of an individual accident need to be reviewed, then the textual description is preferred to the coded entries.

6.2 An Example of ESTEST in use

We now show an example of the ESTEST system in use in a simple application within the RTA domain. We describe each of the steps undertaken with ESTEST and give the output from each step.

We assume three data sources are available:

1) `AccOnt` is a user-developed RDFS ontology concerning the type of obstructions that cause accidents. Figure 6.2 shows the `AccOnt` RDFS schema and some associated RDF triples.

2) `AccDB` is a relational database holding STATS-19 data from a police force. `AccDB` consists of the following tables:

```
accident(acc_ref,road,road_type, hazard_id, acc_desc)
vehicle(acc_ref,veh_no,veh_type)
carriageway_hazards(hazard_id, hazard_desc)
```

In the `accident` table, the `acc_ref` attribute uniquely identifies each accident, the `road` attribute identifies the road the accident occurred on, and `road_type` indicates the type of road. The `hazard_id` contains the `carriageway_hazards` code and this is a foreign key to the carriageway hazards table. We assume that the multiple lines of the text description of the accident have been concatenated into the `acc_desc` column. There may be zero, one or more vehicles associated with an accident and information about each of them is held in a row of the `vehicle` table. Here `veh_reg` uniquely identifies each vehicle involved in an accident and `acc_ref,veh_no` is the key of this table.

Below, we list the textual descriptions of the three accidents from `AccDB` that our example here refers to:

Accident	Description
A001234	FOX RUNS INTO NORTH GATE STREET CAUSING V1 TO SWERVE VIOLENTLY AND LEAVE ROAD OFFSIDE 50M AWAY
B231562	A50 WELFORD ROAD LEICESTER,BRIDGE 200 YDS S ALMOND WAY. V1 TRAV MOTORWAY M6 FAILS TO STOP AT XRDS AND HITS V2 TRAV UPPERTON RD V2 THEN HITS V3 PKD ON OS OF UPPERTON RD
C051633	ESCAPED KANGAROO JUMPS IN FRONT OF V1

3) `AccDBx` is a relational database of towns and roads and consists of the following tables:

```
towns(town)
roads(road,town)
```

The `towns` table contains a list of towns. In the `roads` table, the combination of `road` and `town` uniquely identify a road, the `town` attribute being a foreign key to the `town` table.

Our example is run from a script, with steps included which simulate user input. The script is listed in Appendix B. The output produced by ESTEST while running this script is also listed in Appendix B; relevant excerpts of output are included in our description below. The script comprises the following steps and we explain each of these in 6.2.1-6.2.12 below:

1) Initial configuration, relating to word abbreviations common to the domain, is loaded.

2) The data sources `AccDB`, `AccDBx` and `AccOnt` are made known to ESTEST and are integrated into a global schema.

3) The global schema is queried to determine what accidents were caused by animals – none are found at this stage.

4) ESTEST generates suggested schema elements suitable to be used as sources of named entities, and also suggests possible textual data sources for the IE process.

5) Changes to the IE configuration are loaded simulating user changes.

6) IE configuration rules and macros are generated.

7) IE is performed using these rules and macros.

8) The global schema is now queried again and now one accident (A001234) caused by an animal is found.

9) Further configuration changes are loaded, simulating a user decision to expand, from WordNet, the words associated with the schema concept `animal`.

10) IE configuration rules and macros are regenerated.

11) IE is performed.

12) The global schema is again queried to find accidents caused by animals and two (A001234 & C051633) are now found.

6.2.1 Initial Configuration is Loaded

Three abbreviations common to the domain are loaded e.g. “acc” as an abbreviation for “accident”:

```
Parameters to be loaded:
  Abbreviation of: accident, is: acc
  Abbreviation of: vehicle, is: veh
  Abbreviation of: description, is: desc
```

6.2.2 Integration of Data Sources

ESTEST now integrates the data sources using the approach described in Section 5.2. Firstly, the details of the data sources to be integrated are loaded. As mentioned in Section 5.2.1, the AutoMed schema representing any RDF data source is the same so this schema is created in the AutoMed STR to represent `AccOnt` by the AutoMed RDF wrapper as soon the connection is made to the RDF data source:

```
Loading datasources from definition at:C:\estest\bin\config\dsdx.
 xml
Building RDF modelling language
RDF Wrapper Factory creating RDF Model Oriented Schema accOnt sch
 ema
Details of schema: accOnt
  RDF subject          <<subject>>
  RDF predicate        <<predicate>>
  RDF object           <<object>>
  RDF triple            <<triple,subject,predicate,object>>
  RDF uri               <<uri>>
  RDF blank             <<blank>>
  RDF literal           <<literal>>


Data Sources To Be Integrated:
  DS 1 is accDB
  DS 2 is accDBx
  DS 3 is accOnt
```

The AutoMed relational wrapper is next used to create a schema for `AccDB` in the AutoMed STR:

```
Creating the AutoMed Schemas.
  RelationalDataSource is building a wrapper for schema accDBauto
  Created AutoMed schema for accDB
  Details of schema: accDBauto
    sql_390 table        <<vehicle>>
    sql_390 column       <<vehicle,acc_ref>>
    sql_390 column       <<vehicle,veh_no>>
    sql_390 column       <<vehicle,veh_reg_no>>
    sql_390 column       <<vehicle,veh_type>>
    sql_390 primary_key  <pky_vehicle,vehicle,<<vehicle,acc_ref>>
                         <<vehicle,veh_no>>>>
    sql_390 table        <<carriageway_hazards>>
    sql_390 column       <<carriageway_hazards,hazard_id>>
    sql_390 column       <<carriageway_hazards,hazard_desc>>
    sql_390 primary_key  <<pky_carr_hazard,carriageway_hazards,<<
                         carriageway_hazards,hazard_id>>>>
    sql_390 table        <<acc>>
    sql_390 column       <<acc,acc_ref>>
    sql_390 column       <<acc,year>>
    sql_390 column       <<acc,road>>
    sql_390 column       <<acc,road_type>>
    sql_390 column       <<acc,hazard_id>>
    sql_390 column       <<acc,acc_desc>>
    sql_390 primary_key  <<pky_accident,acc,<<acc,acc_ref>>>>
    sql_390 foreign_key  <fky_vehicle_accident,vehicle,<<vehicle,
                         cc_ref>>,acc,<<acc,acc_ref>>>>
    sql_390 foreign_key  <fky_accident_hazard,acc,<<acc,hazard_id
                         ,carriageway_hazards,<<carriageway_hazar
                         s,hazard_id>>>>
```

The AutoMed relational representation of `AccDB` can be viewed from the AutoMed GUI as illustrated in Figure 6.1.

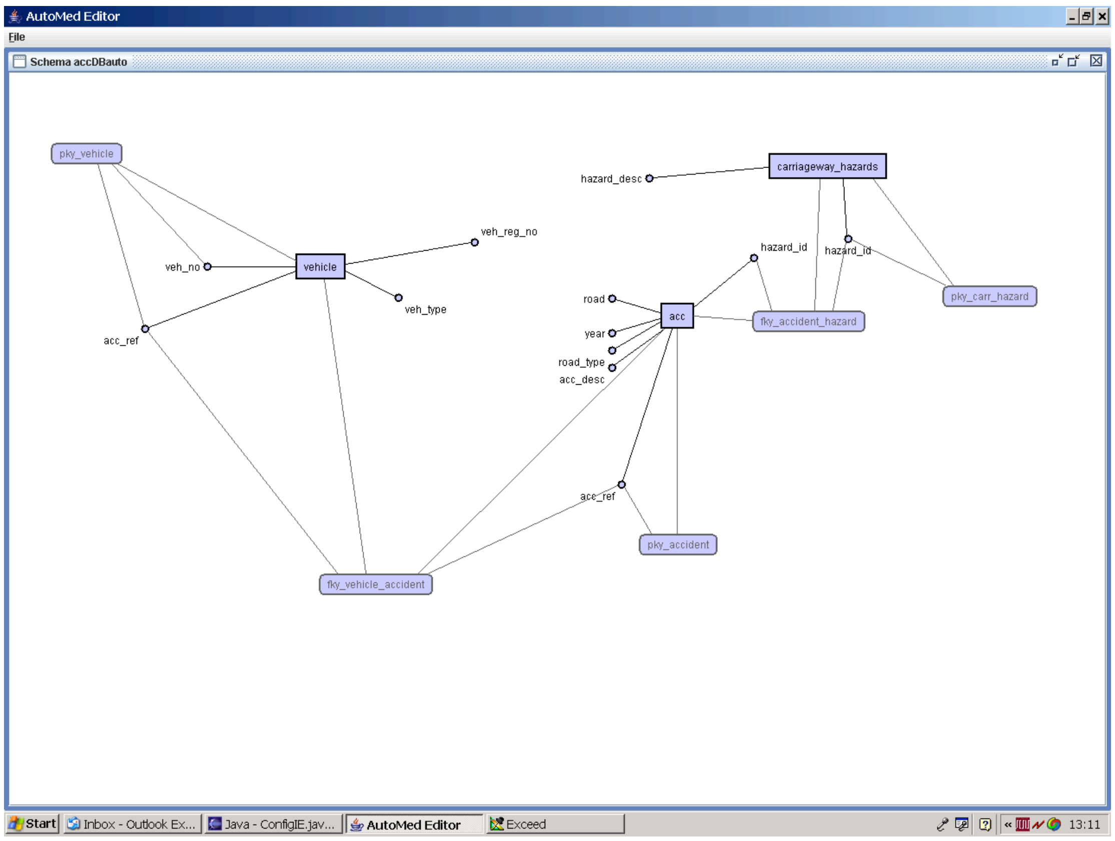


Figure 6.1 AutoMed representation of the AccDB data source.

Next, the AutoMed schema for `AccDBx` is created similarly:

```
RelationalDataSource is building a wrapper for schema accDBxauto
  Created AutoMed schema for accDBx
  Details of schema: accDBxauto
    sql_390 table         <<towns>>
    sql_390 column        <<towns,town>>
    sql_390 primary_key   <<pky_town,towns,<<towns,town>>>>
    sql_390 table         <<roads>>
    sql_390 column        <<roads,road>>
    sql_390 column        <<roads,town>>
    sql_390 primary_key   <<pky_road,roads,<<roads,road>>,<<roads,
                          town>>>>
    sql_390 foreign_key   <fky_accident_hazard,roads,<<roads,town>
                          ,towns,<<towns,town>>>>
```

Each of these three AutoMed schemas are now automatically transformed into the ESTEST data model, and the ESTEST schema is mined for word forms. The output from this process for `AccDB` is below. ESTEST first finds each foreign key, and adds a corresponding `isA` EDM construct (there are none for `AccDB`). Then, for each table and attribute, an EDM `concept` is created along with an `attribute` edge between the table and attribute. The relational constructs are then removed from the intermediate schema, leaving just the EDM representation. Finally, the word forms are obtained from the ESTEST schema and stored in the EMR:

```
Creating the ESTEST Model Schemas.
    Finding foreign keys (for isA relationship).
    Finding tables and columns (for concepts).
    Now delete the relational constructs.....
    The Estest oriented schema for this relational data source is
      accDBautzzh
    Now find word forms for each schema element.
    Schema element 'vehicle', word forms are : 'vehicle', 'veh'
    Schema element 'veh_no', word forms are : 'veh', 'vehicle',
      'veh no', 'vehicle no', 'no'
    Schema element 'veh_reg_no', word forms are : 'veh',
      'vehicle', 'veh reg', 'vehicle reg', 'reg', 'veh no',
      'vehicle no', 'veh_reg no', 'vehicle reg no', 'reg no',
      'no'
    Schema element 'veh_type', word forms are : 'veh', 'vehicle',
      'veh type', 'vehicle type', 'type'
    Schema element 'carriageway_hazards', word forms are :
      'carriage way', 'carriageway hazards', 'hazards'
    Schema element 'hazard_id', word forms are : 'hazard',
'hazard
      id', 'id'
    Schema element 'hazard_desc', word forms are : 'hazard',
      'hazard desc', 'desc', 'hazard description', 'description'
    Schema element 'acc', word forms are : 'acc', 'accident'
    Schema element 'acc_ref', word forms are : 'acc', 'accident',
     'acc ref', 'accident ref', 'ref'
    Schema element 'year', word forms are : 'year'
    Schema element 'road', word forms are : 'road'
    Schema element 'road_type', word forms are : 'road', 'road
      type', 'type'
    Schema element 'acc_desc', word forms are : 'acc',
'accident',
      'acc desc', 'accident desc', 'desc', 'acc description',
      'accident description', 'description'
    Storing Data Source info and metadata in EMR.....
  Created ESTEST schema for accDB
  Details of schema: accDBautzzh
    Estest concept      <<vehicle>>
    Estest concept      <<vehicle_veh_no>>
    Estest attribute    <<attribute,vehicle,vehicle_veh_no>>
    Estest concept      <<vehicle_veh_reg_no>>
    Estest attribute    <<attribute,vehicle,vehicle_veh_reg_no>>
    Estest concept      <<vehicle_veh_type>>
    Estest attribute    <<attribute,vehicle,vehicle_veh_type>>
    Estest concept      <<carriageway_hazards>>
    Estest concept      <<carriageway_hazards_hazard_id>>
    Estest attribute    <<attribute,carriageway_hazards,carriage
                        way_hazards_hazard_id>>
    Estest concept      <<carriageway_hazards_hazard_desc>>
    Estest attribute    <attribute,carriageway_hazards,carriagew
                        ay_hazards_hazard_desc>>
    Estest concept      <<acc>>
    Estest concept      <<acc_acc_ref>>
    Estest attribute    <<attribute,acc,acc_acc_ref>>
    Estest concept      <<acc_year>>
    Estest attribute    <<attribute,acc,acc_year>>
    Estest concept      <<acc_road>>
    Estest attribute     <<attribute,acc,acc_road>>
    Estest concept       <<acc_road_type>>
    Estest attribute     <<attribute,acc,acc_road_type>>
    Estest concept       <<acc_acc_desc>>
    Estest attribute     <<attribute,acc,acc_acc_desc>>
    Estest attribute     <<attribute,vehicle,acc>>
    Estest attribute     <<attribute,acc,carriageway_hazards>>
```

The EDM representation of the `AccDB` data source can be viewed from the AutoMed GUI as illustrated in Figure 6.2.

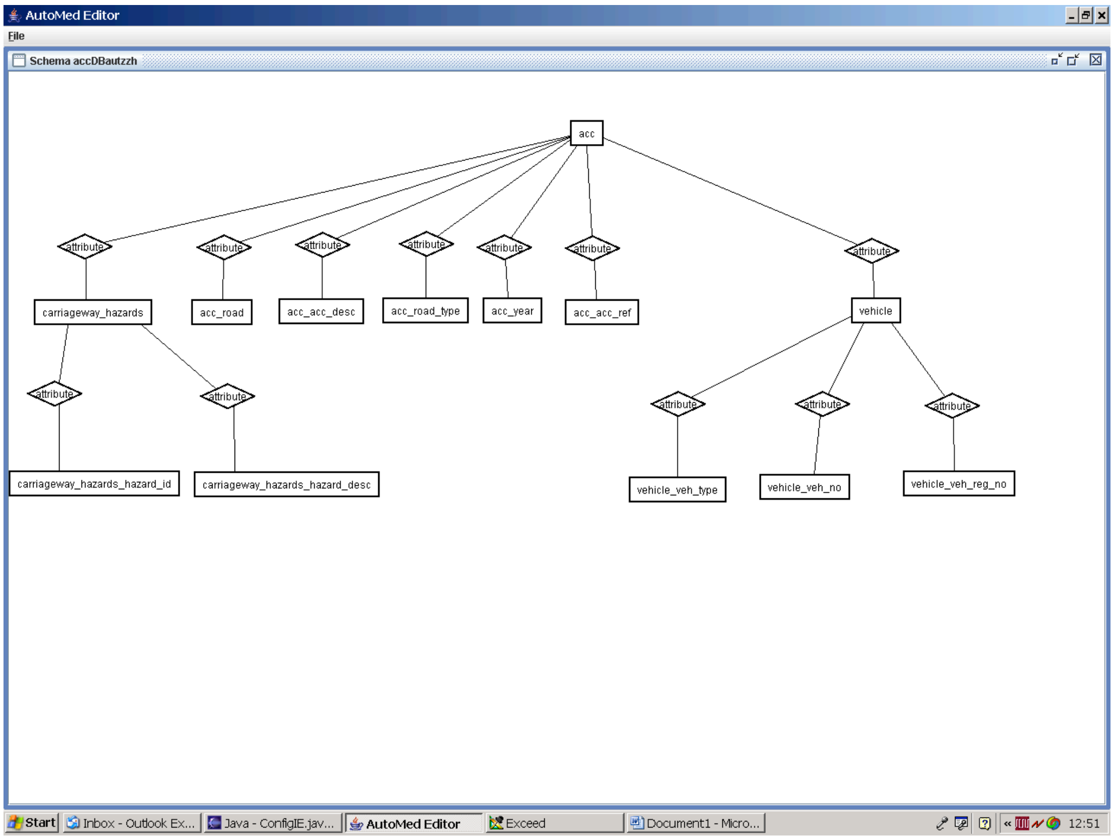


Figure 6.2 The EDM representation of the AccDB data source.

Next, the EDM schema for `AccDBx` is created similarly:

```
Finding foreign keys (for isA relationship).

    Finding tables and columns (for concepts).
    Now delete the relational constructs.....
    The Estest oriented schema for this relational data source is
     accDBxautzd
    Now find word forms for each schema element.
    Schema element 'towns', word forms are : 'towns'
    Schema element 'town', word forms are : 'town'
    Schema element 'roads', word forms are : 'roads'
    Schema element 'road', word forms are : 'road'
    Storing Data Source info and metadata in EMR.....
  Created ESTEST schema for accDBx
  Details of schema: accDBxautzd
    Estest concept      <<towns>>
    Estest concept      <<towns_town>>
    Estest attribute    <<attribute,towns,towns_town>>
    Estest concept      <<roads>>
    Estest concept      <<roads_road>>
    Estest attribute    <<attribute,roads,roads_road>>
    Estest attribute    <<attribute,roads,towns>>
```

Finally, the EDM schema for `AccOnt` is created:

```
OntologyDataSource is about to create ESTEST Schema.
  found class: http://www.dcs.bbk.ac.uk/~dean/kb/acc1#tree
  found class: http://www.dcs.bbk.ac.uk/~dean/kb/acc1#inanimate
  found class: http://www.dcs.bbk.ac.uk/~dean/kb/acc1#obstruction
  found class: http://www.dcs.bbk.ac.uk/~dean/kb/acc1#animal
  found class: http://www.dcs.bbk.ac.uk/~dean/kb/acc1#spillage
  found class: http://www.dcs.bbk.ac.uk/~dean/kb/acc1#accident
  About to check for word forms.
  Schema element 'tree', word forms are : 'tree'
  Schema element 'inanimate', word forms are : 'inanimate'
  Schema element 'obstruction', word forms are : 'obstruction'
  Schema element 'animal', word forms are : 'animal'
  Schema element 'spillage', word forms are : 'spillage'
  Schema element 'accident', word forms are : 'accident', 'acc'
  Schema element 'Resource', word forms are : 'Resource'
  Schema element 'bricks', word forms are : 'bricks'
  Schema element 'cat', word forms are : 'cat'
  Schema element 'fox', word forms are : 'fox'
  Schema element 'oak', word forms are : 'oak'
  Created ESTEST schema for accOnt
  Details of schema: accOntEstest
    Estest concept      <<tree>>
    Estest concept      <<inanimate>>
    Estest concept      <<obstruction>>
    Estest concept      <<animal>>
    Estest concept      <<spillage>>
    Estest concept      <<accident>>
    Estest concept      <<Resource>>
    Estest isA          <<isA,accident,Resource>>
    Estest isA          <<isA,spillage,inanimate>>
    Estest isA          <<isA,inanimate,obstruction>>
    Estest isA          <<isA,animal,obstruction>>
    Estest isA          <<isA,tree,inanimate>>
    Estest isA          <<isA,obstruction,Resource>>
    Estest attribute    <<attribute,accident,obstruction>>
    Estest concept      <<bricks>>
    Estest isA          <<isA,bricks,spillage>>
    Estest concept      <<cat>>
    Estest isA          <<isA,cat,animal>>
    Estest concept      <<fox>>
    Estest isA          <<isA,fox,animal>>
    Estest concept      <<oak>>
    Estest isA          <<isA,oak,tree>>
```

Figure 6.3 shows i) the RDFS schema for `AccOnt`, ii) the associated RDF instance data and iii) the EDM representation of `AccOnt`:

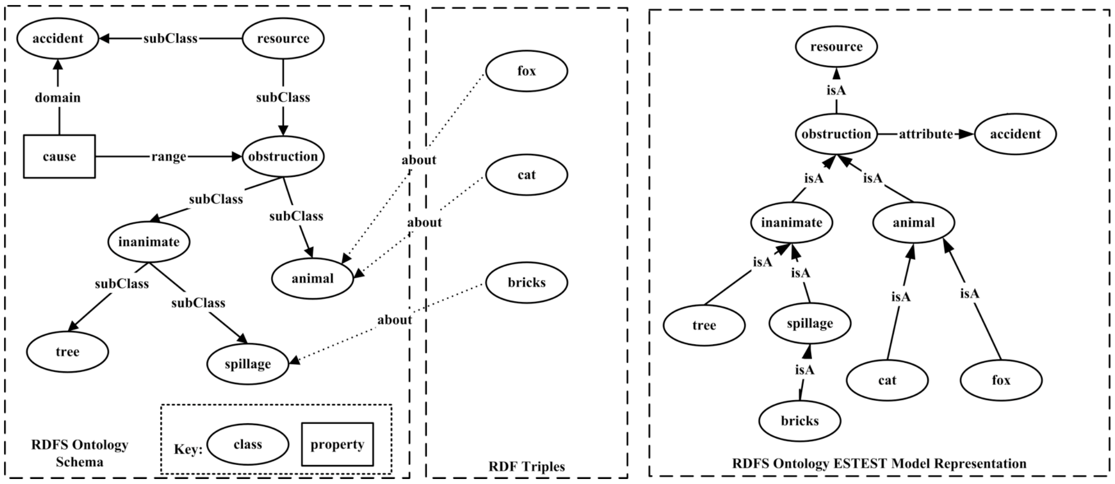


Figure 6.3 The AccOnt data source and its EDM representation

All three data sources are now represented within the AutoMed STR, and have also been transformed into their EDM representations. Now ESTEST attempts to find matches between concepts in the different schemas. Each concept in each schema is compared to each concept appearing in the other schemas. Where several possible matches are found, ESTEST suggests to the user the match with the highest confidence, provided this is over the predefined match threshold. In our example, two matches are found with confidence 0.64, one between `<<accident>>` in `AccOnt` and `<<acc>>` in `AccDB`, and another between `<<acc_road>>` in `AccDB` and `<<road>>` in `AccDBx`:

```
Going to find matches between Schema elements.
The matches are:
  Match with 0.64% confidence on word form ACCIDENT
    Schema 1: accDBautzzh, Concept 1: acc
    Schema 2: accOntEstest, Concept 2: accident
  Match with 0.64% confidence on word form ROAD
    Schema 1: accDBautzzh, Concept 1: acc_road
    Schema 2: accDBxautzd, Concept 2: roads_road
```

ESTEST can suggest the matches to the user and wait for confirmation and augmentation, but in this case the script dictates that matches should be used as suggested by the system, without waiting for confirmation. The integration is now done using the AutoMed `extendToMatch` method: given two schemas `s1` and `s2`, this outputs a new schema `s3` which is `s1` extended with any concepts from `s2` that do not appear in `s1`. We have implemented a generalised version of `extendToMatch` which takes an array of schemas and extends them all to match against each other, and which also overcomes some bugs in the AutoMed version. Although this functionality currently resides in ESTEST, in the future we plan to move it into the core AutoMed toolkit. The following output shows this matching process on the example:

```
Going to rename matching elemets so they have the same name.
In Integrator getPositionOfSchema looking for sid 89
  checking 89, accDBautzzh
  ........ and its a match
In Integrator getPositionOfSchema looking for sid 105
  checking 89, accDBautzzh
  checking 104, accDBxautzd
  checking 105, accOntEstest
  ........ and its a match
 Renamed :accOntEstest, accident  to acc
In Integrator getPositionOfSchema looking for sid 89
  checking 89, accDBautzzh
  ........ and its a match
In Integrator getPositionOfSchema looking for sid 104
  checking 89, accDBautzzh
  checking 104, accDBxautzd
  ........ and its a match
 Renamed :accDBxautzd, roads_road  to acc_road
Extend to match schemas.
 accDBautzzh extending to match accDBxautze
  new extended schema is :accDBautzzn
 accDBautzzn extending to match accOntEstest1a
  new extended schema is :accDBautzzzi
 accDBxautze extending to match accDBautzzh
  new extended schema is :accDBxautzzc
 accDBxautzzc extending to match accOntEstest1a
  new extended schema is :accDBxautzzx
 accOntEstest1a extending to match accDBautzzh
  new extended schema is :accOntEstest1y
 accOntEstest1y extending to match accDBxautze
  new extended schema is :accOntEstest1ze
Assert Identity Transformations between the extended schemas.
 Asserting ID transformation between accDBautzzzi & accDBxautzzx
 Asserting ID transformation between accDBautzzzi & accOntEstest1
 ze
```

Each of the schemas has now been extended to include missing concepts from the others. AutoMed `id` transformations are now automatically asserted between them and an arbitrary one of them is chosen as the global schema.

The HDM data store that will be used to store the results that ESTEST finds during the IE step is now created. In particular, an HDM schema containing each of the concepts in the current global schema is created in the AutoMed STR, as well as a transformation pathway from this HDM schema to its equivalent EDM schema. The EDM schema is finally linked to the global schema by asserting a series of `id` transformations, and the virtual integration is complete. The following output shows the creation of the HDM data store schema:

```
About to create HDM store copy of global schema.
Building the AutoMed HDM model
Creating HDM Store estest_store
Creating transormation pathway from HDM model to ESTEST model glo
 bal schema
Materialising isA relationships.
  Contents of the IsaFunctionList are:
    <<bricks>>        <<bricks>>
    <<spillage>>      <<spillage>> ++ <<bricks>>
    <<cat>>           <<cat>>
    <<animal>>        <<animal>> ++ <<cat>> ++ <<fox>>
    <<fox>>           <<fox>>
    <<oak>>           <<oak>>
    <<tree>>          <<tree>> ++ <<oak>>
    <<acc>>           <<acc>>
    <<Resource>>      <<Resource>> ++ <<acc>> ++ <<obstruction>>
                      ++ <<animal>> ++ <<cat>> ++ <<fox>> ++
                      <<inanimate>> ++ <<spillage>> ++ <<bricks>>
                      ++ <<tree>> ++ <<oak>>
    <<inanimate>>     <<inanimate>> ++ <<spillage>> ++ <<bricks>>
                      + <<tree>> ++ <<oak>>
    <<obstruction>>   <<obstruction>> ++ <<animal>> ++ <<cat>> ++
                      <fox>> ++ <<inanimate>> ++ <<spillage>> ++
                      <<bricks>> ++ <<tree>> ++ <<oak>>
  Integrator loadDef attempting ident with HDM Store
```

The global schema has now been created, and ESTEST outputs its EDM representation:

```
Global Schema is complete, schema name is: accDBautzzzi
  Details of schema: accDBautzzzi
    Estest concept       <<vehicle>>
    Estest concept       <<vehicle_veh_no>>
    Estest attribute     <<attribute,vehicle,vehicle_veh_no>>
    Estest concept       <<vehicle_veh_reg_no>>
    Estest attribute     <<attribute,vehicle,vehicle_veh_reg_no>>
```

```
     Estest concept       <<vehicle_veh_type>>
     Estest attribute     <<attribute,vehicle,vehicle_veh_type>>
     Estest concept       <<carriageway_hazards>>
     Estest concept       <<carriageway_hazards_hazard_id>>
     Estest attribute     <<attribute,carriageway_hazards,carriage
                          way_hazards_hazard_id>>
     Estest concept       <<carriageway_hazards_hazard_desc>>
     Estest attribute     <<attribute,carriageway_hazards,carriage
                          way_hazards_hazard_desc>>
     Estest concept       <<acc>>
     Estest concept       <<acc_acc_ref>>
     Estest attribute     <<attribute,acc,acc_acc_ref>>
     Estest concept       <<acc_year>>
     Estest attribute     <<attribute,acc,acc_year>>
     Estest concept       <<acc_road>>
     Estest attribute     <<attribute,acc,acc_road>>
     Estest concept       <<acc_road_type>>
     Estest attribute     <<attribute,acc,acc_road_type>>
     Estest concept       <<acc_acc_desc>>
     Estest attribute     <<attribute,acc,acc_acc_desc>>
     Estest attribute     <<attribute,vehicle,acc>>
     Estest attribute     <<attribute,acc,carriageway_hazards>>
     Estest concept       <<towns>>
     Estest concept       <<towns_town>>
     Estest attribute     <<attribute,towns,towns_town>>
     Estest concept       <<roads>>
     Estest attribute     <<attribute,roads,towns>>
     Estest attribute     <<attribute,roads,acc_road>>
     Estest concept       <<tree>>
     Estest concept       <<inanimate>>
     Estest concept       <<obstruction>>
     Estest concept       <<animal>>
     Estest concept       <<spillage>>
     Estest concept       <<Resource>>
     Estest isA           <<isA,spillage,inanimate>>
     Estest isA           <<isA,inanimate,obstruction>>
     Estest isA           <<isA,animal,obstruction>>
     Estest isA           <<isA,tree,inanimate>>
     Estest isA           <<isA,obstruction,Resource>>
     Estest concept       <<bricks>>
     Estest isA           <<isA,bricks,spillage>>
     Estest concept       <<cat>>
     Estest isA           <<isA,cat,animal>>
     Estest concept       <<fox>>
     Estest isA           <<isA,fox,animal>>
     Estest concept       <<oak>>
     Estest isA           <<isA,oak,tree>>
     Estest isA           <<isA,acc,Resource>>
     Estest attribute     <<attribute,acc,obstruction>>
```

The integrated global schema can be viewed from the AutoMed GUI as illustrated in Figure 6.4.

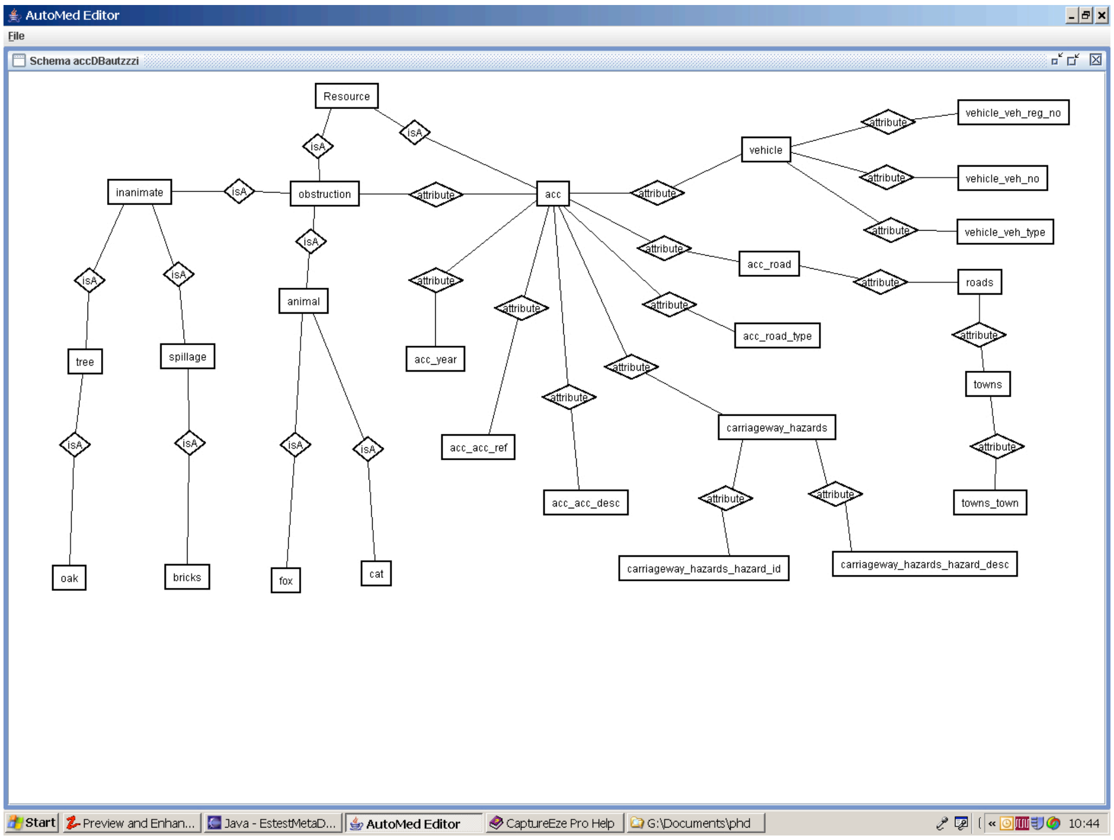


Figure 6.4 Global Schema.

6.2.3 Querying the Global Schema

The global schema can now be queried to see what obstructions caused accidents. The query is `<<attribute,acc,obstruction>>`. The AutoMed Query Processor translates this query into appropriate sub-queries for each data source and submits these to the AutoMed wrappers. The sub-queries are evaluated by the data sources, and the wrappers pass the results back to the Query Processor for merging and any necessary post-processing. In this example, the query posed has no matches in any of the data sources and so the empty list is returned:

```
About to run IQL query. Schema:accDBautzzzi, query:<<attribute,ac
 c,obstruction>>
Connecting to HdmStore for schema: estest_store
  The query was <<attribute,acc,obstruction>>
  Results: []
```

6.2.4 Configuration of Information Extraction Process

ESTEST next suggests configuration data for the Information Extraction process, based on simple heuristics as discussed in Chapter 5. The concepts `<<roads_road>>`, `<<towns_town>>`, `<<acc_acc_ref>>` and `<<acc_road_type>>` are suggested as sources of named entity descriptions based on their type characteristics i.e. reasonably small length character data. Templates to be filled are then identified, corresponding to concepts with attributes:

```
=================================================================
   CONFIGIE STEP (4)
=================================================================

Generating Suggestions for Named Entity.
Suggested possible NE List is:
  Possible Extent NE object - Data Source: 1, Schema
    Object<<acc_acc_ref>>
  Possible Extent NE object - Data Source: 1, Schema
    Object<<acc_road_type>>
  Possible Extent NE object - Data Source: 2, Schema
    Object<<towns_town>>
  Possible Extent NE object - Data Source: 2, Schema
    Object<<roads_road>>
Identifing Text Sources.
Finding Templates.
  Template: <<roads>>
    Attribute: <<towns>>
    Attribute: <<acc_road>>
  Template: <<acc>>
    Attribute: <<acc_acc_ref>>
    Attribute: <<acc_year>>
    Attribute: <<acc_road>>
    Attribute: <<acc_road_type>>
    Attribute: <<acc_acc_desc>>
    Attribute: <<carriageway_hazards>>
    Attribute: <<obstruction>>
  Template: <<vehicle>>
    Attribute: <<vehicle_veh_no>>
    Attribute: <<vehicle_veh_reg_no>>
    Attribute: <<vehicle_veh_type>>
    Attribute: <<acc>>
  Template: <<carriageway_hazards>>
    Attribute: <<carriageway_hazards_hazard_id>>
    Attribute: <<carriageway_hazards_hazard_desc>>
  Template: <<towns>>
    Attribute: <<towns_town>>
ESTEST is set NOT to wait for user confirmation of results.
```

6.2.5 Parameters are Loaded

To simulate user action following these suggestions, additional configuration specifications are now loaded via the example script. Three concepts are suggested as sources for named entity descriptions. For `<<acc_road>>` the set of named entities is based on its extent, while for `<<animal>>` and `<<obstruction>>` the set of named entitles is based on related word forms in the EMR. For `<<obstruction>>` it is also specified that the word forms should be expanded by making use of those associated with nearby concepts in the global schema:

```
Parameters to be loaded:
  Named Entity Parameter: animal is wordform based and schema
    expansion is selected
  Named Entity Parameter: acc_road is extent based and no
    expansion is selected
  Named Entity Parameter: obstruction is wordform based and
    schema expansion is selected
```

6.2.6 Information Extraction Configuration Generated

Using the above configuration parameters, the configuration information for the IE process is now generated. Additional word forms for the schema concepts are found e.g. ‘BRICKS’, ‘TREE’ etc for `<<obstruction>>`. JAPE rules are generated; for each named entity definition there is a macro and a lookup rule which can be amended by the user:

```
Expanding the selected Named Entity schema elements.
Expanding word forms from schema for <<animal>>
Expanding word forms from schema for <<obstruction>>
  OBSTRUCTION, OBSTRUCTION, INANIMATE, INANIMATE, SPILLAGE
  SPILLAGE, BRICKS, BRICKS, TREE, TREE, OAK, OAK, ANIMAL
Generating the Information Extraction JAPE input file
  Macro: acc_acc_ref
   ({Lookup.minorType == acc_acc_ref})


  Rule: acc_acc_ref
  (
  (acc_acc_ref)
  )
  :acc_acc_ref -->
   :acc_acc_ref.acc_acc_ref = {kind ="acc_acc_ref", rule = "acc_ac
   c_ref"}
```

```
Macro: acc_road_type
 ({Lookup.minorType == acc_road_type})


Rule: acc_road_type
(
(acc_road_type)
)
:acc_road_type -->
 :acc_road_type.acc_road_type = {kind ="acc_road_type", rule = "
 acc_road_type"}


Macro: towns_town
 ({Lookup.minorType == towns_town})


Rule: towns_town
(
(towns_town)
)
:towns_town -->
 :towns_town.towns_town = {kind ="towns_town", rule = "towns_tow
 n"}


Macro: roads_road
 ({Lookup.minorType == roads_road})


Rule: roads_road
(
(roads_road)
)
:roads_road -->
 :roads_road.roads_road = {kind ="roads_road", rule = "roads_roa
 d"}


Macro: animal
 ({Lookup.minorType == animal})


Rule: animal
(
(animal)
)
:animal -->
 :animal.animal = {kind ="animal", rule = "animal"}


Macro: acc_road
 ({Lookup.minorType == acc_road})


Rule: acc_road
(
(acc_road)
)
:acc_road -->
```

```
   :acc_road.acc_road = {kind ="acc_road", rule = "acc_road"}


  Macro: obstruction
   ({Lookup.minorType == obstruction})


  Rule: obstruction
  (
  (obstruction)
  )
  :obstruction -->
   :obstruction.obstruction = {kind ="obstruction", rule = "obstru
   ction"}


Created IE input file ie.jape
```

6.2.7 First Information Extraction Step

Now the Information Extraction process is run against each of the three accident reports in the database. Each of the GATE components is created and configured using the specifications described in the previous steps.

For the `SchemaGazeeter`, the schema elements used as named entity sources are listed, for example `accDBautzzzi:obstruction` (the global schema is `accDBautzzzi`) was specified as a word form based source and 10 word forms are loaded from the EMR to be used to find instances of obstructions in the text. `accDBautzzzi:acc_road` is also specified as a named entity source, but in this case the values used come from its extent rather than from the EMR and so the IQL query `distinct <<acc_road>>` is posed against the global schema to retrieve 5 values.

Once the GATE components are all initialised, a pipeline is created and the components added. Finally the text to be processed is loaded, the query `<<attribute,acc,acc_acc_desc>>` is posed against the global schema, and for each result returned a GATE document is created identified by the accident id.

```
Initialising Gate using Gate Home :C:\Program Files\GATE 3.0
Using C:\Program Files\GATE 3.0 as GATE home
Using C:\Program Files\GATE 3.0\plugins as installed plug-ins
directory.
Using C:\Program Files\GATE 3.0\gate.xml as site configuration
file.
```

```
Using C:\Documents and Settings\dean\gate.xml as user
configuration file
CREOLE plugin loaded: file:/C:/Program Files/GATE-
3.1b1/plugins/ANNIE/
Registering Creole directories:
 file:/C:/estest
CREOLE plugin loaded: file:/C:/estest/
Creating Default Tokeniser Gate Processing Resource.
Creating Sentence Splitter Gate Processing Resource.
Creating Database Gazetteer Gate Processing Resource.
  Configuring Database Gazetteer using file:/C:/estest/dbGaz.xml
  Named Entity source to be loaded is accDBautzzzi:obstruction
  Named Entity source to be loaded is accDBautzzzi:acc_road
  Loaded 10 values for word-form NE object <<obstruction>>
    About to run IQL query. Schema:accDBautzzzi, query:distinct
    <<acc_road>>
  Loaded 5 values for extent-based NE object <<acc_road>>
  Reading accDBautzzzi:obstruction
  Reading accDBautzzzi:acc_road
Creating Jape Transducer Gate Processing Resource.
JAPE URL: file:/C:/estest/ie.jape
Assembling Components Into Pipeline.
Gate is now initialised and the ESTEST application is built.
No JAPE URI specified - default will be used
Configuring Database Gazetteer using file:/C:/estest/dbGaz.xml
Named Entity source to be loaded is accDBautzzzi:obstruction
Named Entity source to be loaded is accDBautzzzi:acc_road
Loaded 10 values for word-form NE object <<obstruction>>
  About to run IQL query. Schema:accDBautzzzi, query:distinct
    <<acc_road>>
Loaded 5 values for extent-based NE object <<acc_road>>
Reading accDBautzzzi:obstruction
Reading accDBautzzzi:acc_road
About to run IQL query. Schema:accDBautzzzi, query:<<attribute,ac
 c,acc_acc_desc>>
Added doc FOX RUNS INTO ABBEY STREET CAUSING V1 TO SWERVE VIOLENT
 LY AND LEAVE ROAD OFFSIDE 50M AWAY
template instance is A001234
Added doc A50 WELFORD ROAD LEICESTER,BRIDGE 200 YDS S ROMAN WAY.
 V1 TRAV MOTORWAY M6 FAILS TO STOP AT XRDS AND HITS V2 TRAV UPPER
 TON RD V2 THEN HITS V3 PKD ON OS OF UPPERTON RD
template instance is B231562
Added doc ESCAPED KANGAROO JUMPS IN FRONT OF V1
template instance is C051633
```

Now that the IE pipeline has been built, and each component configured, each of the GATE documents is processed in turn in order to extract and store annotations:

```
----------------------------------------------------------------

Document to be processed by IE : 'FOX RUNS INTO ABBEY STREET CAUS
 ING V1 TO SWERVE VIOLENTLY AND LEAVE ROAD OFFSIDE 50M AWAY'
Running processing resources over corpus...

      Annotations:
   ......Annotation Set jape
Anotation type : obstruction rule/obstruction kind/obstruction
adding FOX/<<obstruction>>
```

```
Anotation type : acc_road rule/acc_road kind/acc_road
adding ABBEY STREET/<<acc_road>>
Annotation Details:
  Schema element = '<<obstruction>>', value = 'FOX' and the ID
    Will be generated.
  Schema element = '<<acc_road>>', value = 'ABBEY STREET' and the
    ID will be generated.
adding node to HDM Store with generated id <<fox>> [estestInstanc
 e1]
adding edge <<isA,fox,animal>>[A001234,estestInstance1]
adding edge <<isA,animal,obstruction>>[A001234,estestInstance1]
adding template attribute edge <<attribute,acc,obstruction>>[A001
 234,estestInstance1]
adding node to HDM Store with generated id <<acc_road>> [estestIn
 stance2]
adding template attribute edge <<attribute,acc,acc_road>>[A001234
 ,estestInstance2]

-----------------------------------------------------------------

Document to be processed by IE : 'A50 WELFORD ROAD LEICESTER,BRID
 GE 200 YDS S ROMAN WAY. V1 TRAV MOTORWAY M6 FAILS TO STOP AT XRD
 S AND HITS V2 TRAV UPPERTON RD V2 THEN HITS V3 PKD ON OS OF UPPE
 RTON RD'
Running processing resources over corpus...

      Annotations:
   ......Annotation Set jape
Annotation Details:

-----------------------------------------------------------------

Document to be processed by IE : 'ESCAPED KANGAROO JUMPS IN FRONT
  OF V1'
Running processing resources over corpus...

      Annotations:
   ......Annotation Set jape
Annotation Details:
```

We see that the first report above generates a match, as the report contains 'FOX' which is a word form related to `<<obstruction>>`, following the expansion of word forms from the schema and traversing the `<<isA,fox,animal>>` and `<<isA,animal,obstruction>>` relationships.

ESTEST creates a unique instance identifier for each newly extracted fact, one for the fox and one for the road. It then creates the following nodes and edges to be added to the HDM store:

```
<<fox>>                        [estestInstance1]
<<animal>>                     [estestInstance1]
<<obstruction>>                [estestInstance1]
<<isA,fox,animal>>             [estestInstance1, estestInstance1]
```

```
<<isA,animal,obstruction>>      [estestInstance1, estestInstance1]
<<attribute,acc,obstruction>>   [A001234, estestInstance1]
<<acc_road>>                    [estestInstance2]
<<attribute,acc,acc_road>>      [A001234,estestInstance2]
```

In the case of the fox, creating a new unique instance `estestInstance1` makes sense: the extent of `<<fox>>` now includes `[estestInstance1]`, indicating that the animal that caused the accident A001234 was a fox. However, this is less clear for the new unique instance of `<<road>>`, `[estestInstance2]`. Here, it may make more sense to store the string found in the road annotation as the identifier, rather than generate a new unique system identifier. A similar problem occurs when the values that should be stored are substrings of the annotation. For example, when looking for distances, the string that matches the distance annotation might be "approx 30 meters" but the value that should be stored is 30. Extending ESTEST's automatic processing of annotations extracted from text is discussed in Section 6.4 and in Chapter 7.

The second report above contains no mention of animals and the third refers to 'KANGAROO' which is not in the list of word forms associated with animals.

6.2.8 Second Query Step

The global schema is now queried again. Now, the extent of `<<attribute,acc, obstruction>>` is `{[A001234, estestInstance1]}`, showing that accident A001234 was caused by an animal, and this is returned by the query:

```
About to run IQL query. Schema:accDBautzzzi, query:<<attribute,ac
 c,obstruction>>
  The query was <<attribute,acc,obstruction>>
  Results: [{A001234 ,estestInstance1 }]
```

6.2.9 Parameters Loaded to Expand Animal Word Forms

Suppose now that the user suspects the results may not be complete and decides to expand the list of word forms associated with animals. WordNet is organised as a semantic net of synsets, that is a list of synonyms for each concept. The synset for the animal WordNet concept is 1780968 and so this is linked to the `<<animal>>` concept in ESTEST and the link used to find word forms from WordNet for `<<animal>>` (in the end-user workbench, we envisage that an interface to traverse through WordNet and select a synset would establish this link and could work in a similar way to the GUI provided with WordNet). `<<obstruction>>` is also selected for expansion from the meta-data, as `<<isA,animal,obstruction>>`, and then the new word forms obtained from WordNet for `<<animal>>` will be found.

```
ParameterDefintionContentHandler & its the end of a end of a syn
 set
offSetString is :1780968
offSet is :1780968
Parameters to be loaded:
  Synset Parameter: animal points to synset  1780968
  Named Entity Parameter: animal is wordform based and word net e
   xpansion is selected
  Named Entity Parameter: obstruction is wordform based and schem
   a expansion is selected
```

6.2.10 Configuration of Information Extraction Process

The configuration information for the IE process is now regenerated and additional word forms collected for `<<animal>>` from WordNet. The additional word forms for `<<animal>>` include possible candidates such as 'COW', 'STAG' but also a number of less likely road perils, not only the plausible but rare 'KANGAROO', but also 'SEA COW' and 'WATER RAT'. The word forms for `<<animal>>` are also associated with `<<obstruction>>` because of the `isA` relationship between the two concepts. The output is similar to that in 6.2.6 except for the longer list of animal word forms, which is now:

```
FEMALE MAMMAL, TUSKER, PROTOTHERIAN, METATHERIAN, PLACENTAL
PLACENTAL MAMMAL, EUTHERIAN, EUTHERIAN MAMMAL
FOSSORIAL MAMMAL, MONOTREME, EGG-LAYING MAMMAL, MARSUPIAL
POUCHED MAMMAL, LIVESTOCK, STOCK, FARM ANIMAL, BULL, COW
YEARLING, BUCK, DOE, INSECTIVORE, AQUATIC MAMMAL, CARNIVORE
FISSIPEDIA, AARDVARK, ANT BEAR, ANTEATER, ORYCTEROPUS AFER
BAT, CHIROPTERAN, LAGOMORPH, GNAWING MAMMAL, RODENT, GNAWER
GNAWING ANIMAL, UNGULATA, UNGULATE, HOOFED MAMMAL
UNGUICULATA, UNGUICULATE, UNGUICULATE MAMMAL, HYRAX, CONEY
CONY, DASSIE, DAS, PACHYDERM, EDENTATE, PANGOLIN
SCALY ANTEATER, ANTEATER, PRIMATE, TREE SHREW, FLYING LEMUR
FLYING CAT, COLUGO, PROBOSCIDEAN, PROBOSCIDIAN
PLANTIGRADE MAMMAL, DIGITIGRADE MAMMAL, NAKED MOLE RAT
DAMARALAND MOLE RAT, ECHIDNA, SPINY ANTEATER, ANTEATER
ECHIDNA, SPINY ANTEATER, ANTEATER, PLATYPUS, DUCKBILL
DUCKBILLED PLATYPUS, DUCK-BILLED PLATYPUS
ORNITHORHYNCHUS ANATINUS, OPOSSUM, POSSUM, OPOSSUM RAT
BANDICOOT, KANGAROO, PHALANGER, OPOSSUM, POSSUM, WOMBAT
DASYURID MARSUPIAL, DASYURID, POUCHED MOLE, MARSUPIAL MOLE
NOTORYCTUS TYPHLOPS, STAG, MOLE, SHREW, SHREWMOUSE, HEDGEHOG
ERINACEUS EUROPAEUS, ERINACEUS EUROPEAEUS, TENREC, TENDRAC
OTTER SHREW, POTAMOGALE, POTAMOGALE VELOX, CETACEAN
CETACEAN MAMMAL, BLOWER, SEA COW, SIRENIAN MAMMAL, SIRENIAN
PINNIPED MAMMAL, PINNIPED, PINNATIPED, FISSIPED MAMMAL
FISSIPED, CANINE, CANID, FELINE, FELID, BEAR, VIVERRINE
VIVERRINE MAMMAL, MUSTELINE MAMMAL, MUSTELID, MUSTELINE
PROCYONID, FRUIT BAT, MEGABAT, CARNIVOROUS BAT, MICROBAT
DUPLICIDENTATA, LEPORID, LEPORID MAMMAL, PIKA, MOUSE HARE
ROCK RABBIT, CONEY, CONY, MOUSE, RAT, MURINE, WATER RAT
NEW WORLD MOUSE, MUSKRAT, MUSQUASH, ONDATRA ZIBETHICA
ROUND-TAILED MUSKRAT, FLORIDA WATER RAT, NEOFIBER ALLENI
COTTON RAT, SIGMODON HISPIDUS, WOOD RAT, WOOD-RAT, HAMSTER
GERBIL, GERBILLE, LEMMING, PORCUPINE, HEDGEHOG
JUMPING MOUSE, JERBOA, DORMOUSE, SQUIRREL, PRAIRIE DOG
PRAIRIE MARMOT, MARMOT, BEAVER, MOUNTAIN BEAVER, SEWELLEL
APLODONTIA RUFA, CAVY, MARA, DOLICHOTIS PATAGONUM, CAPYBARA
CAPIBARA, HYDROCHOERUS HYDROCHAERIS, AGOUTI
DASYPROCTA AGUTI, PACA, CUNICULUS PACA, MOUNTAIN PACA, COYPU
NUTRIA, MYOCASTOR COYPUS, CHINCHILLA, CHINCHILLA LANIGER
MOUNTAIN CHINCHILLA, MOUNTAIN VISCACHA, VISCACHA
CHINCHILLON, LAGOSTOMUS MAXIMUS, ABROCOME, CHINCHILLA RAT
RAT CHINCHILLA, MOLE RAT, MOLE RAT, SAND RAT, DINOCERATE
ODD-TOED UNGULATE, PERISSODACTYL, PERISSODACTYL MAMMAL
EVEN-TOED UNGULATE, ARTIODACTYL, ARTIODACTYL MAMMAL
ROCK HYRAX, ROCK RABBIT, PROCAVIA CAPENSIS, ARMADILLO, SLOTH
TREE SLOTH, MEGATHERIAN, MEGATHERIID, MEGATHERIAN MAMMAL
MYLODONTID, MYLODON, ANTEATER, NEW WORLD ANTEATER, SIMIAN
APE, ANTHROPOID, HOMINOID, HOMINID, MONKEY, PROSIMIAN, LEMUR
TARSIER, PENTAIL, PEN-TAIL, PEN-TAILED TREE SHREW
CYNOCEPHALUS VARIEGATUS, ELEPHANT, MASTODON, MASTODONT
```

This expansion shows the benefits and difficulties of using WordNet to expand the word forms available in ESTEST for a concept. It can be seen that this new list contains some useful additions such as “BULL” and “COW’, as well as some which are plausible but unlikely to have cause road traffic accidents such as “MOUSE”, as well as a large number it is safe to assume have not ever featured on a STATS-19 report such as “LAGOSTOMUS MAXIMUS”. Using WordNet expansion is therefore likely to have the effect of increasing recall while lowering precision.

6.2.11 Second Information Extraction Step

The IE process is run again. ESTEST now finds a match for the fox in A00123 as before. In addition, as a larger list of animals is now recognised, it also finds a match for the kangaroo mentioned in report C051633:

```
Document to be processed by IE : 'ESCAPED KANGAROO JUMPS IN FRONT
  OF V1'
Running processing resources over corpus...

      Annotations:
   ......Annotation Set jape
Anotation type : obstruction rule/obstruction kind/obstruction
adding KANGAROO/<<obstruction>>
Annotation Details:
  Schema element = '<<obstruction>>', value = 'KANGAROO' and the
   ID will be generated.
adding node to HDM Store with generated id <<animal>> [estestInst
 ance5]
adding edge <<isA,animal,obstruction>>[C051633,estestInstance5]
adding template attribute edge <<attribute,acc,obstruction>>[C051
 633,estestInstance5]
```

6.2.12 Final Query Step

Querying the global schema now returns the complete set of results, since the extent of `<<attribute,acc,obstruction>>` is now `{[A001234, estestInstance1], [C0051633, estestInstance2]}` showing that both accident A001234 and C0051633 were caused by animals in the road:

```
About to run IQL query. Schema:accDBautzzzi, query:<<attribute,ac
 c,obstruction>>
  The query was <<attribute,acc,obstruction>>
  Results: [{A001234 ,estestInstance3 },{C051633 ,estestInstance5
   }]
Closing debug log file.
```

6.2.13 Limitations of the Example

This example is intended to demonstrate ESTEST working in a straightforward way rather than to be a real application or to make any claim about its performance. The rules generated would in a real application require manual enhancement. For example, if looking for animals responsible for accidents, reports such as “the fox ran in front of the car” should match while “the car swerved, went through a hedge and hit a cow” should not.

The next section gives details of an evaluation of ESTEST using real road traffic accident data.

6.3 Evaluation using Road Traffic Accident Data

There are a number of variables which affect the performance of ESTEST, including the availability of structured data sources relating to the text, the degree of similarity between the text instances, the amount of effort spent by the user in configuring the system to the specific application domain, and the domain expertise of the user.

In order to provide some initial confirmation of the potential of our approach we have experimented with six-months’ worth of Road Traffic Accident reports from one of Britain's 50 police forces, consisting of 1658 reports.

We identified a number of inconsistencies while setting up this data. The bulk of STATS-20 data is made up of one or two-digit numeric codes. The schema defines the allowable range of values for each of these codes and the meaning of each value. During set-up of the experiments, a number of places where codes were outside the allowable range of values were discovered. To investigate these issues, a Java program was written to check each code of each report against the allowable range of values from the schema – in cases where more than 3% of the reports had values outside this range, the code was assumed to be corrupt and was ignored. The 3% limit was chosen to allow for occasional coding errors. A significant number of codes failed this test, and we note that this test does not prove that the remaining codes actually do represent what the schema dictates, but rather that they are within the correct range of allowable values. A pattern emerged where blocks of seemingly correct values were interspersed with some additional incorrect ones.

We referred these findings to the RTA experts at the Centre for Transport studies at University College London who confirmed that the schema we used was the one believed to be correct and that the issues of data quality that we identified were valid.

To overcome this uncertainty in the data, we restricted our experiments below to codes that could reasonably be assumed to be correct by the fact that their value was in the allowable range for at least 97% of reports and, in addition, that there was an additional method of spot checking these numeric codes, for example checking for an unusual value that will be likely to be mentioned in the corresponding textual description, such as code 5 for road surface condition which represents a flood.

Within these limitations, we identified five queries of varying complexity that cannot be answered fully by the STATS-20 structured data alone. These queries are shown in the table below:

Query Num	**Query**
Q1	Which accidents involved red traffic lights?
Q2	How many accidents took place within 30-50m of a junction?
Q3	How many accidents involve drunk pedestrians
Q4	Which accidents were caused by animals?
Q5	How many resulted in a collision with a lamppost?

In order to provide a baseline we answered these queries in three ways i) using just the structured data, ii) using just IE, and iii) using the full ESTEST global schema and integrated data. To obtain the results from an IE-only process, only the data extracted by IE is used to answer these queries — that is, only the data within the HDM store, and not the relational RTA data. Similarly, to answer the queries using just the structured data, only the relational RTA data is queried and not the data extracted by IE and stored within the HDM. Finally, for iii), the full ESTEST global schema, integrating both the RTA and HDM data is used. In the case of Q1 and Q2, while the structured data is unable to directly support either query, an attribute does exist in the STATS-20 schema supporting a more general version of each query – we assume that in such cases the user would retrieve these more general results and then would manually examine the reports returned to exclude false positives (the alternative to this approach would have been to assume no results can be returned from the structured data).

The system was first configured (by the author) using a randomly chosen set of 300 reports from the full set of 1658 available reports. Configuring the system took 5 hours to review the 300 reports and to develop a domain-ontology and JAPE macros / rules for the IE process in order to cover the matches for each query found in the textual parts of the 300-report sample.

We then ran ESTEST over the remaining 1358 unseen reports, in each of the configurations i)-iii) described above, and we compared the results obtained to a subsequent manual examination of these reports. The results obtained for each query are now discussed in turn. The tables of results below give the actual number of relevant reports for each query (as determined by manual inspection), followed by the performance achieved by configurations i)-iii). In each case, the performance is shown by listing the number of reports identified, the number of these that were correct, and then showing this as Recall (the number of correctly identified reports as a percentage of all the correct reports) and Precision (the number of correctly identified reports as a percentage of all the identified reports).

6.3.1 Q1: Which accidents involved red traffic lights?

Attribute 1-17 in the STATS-19 format indicates what, if any, junction control was in place at the location of the accident. The possible values of this attribute are shown below:

Value	Meaning
1	Authorised person
2	Automatic traffic signal
3	Stop Sign
4	Give way sign or markings
5	Uncontrolled

Therefore, using the structured data alone under configuration i), accidents with a value of 2 will include those accidents involving red traffic lights, but will also include accidents where the traffic lights were green, or other automated traffic signals such as pelican crossings.

Regarding the textual parts of the reports, from the instances found in the 300 sample reports, the following IE rules were created to detect all occurrences of red traffic lights mentioned in these reports:

```
Macro: TRAFFIC
(
({Token.string == "TRAFFIC"} |
 {Token.string == "TRAFF"} |
 {Token.string == "SIGNAL"} |
 {Token.string == "TRAF"} )
)

Macro: LIGHT
(
({Token.string == "LIGHT"} |
 {Token.string == "SIGNAL"} |
 {Token.string == "SIGNALS"} |
 {Token.string == "LIGHTS"} )
)

Macro: SEP
(
 (SPACE) |
 {Token.string == "/"}
)

Macro: TL
(
 (
  ((TRAFFIC) (SEP)? (LIGHT)?) |
  (LIGHT) |
  ({Token.string == "R"} {Token.string == "/"}
   {Token.string == "FILTER"}) |
  ({Token.string == "T"} {Token.string == "/"}
   {Token.string == "LIGHT"}) |
  ({Token.string == "T"} {Token.string == "/"}
   {Token.string == "LIGHTS"}) |
    ({Token.string == "T"} {Token.string == "/"}
     {Token.string == "L"})
  )
)

Rule: traffic
// e.g. RED T/LIGHT or LIGHTS AT RED
(
  (({Token.string == "RED"}) (SPACE) (TL)) |
  ((TL) (SPACE) ({Token.string == "AT"})? (SPACE)
   ({Token.string == "RED"})  )
)
:traffic -->
    :traffic.traffic = {kind = "traffic", rule = "traffic"}
```

The results after running the system, with the three different configurations i)-iii), on the 1358 remaining reports produced the following results:

Found by Manual Inspection		26
Structured Data Only	Reports Found	120
	Number of correct reports found	22
	Recall	85%
	Precision	18%
IE Data Only	Reports Found	21
	Number of correct reports found	19
	Recall	73%
	Precision	91%
Combined Structured and IE Data	Reports Found	19
	Number of correct reports found	19
	Recall	73%
	Precision	100%

It can be seen that using the Structured Data only results in high recall but very low precision, whereas using the IE Data only results in lower recall but very high precision. Combing the two data sources in ESTEST meant looking only for those reports with both a junction control value of 2 and a mention of red traffic lights within the text. This removed the two false positives from the IE Data only results (both of these mentioned red traffic lights which were not involved in the accident but instead in the time preceding it i.e. the lights had changed from red, so while red traffic lights were mentioned it was the fact that they had turned green which contributed to the accident), giving the same recall as the IE results but with 100% precision.

6.3.2 Q2: How many accidents took place 30-50m of a junction?

Attribute 1-16 in the STATS-19 format gives details of the junction at which the accident occurred. Its possible values are shown below:

Value	Meaning
0	Not at or within 20 metres of a junction
1	Roundabout
2	Mini-roundabout
3	'T' or staggered junction
4	'Y' junction
5	Slip road
6	Crossroads
7	Multiple junction
8	Using private drive or entrance
9	Other junction

As no other attribute in the structured data gives the distance from a junction, the best possible query without resorting to the text is to find those reports which have the value 0.

Regarding the textual parts of the reports, the following rules were constructed to match all relevant instances within the text of the 300 sample reports:

```
Macro: DISTANCE
// e.g "23 METERS" or "Approx 10M"

(
  (APPROX)?
  (SPACE)?
  (PAREN)?
  ({Token.kind == number,Token.length == "1"} |
   {Token.kind == number,Token.length == "2"} |
   {Token.kind == number,Token.length == "3"} |
   {Token.kind == number,Token.length == "4"})
  (SPACE)?
  ({Token.string == "M"} |
   {Token.string == "METERS"} |
   {Token.string == "MET"} |
   {Token.string == "MTS"} |
   {Token.string == "MS"}      )
  (PAREN)?
)

Rule: distance
 (
  (DISTANCE)
  )
 :distance -->
   :distance.distance = {kind = "distance", rule = "distance" ,
      idAnnotationType="distanceVal"}
```

The following results were obtained after running the system with the three configurations:

Found by Manual Inspection		79
Structured Data Only	Reports Found	575
	Number of correct reports found	74
	Recall	94%
	Precision	13%
IE Data Only	Reports Found	80
	Number of correct reports found	79
	Recall	100%
	Precision	99%
Combined Structured and IE Data	Reports Found	79
	Number of correct reports found	79
	Recall	100%
	Precision	100%

Again the Structured Data Only results have high recall but with unacceptably low precision, more pronounced than in the first query. The IE Data Only results have just one false positive. In the distance macro shown above, the space before the distance is defined as optional by the question mark in the definition `'(SPACE)?'`, and as a result match was made with a report containing the name of a road ‘A40 M’. This report has a junction detail value of 6, and so ESTEST discards it in the Combined Data configuration, obtaining 100% for both recall and precision (however, if the value had been 0, as it could have been, ESTEST would have had the same results as for the IE Data Only).

6.3.3 Q3: How many accidents involve drunken pedestrians?

No part of the structured data gives any information on the state of pedestrians, and so for this query only the textual description of the accidents is used. The following Jape rules are used for the IE, matching all relevant instances within the 300 initial reports

```
Macro: DRUNK
(
 {Token.string == "DRUNK"} |
 {Token.string == "DRUNKEN"} |
 {Token.string == "DRINKER"}
)

Macro: PEDESTRIAN
(
 {Token.string == "PEDESTRIAN"} |
 {Token.string == "PED"} |
 {Token.string == "PERSON"}
)

Rule: drunkped
(
   ((DRUNK) (SPACE) (PEDESTRIAN)) |
   ((PEDESTRIAN) (SPACE) (DRUNK))
)
 :drunkped -->
  :drunkped.drunkped = {kind = "drunkped", rule = "drunkped"}
```

The following results were obtained running the system with the three configurations:

Found by Manual Inspection		9
Structured Data Only	Reports Found	0
	Number of correct reports found	0
	Recall	0%
	Precision	—
IE Data Only	Reports Found	7
	Number of correct reports found	7
	Recall	78%
	Precision	100%
Combined Structured and IE Data	Reports Found	7
	Number of correct reports found	7
	Recall	78%
	Precision	100%

In this case, as there is no structured data available, the results for IE Data Only and ESTEST's Combined Data are the same. There were 2 reports missed by both approaches; both of these referred to groups of drunken pedestrians and therefore had the text 'PEDESTRIANS' in one case and 'PEOPLE' in the other, which failed to match the above JAPE rules

6.3.4 Q4: Which accidents were caused by animals?

Attribute 1-25 in the STATS-19 lists any carriageway hazards involved in the accident:

Value	Meaning
0	None
1	Dislodged vehicle load in carriageway
2	Other object in carriageway
3	Involvement with previous accident
4	Dog in carriageway

5	Other animal in carriageway

Therefore, to answer this query from the structured data, reports with values 4 or 5 are selected.

For the IE only approach, the data sources integrated include an ontology with just 10 likely animals, and this was used as a named entity source. The following JAPE rules were created:

```
Macro: animal
         ({Lookup.minorType == animal})
Rule: animalOnLead
(
 (animal) {Token.string == "ON LEAD"}
)
:animalOnLead -->
    :animalOnLead.animalOnLead = {kind ="animalOnLead", rule =
"animalOnLead"}

Rule: animalRoad
(
 (animal) ({Token.string == "S"})?
)
:animalRoad -->
    :animalRoad.animalRoad = {kind ="animalRoad", rule =
"animalRoad", estestStore="no" }
```

The `animalOnLead` rule is there to fire in the place of `animalRoad` when the text states the animal is on a lead — from the 300 sample reports, we observed that when this is the case, the animal is not thought to constitute a carriageway hazard.

The following results were obtained running the system with the three configurations:

Found by Manual Inspection		8
Structured Data Only	Reports Found	8
	Number of correct reports found	8
	Recall	100%
	Precision	100%
IE Data Only	Reports Found	8

	Number of correct reports found	7
	Recall	88%
	Precision	87%
Combined Structured and IE Data	Reports Found	7
	Number of correct reports found	7
	Recall	88%
	Precision	100%

For this query, the Structured Data returned the correct results with no false positives. IE on its own found one incorrect report (which mentioned a "FOX CUB BUS" which is presumably either a make of bus or bus operator) and missed one report which mentioned "ANIMAL IN CARRIAGEWAY" rather than specifying the kind of animal.

Combining the two, ESTEST's Combined Data discards the false positive, but as matches are only being accepted where positive results are found for both IE and the structured data query, it fails to find the report missed by IE and therefore has a lower recall than the structured data query. ESTEST however would be able to answer the query "What kinds of animals cause accidents?", whereas from the structured data alone it is only possible to find the numbers of dogs versus all other kinds of animals.

6.3.5 Q5: How many accidents resulted in a collision with a lamppost?

No part of the structured data gives any information on the state of pedestrians, and so for this query only the textual description of the accidents is used. The following rule is created, as in the 300 sample reports lampposts are only mentioned when they are hit:

```
Rule: LAMPPOST
```

```
(
 {Token.string == "LAMPPOST"} |
 ({Token.string == "LAMP"} (SPACE) {Token.string == "POST"})
)
:lamppost -->
  :lamppost.lamppost = {kind = "lamppost", rule = "lamppost"}
```

The following results were obtained running the system with the three configurations:

Found by Manual Inspection		20
Structured Data Only	Reports Found	0
	Number of correct reports found	0
	Recall	0%
	Precision	—
IE Data Only	Reports Found	21
	Number of correct reports found	20
	Recall	100%
	Precision	95%
Combined Structured and IE Data	Reports Found	21
	Number of correct reports found	20
	Recall	100%
	Precision	95%

One false positive was found with configurations ii) and iii), and this was a report where the location of the accident was described as being opposite a lamppost on a particular street. As there is no relevant structured data, the results for ESTEST Combined Data are the same as for IE Data Only.

6.3.6 Summary of the Results

To summarise, the recall and precision results from the five queries are as follows:

Query	Structured Data Only		IE Data Only		ESTEST Combined Data	
	Recall	Precision	Recall	Precision	Recall	Precision
Q1	85%	18%	73%	91%	73%	100%
Q2	94%	13%	100%	99%	100%	100%
Q3	0%	–	78%	100%	78%	100%
Q4	100%	100%	88%	87%	88%	100%
Q5	0%	–	100%	95%	100%	95%

Table 6.1 Results of RTA Query Experiments

We see that the ESTEST Combined Data results are promising, even with the short time spent configuring the system by a user who is not a domain expert. The recall and precision are in line with state-of-the-art IE systems while the system is also able to automatically combine the extracted information with related structured data and to support queries which could not be answered by the structured or unstructured data alone. ESTEST is at least as good as either alternative approach, except for Q4, but even here it does have the advantage of being able to support finer granularity queries on the same data than the structured data only approach.

While conducting these experiments and in analysing the results, the following observations were made:

- The available sample size was relatively small, and only a few examples of each query were present. With a larger sample size, and as the rules are iteratively improved over time, we would expect the performance of the IE component to similarly improve over time.
- The ESTEST system was used to gather the IE-only results and, while the results produced are identical to those that would have been obtained by providing the same input to GATE, ESTEST still offers the advantage of semi-automatically creating the configuration and the query interface over the extracted data in the

HDM store, thus providing a generic and straightforward way of processing the results of the IE process compared to the annotations produced by GATE.

- In retrospect, the queries Q1-Q5 chosen were too focused on retrieving numbers of documents — essentially an IR task was performed which did not highlight the finer querying granularity of IE and future evaluations should do so.
- While checking the ESTEST results against the list of reports found by manual inspection, a surprisingly high number of errors were found in our manual marking process. If a similar experiment is repeated, recording the number of corrections made to this list would mean that the performance of a non-expert human performing the same task could be measured in terms of recall and precision and compared with the results achieved by ESTEST

6.4 Discussion

The road traffic accident domain fits the description of ‘partially structured data’ that we outlined earlier in this thesis, as it consists of a combination of structured data and free text. The text is stored as such not because it has secondary importance (in fact, when there is a disagreement between the text and the structured data the text is preferred), but rather because of the unpredictability of the specific circumstances of a road traffic accident and the difficulties of changing the schema format to include new data items of relevance that evolve over time.

In the simple example of ESTEST in use given earlier in this chapter, we showed how metadata from multiple heterogeneous data sources (two RTA relational databases and an RDF / RDFS ontology) can be automatically extracted and used in schema integration. This virtual global schema was then used to assist in the configuration of an IE process for extracting details of animals that were the cause of accidents. Such extracted data is automatically merged into the virtual global schema and is available to support queries on the global schema. To simulate the evolutionary way in which we envisage ESTEST being used, we then expanded the word forms associated with the concept `<<animal>>` automatically from WordNet. As a result, the IE process improved its recall, and querying the global schema returned the complete set of reports in which animals had in fact caused accidents.

The experiments conducted on real RTA data demonstrate that, even with the limitations on the data available and with a short set-up time, ESTEST is able to extract useful information and structured queries can be answered where before they could not.

Previous work on making use of the text in UK Road Traffic Accident reports has relied on expert knowledge of the sub-language used in the reports in order to code description logic-based grammar rules [Wu, J. and Heydecker, B. 1998]. In contrast, ESTEST makes use of the structured data as well as the text to answer queries, and our approach is more generally applicable to other application domains as it does not depend on a specific restricted sub-language.

Using this initial version of ESTEST, two limitations were observed:

Firstly, as mentioned in Section 6.2.7, a more general way of processing annotations for generating new instance data would be beneficial, both for storing a string value as the new instance identifier and for extracting identifiers that are substrings of the matching annotation. Because of the way that the JAPE rules work, it is not straightforward to obtain values to store in annotations: typically, the rule that fires against a concept will encompass more text than the value that should be used as the instance identifier. For example, for a distance value the rule that fires might encompass the text “about 25 meters” but the value that needs to be extracted is “25”. For the initial version of ESTEST, an algorithm to look for numeric values in annotation strings was used for the RTA experiments, but an extension to the JAPE language syntax would be a more general and effective solution, as this is a frequent requirement of ESTEST, namely the integration of extracted values into the virtual global database.

A second limitation in the design of ESTEST results from the likelihood that the user will iterate through the cycle several times. In general, it is desirable that, for each iteration, ESTEST removes previous results and assumes that the new configuration will extract all the results required. Without this assumption, it would be hard to correct errors from previous iterations, resulting in false positives being stored in the database. However, in this initial version of ESTEST, that meant that the stub grammar rules generated on each run overwrote the, often time-consuming, grammar rule changes made by the user. A better solution would be to associate grammar rules with the concepts to which they are related, and allow these rules to be amended or not, as the user chooses, on each iteration.

In the next chapter, we describe extensions to ESTEST made to overcome these two limitations and also to extend the research contribution of ESTEST in two further directions.

Chapter 7

Extending the ESTEST System

The first version of ESTEST (described in Chapters 5 & 6) serves as a test bed for demonstrating, end to end, our approach to better exploiting partially structured data. It is also a platform from which to develop specific areas further.

In this chapter, we describe four extensions to the original ESTEST system. Two of these are novel research contributions: firstly, making use of IE in order to provide a new schema matching technique, and secondly combining NLP coreference resolution techniques with database duplication detection techniques in order to obtain better results than is possible using either alone; these contributions are described in Sections 7.2 and 7.3, respectively.

The other two extensions described in this chapter are solutions to the two limitations identified in Chapter 6: Section 7.4 describes an extension which stores IE rule patterns as virtual global schema metadata in order to improve the usability of ESTEST, and in Section 7.5 we describe extending JAPE rules to allow automatic extraction of values relating to annotations for use in database applications. Throughout this chapter we illustrate the extensions made though an example, based on the crime domain, which is described in Section 7.1.

7.1 Crime Example

We have mentioned previously, in Chapter 2, that there are a range of applications in the crime domain which are built on partially structured data, for example systems built to support serious crime investigations, and systems for the collection of operational intelligence gathering; developing software for this domain is currently the subject of both research and industrial activity. We have collaborated with researchers from the Birkbeck Centre for Crime Informatics and have analysed several real crime databases which have been made available to them by police forces in the UK. A common feature of this domain is its use of text: for example, applications such as operational intelligence gathering make use of text reports containing observations of police officers on patrol; scene of crime applications include textual descriptions of the conditions found; serious crime investigations make use of witness statements. In these applications, the queries required will only become known over time, when looking for common patterns with earlier incidents for example.

In our running example through this chapter, we assume three data sources are available:

- OpIntel is a relational database containing textual reports of operation intelligence gathered by police officers on patrol.
- CarsDB is a relational database holding information on cars known to the police, with attributes such as the registration, colour, manufacturer and model.
- CrimeOnt is an RDF / RDFS ontology which states, amongst other things, that vehicles and public houses are attributes of operational intelligence reports.

The example demonstrates the four enhancements made to the ESTEST system. The data sources are integrated making use of correspondences identified by IE processing of the schema element names and description metadata. As in the previous version of ESTEST, the IE processing of the textual operational intelligence information then takes place and details of cars mentioned in the reports are extracted. However, in the extended version of the system, the extracted details of these textual references of cars are now compared to those in the structured `CarsDB`, and ESTEST is able to determine if the attributes extracted represent a new instance of a car or should be merged with an existing instance. The JAPE rules are generated from patterns stored in the EMR, rather than stubs which subsequently need to be amended by the user, as was the case in the first version of ESTEST. To demonstrate our approach for specifying the values to be used to identify the new entities stored, the details of cars found are identified by the associated car registration attributes, which are either found in the text or are matched with instances in the structured data.

Appendix C contains the complete ESTEST output from running this example, together with the data source definitions and ESTEST configuration scripts. In this chapter we do not comment on the steps of the example that are similar to those described in the earlier example in Chapter 6. Instead, only the relevant excerpts required to highlight the enhancements are included in Sections 7.2 – 7.5, each of which describes one enhancement in detail.

7.2 Information Extraction for Schema Matching

The first extension made to the ESTEST system is to make use of IE in schema matching. Schema matching is a long-standing problem in data integration research. The central issue is, given a number of schemas to be integrated, find semantic relationships between the elements of these schemas [Kashyap, V. and Sheth, A. 1996]. A comprehensive survey of approaches to automatic schema matching is [Rahm, E. and Bernstein, P.A. 2001] which gives a taxonomy based on the following criteria:

- *Instance vs schema based*: does the approach make use of instance data or only schema level information?
- *Element vs structure:* is the approach based on information from just each schema element or does it take into account more complex structures in the schema?
- *Language vs constraint:* is the approach based on a linguistic approach, e.g. the name of each schema element, or is it constraint-based e.g. using type information?

- *Matching cardinality:* are the mappings generated between elements in the two schemas 1:1, 1:n, n:1 or n:m?
- *Auxiliary information:* matchers may rely on not only the two input schemas but also on auxiliary information such as dictionaries, or on results from previous iterations of the schema matching process.

The linguistic approaches in the survey of [Rahm, E. and Bernstein, P.A. 2001] are divided into *name-based* and *description matching*. Name-based approaches are based on equality of names after pre-processing to deal with abbreviations, prefixes, and naming conventions. That survey also considers the use of synonyms, hyponyms, user-provided name matches and other similarity measures such as *soundex*, and *edit distance* where the number of delete, insert and replace operations required to transform string A into string B is used as a measure of how alike the two are. The possibility of using natural language understanding technology to exploit the schema descriptions is mentioned in the survey, but this is the only category where no prior work is explicitly cited and we are similarly unaware of any previous system which makes use of description metadata for schema matching.

The schema matching in the initial version of ESTEST, described in Chapter 5, collects word forms and associates them with concepts in the global schema. It also collects type information to be used in its match function. Domain abbreviations are used, and an extensible set of functions for pre-processing schema element names are built into the relevant wrappers e.g. the relational wrapper will find the word form "Account Name" from a schema element name "accName" given an abbreviation of "acc" for "account".

In terms of the schema matching taxonomy given above, which is also referred to in [Elemagarmind, A., Ipeirotis, P. et al. 2007], ESTEST makes use of schema information and not instance; element and not structure; both language and constraint; 1:1 mappings; and makes use of auxiliary information. The closest similar systems are LSD, TranScm and CUPID which we described in Chapter 2.

7.2.1 ESTEST Extensions for IE in Schema Matching

In the initial version of ESTEST described in Chapter 6, IE is configured with the help of the virtual global schema integrating the available structured data sources, and then ESTEST processes any associated free text data in order to extract new structured data. In the extended ESTEST, we add a new IE process which processes the available schema names and any textual metadata information, and uses the extracted word forms to identify correspondences between schema elements across different data sources.

A feature of the class of applications that ESTEST targets is that new structured data sources may become available over time, and they can be integrated into the virtual global schema and used to assist in IE. Therefore, the initial version of ESTEST included a schema matching component, while in the extended version of the system we make use of IE techniques to provide a novel approach to this task by making use of schema element descriptions to assist the schema matching component used to construct the global schema.

Each of the ESTEST wrappers (described in Section 5.2) are able to extract metadata for data sources organised according to the model they represent. The extended version of ESTEST extracts, in addition, relevant textual description metadata. For example, the relational wrapper retrieves the JDBC *remarks* data, which allows for a free text description of a column or a table, while the ontology wrapper retrieves XML comments.

Development teams designing databases often make use of naming conventions for schema elements, for example, specifying the case to be used and plurality of nouns in names. In the extended version of ESTEST, we have developed an extensible `schemaNameTokeniser` component which detects the use of common naming conventions in schema element names and descriptions. This is able to transform names into word forms, making use of abbreviations. For example, “AccountNum”, ”accNum” and ”ACCOUNT-NUMBER” can all be transformed into the word form "account number". The schema tokeniser is implemented as a GATE tokeniser component so that it can be run in any pipeline as required. In the same way that the GATE `EnglishTokeniser` splits text into annotations representing words and punctuation, the tokens produced by our `SchemaNameTokeniser` produces separated words from the schema names. Our component makes use of a set of regular expressions to identify naming conventions; these cover the commonly used conventions of which we are aware and can be easily expanded.

The steps taken in the extended version of ESTEST to make use of IE in schema matching are as follows:

i) For all available data sources, the available schema metadata is extracted, including the textual names and description metadata for all the elements of the schema.

ii) The `SchemaNameTokenser` processes the schema names, and extracts word forms to be stored in the EMR.

iii) A GATE pipeline is constructed in order to process the textual description metadata. This pipeline performs named entity recognition on the schema element descriptions, identifying references to schema elements by matching the word forms extracted from the schema names. The pipeline is created automatically by constructing an instance of our `SchemaGazetteer` component and a JAPE processor configured with a grammar which treats each element in the schema as a named entity source using the word forms extracted from schema element names.

iv) Where a match is found between a schema element acting as a named entity source and a description of a schema element from a different data source, then a possible correspondence between these schema elements is inferred.

7.2.2 Schema Matching in the Crime Example

We now show how this process takes place in the example from the crime domain. No correspondences would be found if this example were to be run using the original version of ESTEST of Chapter 5. The match that should be found is between the `car` table in the `CarsDB` data source and the `vehicle` RDFS class in the `CrimeOnt` data source, but there is no common word form between them. However, the Postgres DDL for the CarsDB includes comments about the table and columns as shown below. We see that the comment for `car` includes the word "VEHICLE" and so will be found by our new IE processing of schema element descriptions:

```
comment on table car is 'VEHICLE SEEN DURING OPERATIONAL
INTELLIGENCE GATHERING';
comment on column car.reg is 'UK REG MARK IE TWO CHAR AREA CODE,
AGE, AND THREE RANDOM LETTERS';
comment on column car.manufacturer is 'NAME OF MANUFACTURER E.G.
FORD';
comment on column car.model is 'NAME OF THE MODEL OF THE CAR E.G.
FIESTA';
comment on column car.colour is 'COLOUR OF THE CAR E.G. BLUE';
```

The first step is for ESTEST to process the schema element names using the `SchemaNameTokeniser` component. This happens and the word forms "car" and "vehicle" are extracted from the respective schemas and are stored in the EMR:

```
Now find word forms for each schema element using the
SchemaNameTokeniser component.

Creating schema name tokeniser to process names of schema
elements.

Using C:\Program Files\GATE 3.0 as GATE home
Using C:\Program Files\GATE 3.0\plugins as installed plug-ins
directory.
Using C:\Program Files\GATE 3.0\gate.xml as site configuration
file.
Using C:\Documents and Settings\dean\gate.xml as user
configuration file
CREOLE plugin loaded: file:/C:/Program Files/GATE-
3.1b1/plugins/ANNIE/
CREOLE plugin loaded: file:/C:/estest/

Storing Data Source info and metadata in EMR.....
```

Next, the GATE pipeline for processing description metadata is created (the JAPE rules to process the schema metadata contained in `smie.jape` are automatically created and are shown in Appendix C.2):

```
=================================================================
   CREATING GATE PIPELINE FOR SCHEMA MATCHING.
=================================================================

Creating Default Tokeniser Gate processing resource.
Creating Sentence-Splitter Gate processing resource.
Creating Database-Gazetteer Gate processing resource with all sch
 ema objects as NE sources based on the word forms extracted from
  schema names.
  No data source URL provided so loading word form named entities
   for all schema elements.
  Loading definition of Named Entity

=================================================================

   PROCESSING TEXTUAL METADATA FOR SCHEMA MATCHING
=================================================================

Creating Jape rules for processing schema metadata: smie.jape pat
 h: C:\estest\smie.jape
Creating Jape Transducer Gate Processing Resource.
JAPE URL: file:/C:/estest/smie.jape
Assembling Components Into Pipeline.
Gate is now initialised and the ESTEST application is built.
```

Now each description is processed, and no matches are found from the other descriptions. However, when the description of the `car` table is processed, a match is found with the `vehicle` class, and the schema id and schema element id are displayed to identify the elements involved in the match:

```
Document to be processed by IE : 'VEHICLE SEEN DURING OPERATIONAL
  INTELLIGENCE GATHERING'
Running processing resources over document...
Match between the textual metadata of schema element 84/62, and
th
 e schema element 85/104
```

Once processing is complete, this remains the only match identified and it is used by ESTEST in creating the global schema:

```
Going to find matches between Schema elements.

The matches are:
  Match with 0.5% confidence on word form meta-match
    Schema 1: Carsautzv, Concept 1: car
    Schema 2: CrimeOntEstest, Concept 2: vehicle
```

7.2.3 Making Use of WordNet in Schema Matching

We have also experimented with an additional novel approach to suggesting correspondences, using a measure of semantic distance between nouns in WordNet. We described in Chapter 5 how the initial version of ESTEST made use of WordNet to expand the number of word forms associated with concepts in the EMR in order to increase the number of matches obtained by IE rules. WordNet is organised as a semantic net with a set of synonyms, termed a *synset*, associated to each concept. Concepts are linked by relationships, including hyponyms. This subset of the structure of WordNet is structurally similar to our EMR, which is organised into concepts with related sets of word forms.

In our WordNet-based schema matching approach, after the collection of word forms from the IE processing of schema metadata, matches are suggested by comparing the distance between concepts in WordNet. This estimate of semantic distance is implemented as an extension to the ESTEST `WordNet` class that we previously used for expanding the number of word forms associated with a concept. Our straightforward, but extensible, measure finds the minimum distance between concepts linked to the word forms. Each word form can be linked to a number of concepts; for example, the word form 'chair' has four senses as a noun: a seat, a professorship, an officer of an organisation, and an instrument of execution by electrocution. The distance between 'chair' and 'stool' will be very different, depending on which sense of each is chosen. In our simple measure, the distance between each pair of senses is found and the minimum distance is used.

Measures of semantic distance in WordNet are a research topic in its own right: see [Budanitsky, A. and Hirst, G. 2001] for an evaluation of various approaches. An assumption of any measure based on WordNet is that the distance between concepts in the semantic net is in some way related to how far removed the concepts are in a human conception of the real world. WordNet concepts are constructed from the English language and therefore the density of words in an area will affect distance;

the number of edges will be an indication of the density of words in that area of the net rather than how different the concepts are semantically. Despite the limitations apparent in using WordNet as a measure of the semantic distance between two real-world concepts, the alternatives such as measuring the edit distance, seem to us to be no more likely to be useful in constructing effective applications.

We have not shown WordNet being used for schema matching in the crime example. However, taking the noun word forms collected by the `SchemaNameTokeniser` and finding the distance between them using our simple measure yields the following results:

Word Form	**Word Form**	**Distance**
CAR	COLOUR	12
CAR	ID	16
CAR	PC	11
CAR	PUB	11
CAR	VEHICLE	4
COLOUR	ID	10
COLOUR	PC	11
COLOUR	PUB	9
COLOUR	VEHICLE	8
ID	PC	15
ID	PUB	13
ID	VEHICLE	12
PC	PUB	10
PC	VEHICLE	7
PUB	VEHICLE	7

If this WordNet approach had been used in our running example, in place of the IE processing of schema element textual metadata, then the same match would have been found since the closest shortest distance is between “CAR” and “VEHICLE”, with a distance of 4 synsets. However, the above results also illustrate that as concepts become more removed from one another, the distance metric between them does not scale in a linear fashion. For example, the distance between “PUB” and “VEHICLE” is just 7. We have observed this as a common characteristic of distance in WordNet and it seems to be related to the speed with which paths in WordNet lead to very general, abstract concepts. For example, the path between “CAR” and “VEHICLE” is:

```
car, auto, automobile, machine, motorcar
motor_vehicle, automotive_vehicle
self-propelled_vehicle
wheeled_vehicle
vehicle
```

However, between “PUB” and “VEHICLE” are the very general concepts “artifact” (defined in WordNet as “a man-made object taken as a whole”) and “instrumentation” (defined in WordNet as “an artifact (or system of artifacts) that is instrumental in accomplishing some end”):

```
public_house, pub, saloon, pothouse, gin_mill, taphouse
tavern, tap_house
building, edifice
structure, construction
artifact, artefact
instrumentality, instrumentation
conveyance, transport
vehicle
```

The potential for using WordNet is therefore likely to be a decision on whether or not concepts are close, rather than to compare their relative distances. This approach can also be used in conjunction with our IE processing of schema element textual metadata. We envisage both approaches being useful in providing intelligent ordering and filtering of suggestions to the end-user in the ultimate workbench that we envisage (discussed in Chapter 8).

7.3 Combining Duplicate Detection and Coreference Resolution

In both database and natural language processing, there is often a requirement to detect duplicate references to the same real-world entity. In databases, this may be in order to detect and repair multiple representations of the same entity, while in natural language, deciding when multiple fragments of text refer to the same entity has proved hard to solve automatically. For example, in the text "The car was driving too fast. It crashed.", there are two references to the same vehicle, "The car" and "It", but the IE techniques we have made use of so far would not be able to handle this level of complexity. Techniques that do address this using a greater level of linguistic awareness are described in Section 3.1.

ESTEST provides the ability to make use of both structured data and free text by combining database duplication detection and NLP coreference resolution techniques. We have extended the ESTEST system to combine, as far as we are aware for the first time, techniques from these two fields in order to achieve better results than would be obtainable for each independently. In the rest of this Section 7.3, we describe the state of the art in both database and NLP techniques, and then describe how ESTEST is able to combine evidence from new extensible components that implement approaches from each discipline.

7.3.1 Detecting Duplicates in Databases

Deciding if two instances in a database in fact refer to the same real-world entity is a long-standing problem in database research e.g. [Newcombe, H.B., Kennedy, J.M. et al. 1959; Winkler, W. 1994]. Attempting to solve the problem across a range of application domains has led to a variety of terms being used in the literature for different flavours of the same fundamental task.

The statistics community has undertaken over five decades of *record-linkage* work, particularly in the context of census results. A central task in this domain is the need to merge records from two files, and the standard approach is based on the Fellegi-Sunter mathematical model [Fellegi, I.P. and Sunter, A.B. 1969] which formalises the idea of using odds-ratios of frequencies originally introduced in [Newcombe, H.B., Kennedy, J.M. et al. 1959]. In systems based on this approach, the evidence is weighed using conditional probabilities and compared to a threshold, and the records are either designated as matches, non-matches or where no conclusion can be made are flagged for human review. This statistical, record-based approach is, not surprisingly, an active research area at the US Census Bureau. [Winkler, W.E. 2006] gives an overview the current state of record linkage research and of related topics such as error rate estimation and string comparison metrics.

In database integration, the term *merge / purge* [Hernandez, M.A. and Stolfo, S.J. 1998] is used to describe approaches such as sorting the data and traversing it considering a 'window' of records for possible merging. *Data cleansing* [Müller, H. and Freytag, J.-C. 2003] concentrates on finding anomalies in large datasets and cleansing the dataset by using these anomalies to decide on merging duplicates and on deriving missing values. *Duplicate elimination* [Bitton, D. and DeWitt, D.J. 1983] refers to variations on merge-sort algorithms and hash functions for detecting duplicates. Other terms such as *object identification* and *reference disambiguation* are also used in the literature. Throughout this thesis, we use the term *detecting duplicates* to cover the general problem these different terms encompass.

We are not aware of any attempt to make use of NLP techniques in finding duplicates in databases, beyond character-based similarity metrics such as edit distance.

7.3.2 Coreference Annotation in NLP

In linguistics, *anaphora resolution* [Hirst, G. 1981] refers to the task of identifying references in text to some previously mentioned item - the term anaphora is derived from the Greek for "carrying back" [Mitkov, R. 1999]. In the sentence "Joe spoke about his day", the word "his" is the *anaphor* which refers back to the *antecedent* "Joe". A range of approaches have been developed for anaphora resolution, including purely syntax-based techniques, e.g. [Hobbs, J.R. 1976], while some depend on semantics, e.g. [Wilks, Y. 1975] and other alternatives are based on statistical analysis, e.g. [Ge, N., Hale, J. et al. 1998].

As part of their description of the FASTUS IE system, [Kameyama, M. 1997] describe why the task of finding references to the same entity in IE is more general than anaphora resolution, being based on merging information without necessarily being able to rely purely on linguistic anaphoric expressions. Subsequently in IE, the term *coreference annotation* [Morton, T. 1997] has come to be used to describe the task of identifying where two noun phrases refer to the same real-world entity, and this is a generalisation of anaphora resolution. Coreference annotation involves finding chains of references to the same entity throughout the processed text.

There are a number of types of coreference, including: *pronominal coreference,* where the proper antecedents are found for pronouns such as "I", "me", "my" and "yourself"; *proper names coreference,* which deals with variations of names e.g. "Big Blue" and "IBM"; through to more complicated linguistic references such as *demonstrative coreference* where phrases like "this" and "that" co-refer to objects in the text. Despite the wide range of coreference types, [Bagga, A. 1998] shows that a small number of types of coreference account for most of the occurrences in real text: they find that proper names account for 28% of all instances, pronouns 21% but demonstrative phrases only 2%. It is expected therefore that reasonable coreference annotation results can be achieved by handling effectively these main categories of coreference.

7.3.3 Coreference Annotation in GATE

The GATE system provides support for these two main categories of coreference, by providing an `OrthoMatcher` component which performs proper names coreference annotation, and a JAPE grammar which when executed performs pronominal coreference annotation.

The `OrthoMatcher` is executed in a GATE pipeline following the named entity recognition step. No new named entities will be found as a result of finding matches, but types may be assigned to previously unclassified proper names if other annotations in the chain were typed. The input to the `OrthoMatcher` component is a list of sets of aliases for named entities. For example, one entry in the list might be {"IBM", "International Business Machines"}. Also input are a list of exceptions that might otherwise be matched incorrectly, e.g. "Eastern Airways" is not the same organisation as "Eastern Air". As well as these string comparison rules that apply to any annotation type, there are some specific rules for the core MUC IE types i.e. person, organisation, location and date. For example, the various ways companies can be named, e.g. with "Ltd" at the end of the name, is handled by one of these specialist rules.

GATE's pronominal coreference module resolves pronominal coreference for locations, people and organisations, including *pleonastic it,* e.g. "It is snowing", where it is important to detect that the pronoun does not refer to a particular antecedent and so prevent the creation of a false positive.

The effect of adding these two coreference components to a GATE pipeline is to create *annotation chains* linking multiple references to the same entity. These chains are implemented by adding an attribute to each annotation in the chain containing the matching attributes. For example, running the default GATE configuration over a document containing the text "Tony Blair spoke next, he said that NASA was an abbreviation for 'National Aeronautics and Space Administration', but who believes him?" would produce annotations including the following `person` and `organisation` annotations:

Type	**ID**	**Text**	**Attributes**
Person	1	Tony Blair	{gender=male, rule=PersonFinal, rule1=PersonFull}
Organization	3	NASA	{orgType=government, rule1=GazOrganization, rule2=OrgFinal}
Organization	4	National Aeronautics and Space Administration	{orgType=government, rule1=GazOrganization, rule2=OrgFinal}

Running the same text through a pipeline with the two coreference steps added will now result in the following annotations, which include two annotation chains, one for the Prime Minister and one for NASA. The pronouns have been classified as being of type `person`.

Type	ID	Text	Attributes
Person	1	Tony Blair	{gender=male, matches=[1,2,5], rule=PersonFinal, rule1=PersonFull}
Person	2	he	{ENTITY_MENTION_TYPE=PRONOUN, antecedent_offset=0, matches=[1,2,5]}
Organization	3	NASA	{matches=[3,4], orgType=government, rule1=GazOrganization, rule2=OrgFinal}
Organization	4	National Aeronautics and Space Administration	{matches=[3,4], orgType=government, rule1=GazOrganization, rule2=OrgFinal}
Person	5	him	{ENTITY_MENTION_TYPE=PRONOUN, antecedent_offset=0, matches=[910, 936, 937]}

7.3.4 Combining the two approaches in ESTEST

We argue in this thesis that coreference annotation in NLP is essentially the same task as duplicate detection in databases. The difference is not in the task to be performed but rather in the structure of the data to be processed, free text in the case of NLP and structured data for databases.

As ESTEST, unlike other systems, is able to combine structured data and free text by applying both IE and data integration techniques, the system is well placed to improve the performance of the duplicate detection task by combining evidence from both approaches. To achieve this, a coreference component, a template constructor, and a duplicate detection component are required for ESTEST. The coreference component may find new structured data which in turn will be useful for the duplicate detection component, and vice versa.

These new components of ESTEST use state-of-the-art techniques, but as there is much active research in both areas they are designed to be extensible and modular. Our research contribution here is not in coreference resolution research nor in database duplication detection, but in combining the two approaches. As we will see later in this section, combining these techniques can yield results that are better than using either in isolation.

In particular, in order to enable coreference detection, the GATE IE pipeline constructed by ETEST has been expanded to include the following:

1) The standard GATE `OrthoMatcher` component is added to the pipeline. The configuration for this component is automatically created from data in the EMR (ESTEST Metadata Repository). When building an IE application using GATE alone, this component would have to be configured by hand for the domain of interest, and the default configuration file provided contains only a handful of examples to show the format of the entries. In contrast, in ESTEST we use the abbreviations and alternative word forms collected within the EMR in order to automatically create the configuration for the `OrthoMatcher` component.

2) The standard JAPE grammar for pronominal coreference is then executed over the text.

3) Template instances are automatically constructed for the annotations that match schema elements.

4) ESTEST then extracts the coreference chains from the annotations and uses these to merge templates. Each co-reference chain is examined in turn and its

attributes examined and compared pair-wise to decide if the chains should be merged. This process removes duplicates from within the free text.

5) The resulting set of templates are now compared to the instances already known from the structured data. A decision is made whether to store each template found in the free text as a new instance, or whether instead to merge it with an existing instance.

6) Any annotations which refer to schema elements but which are not part of any template are stored as before.

The same process is used for both the decision on whether to merge templates found in the free text, and whether to store templates as a new instance or to merge with an existing instance. The available evidence is compared, using the size of the extent of each attribute as a straightforward method of weighting the evidence of different attributes in a template. For example, in a template with attributes `name` and `gender`, `name` would be weighted more highly as it is more discriminating as a search argument. If the database contained the following two `person` concepts, linked by a suggested coreference chain with no, or only limited, conflicting evidence:

Person-1	Person-2
Name: George Bush	Name: ?
Gender: ?	Gender: Male

then they would be merged into:

Person-1
Name: George Bush
Gender: Male

In contrast, if the amount of conflicting evidence exceeded a confidence threshold, then the separate instances would be left unmerged and the coreference chain ignored e.g.

Person-1	Person-2
Name: George Bush	Name: Jane Fonda
Gender: Male	Gender: Female

If there is some contradictory evidence, falling between these two confidence thresholds, then the coreference chain is highlighted for the user to decide whether to merge, together with the conflicting attributes and the confidence level based on the weighting of these attributes.

To decide the confidence level, the attributes of each pair of possible matches in the chain are compared. Where there is no value for an attribute for one or both of the concepts then no positive or negative evidence is assumed. If both concepts have the same value for the attribute then that is considered as positive evidence weighted according to the selectivity of the attribute. If they have different values then similarly weighted negative evidence is added to the confidence total.

For example, if two instances of `person` have the same name but different genders, then according to the following evidence they are likely to be the same person and should be merged:

Person-1 (84% match with Person-2)	Person-2
Name: George Bush (92%)	Name: George Bush
Gender: Male (8%)	Gender: Female

7.3.4 Duplicate detection in the Crime Example

We made use of the crime example to experiment with our approach. The first step described above is to make use of GATE's `OrthoMatcher` component by automatically creating an input file based on the word forms associated with the schema elements in the EMR. As mentioned, this component is used to provide alternative names for the same instance of an entity, for example "IBM" and "Big Blue". In the crime example, the following matches are found:

1	OP, INTEL, OP INTEL
2	ID, REPORT, REPORT ID
3	CAR,VEHICLE

Using this component, it became clear that with such general matches the co-reference chains were not producing any value, and in fact every annotation of the same type matched, so the recall was high but the precision too low.

It may be that there are uses for this approach in particular domains, and for annotations referring to people there is value in exploiting the extra evidence available in natural language. However, for ESTEST we decided instead to look for coreference matches by constructing templates and attempting to merge these in cases where there are no conflicting attributes. It would also be possible to restrict merges to templates found in close proximity in the text. While the GATE coreference components remain in ESTEST, their results are not used and their output has been suppressed in the listing given in Appendix C.

In our running example, three operational intelligence reports which are processed by ESTEST contain references to cars:

Report Num	**Operational Intelligence Report**	**Notes**
1	GEORGE BUSH HAS A NEW YELLOW CAR REGISTRATION LO78 HYS. IT IS A FORD MONDEO.	This is in `CarsDB` but without the fact that the car is yellow. There are two references to the same car in this text “NEW YELLOW CAR REGISTRATION LO78 HYS” and “FORD MONDEO”
2	TONY BLAIR SEEN COMING OUT OF THE PERSEVERANCE PUBLIC HOUSE DRIVES OFF IN A GREEN FORD PUMA UY22 QWC.	This car is not in `CarsDB`
3	NICHOLAS SARKOZY NOW DRIVING BLUE CITRON 2CV CE21 FGH.	This car is in CarsDB but is recorded there as a Red Citron 2CV

We now describe the steps ESTEST takes in processing each of these pieces of text, and we highlight the relevant output from the system:

Report Number 1: “GEORGE BUSH HAS A NEW YELLOW CAR REGISTRATION LO78 HYS. IT IS A FORD MONDEO”. 73 annotations are produced from these two sentences. It is interesting to note the large number of annotations produced, even from so short a piece of text and so few rules. An example of a car annotation from the output is:

```
AnnotationImpl: id=62; type=car; features={kind=car, rule=
new_car0, idAnnotationType=car_reg, matches=[62, 63, 64, 60,
61]}; start=NodeImpl: id=15; offset=45; end=NodeImpl: id=20;
offset=54
```

ESTEST next finds the size of the extent of each schema element in order to use these numbers to weigh the evidence found of matches in the text:

```
The total concept counts in the DB:

  Concept: car_reg, count: 142
  Concept: op_intel, count: 3
  Concept: manufacturer, count: 3
  Concept: colour_colour, count: 22
  Concept: colour, count: 22
  Concept: model_model, count: 11
  Concept: Resource, count: 0
  Concept: pub, count: 0
  Concept: op_intel_pc, count: 1
  Concept: op_intel_report_id, count: 3
  Concept: car, count: 142
  Concept: opIntel, count: 0
  Concept: op_intel_intel, count: 3
  Concept: model, count: 11
  Concept: manufacturer_manufacturer, count: 3
```

Next, ESTEST extracts the annotations of interest to be considered in creating templates (more detail of how this is achieved is given in Section 7.5):

```
Annotations of interest

Annotation Details:

Schema element = '<<car>>', value = 'YELLOW CAR REGISTRATION LO78
HYS', Gate ID is 48 and the ID is the value of the related
car_reg
```

```
Schema element = '<<car>>', value = ' FORD MONDEO', Gate ID is 68
and the ID is the value of the related car_reg

Schema element = '<<colour>>', value = 'YELLOW', Gate ID is 47
and the ID is the value of the related colour

Schema element = '<<car_reg>>', value = 'LO78 HYS', Gate ID is 65
and the ID is the value of the related car_reg

Schema element = '<<model>>', value = 'MONDEO', Gate ID is 74 and
the ID is the value of the related model

Schema element = '<<manufacturer>>', value = 'FORD', Gate ID is
69 and the ID is the value of the related manufacturer
```

Two templates are found, one for each reference to a car in the text:

```
2 templates found

    Templates are:
      Template: 1 -- <<car>>, Instance ID: estestInstance1
        Attribute: <<car_reg>>, Instance ID: LO78 HYS
        Attribute: <<colour>>, Instance ID: YELLOW


      Template: 2 -- <<car>>, Instance ID: estestInstance2
        Attribute: <<manufacturer>>, Instance ID: FORD
        Attribute: <<model>>, Instance ID: MONDEO
```

These contain no conflicting attributes and so it is assumed that they refer to the same entity (this assumption replaces the alternative co-reference chain approach):

```
merging template1 into3

  Set idAnnotationType to <<car_reg>>for template <<car>>

  Replacing templateInstanceId null with estestInstance1

  Comparing <<car_reg>> to <<car_reg>>

  Merging templates has found a new annotation of the template id
   type <<car_reg>>, using this as the id of the template, id is
   LO78 HYS

  Comparing <<colour>> to <<car_reg>>

  Changing template instance id from estestInstance1, to LO78 HYS

  merging template2 into3

  Comparing <<manufacturer>> to <<car_reg>>

  Comparing <<model>> to <<car_reg>>

  mergeTemplates() is returning 1 merged templates
```

```
1 templates after merging
```

The resulting merged template contains all the available attributes of the car, and uses the car registration number as the identifier for the car (this is discussed in more detail in Section 7.5):

```
Template: 3 -- <<car>>, Instance ID: LO78 HYS
        Attribute: <<car_reg>>, Instance ID: LO78 HYS
        Attribute: <<colour>>, Instance ID: YELLOW
        Attribute: <<manufacturer>>, Instance ID: FORD
        Attribute: <<model>>, Instance ID: MONDEO
```

The merged template is now compared to the instances already contained in the structured data and the size of the attributes' extents is used to weight the attributes. For example, having the same registration mark is more credible evidence that the car in the text is a reference to one already known than would be the fact that they were both the same colour:

```
Evidence of a match with <<car>>,<<car_reg>>
  --> LO78 HYS,LO78 HYS, weight is 79.78%

Evidence of a match with <<car>>,<<manufacturer>>
  --> BD51 ABC,FORD, weight is 1.69%

Evidence of a match with<<car>>,<<manufacturer>>
  --> IK83 OKE, FORD, weight is 1.69%

Evidence of a match with<<car>>,<<manufacturer>>
  --> LO78 HYS, FORD, weight is 1.69%

Evidence of a match with<<car>>,<<model>>
  --> LO78 HYS,MONDEO, weight is 6.18%

Match with BD51 ABC, evidence is 1.69%
Match with LO78 HYS, evidence is 87.64%
Match with IK83 OKE, evidence is 1.69%

Best match was LO78 HYS at 87.64%

Found match with more than 50% likelihood LO78 HYS, evidence:
   87.64%
```

We see that the correct match is found with 'LO78 HYS'. The details from the template are then stored in the HDM store, including the new fact that this car is yellow. This is the only fact that actually needs to be stored in the HDM store as the others already exist in the CarsDB data source. However, there is no disadvantage in duplicating known facts within the HDM since distinct result sets are returned from queries and in this way there is a useful record of the totality of facts found in the text.

```
Storing Templates.


Storing template: <<car>> / LO78 HYS
   Storing template attribute edge
   <<attribute,car,car_reg>> [1,LO78 HYS]

   Storing template attribute edge
   <<attribute,car,colour>> [1,YELLOW]

   Storing template attribute edge
   <<attribute,car,manufacturer>> [1,FORD]


   Storing template attribute edge
   <<attribute,car,model>>[1,MONDEO]
```

Report Number 2: “'TONY BLAIR SEEN COMING OUT OF THE PERSEVERANCE PUBLIC HOUSE DRIVES OFF IN A GREEN FORD PUMA UY22 QWC”. The template extracted from the text is:

```
Template: 5 -- <<car>>, Instance ID: UY22 QWC

        Attribute: <<car_reg>>, Instance ID: UY22 QWC
        Attribute: <<colour>>, Instance ID: GREEN
        Attribute: <<manufacturer>>, Instance ID: FORD
        Attribute: <<model>>, Instance ID: PUMA
```

Comparing the template to the known structured data produces some evidence of a match, for example with cars of the same make or model, but the total weight is less than 50% so no match is assumed and the template is stored as a new instance in the HDM store:

```
Evidence of a match with <<car>>,<<colour>>
  --> IK83 OKE,GREEN,

weight is 12.36%

Evidence of a match with <<car>>,<<manufacturer>>
  --> LO78 HYS,FORD, weight is 1.69%
```

```
Evidence of a match with <<car>>,<<manufacturer>>
  --> BD51 ABC,FORD, weight is 1.69%

Evidence of a match with <<car>>,<<manufacturer>>
  --> IK83 OKE,FORD, weight is 1.69%

Evidence of a match with<<car>>,<<manufacturer>>
  --> LO78 HYS,FORD, weight is 1.69%

Evidence of a match with<<car>>,<<model>>
  --> BD51 ABC,PUMA, weight is 6.18%

Match with BD51 ABC, evidence is 7.87%
Match with LO78 HYS, evidence is 3.37%
Match with IK83 OKE, evidence is 14.0%

Best match was IK83 OKE at 14.04%
```

Report Number 3: " NICHOLAS SARKOZY NOW DRIVING BLUE CITRON 2CV CE21 FGH". We assume in this example the French President's car has been re-sprayed, as the colour in the text (blue) does not match the colour in CarsDB (red). The template extracted from the text is:

```
Template: 6 -- <<car>>, Instance ID: estestInstance15


   Attribute: <<car_reg>>, Instance ID: CE21 FGH
   Attribute: <<colour>>, Instance ID: BLUE
   Attribute: <<manufacturer>>, Instance ID: CITRON
   Attribute: <<model>>, Instance ID: 2CV
```

Comparing the template to the known instances in the structured data provides a match with the correct car even after discounting the colours not matching. It may make sense to be able to make contradictions weigh more heavily than matches, and this could be supported in the end-user workbench that we describe as possible future work in Chapter 8.

```
Evidence of a match with <<car>>,<<car_reg>>
   --> CE21 FGH,CE21 FGH, weight is 79.89%

Contradiction with existing edge: <<car>>,<<colour>>
   --> CE21 FGH, the annotation attribute is BLUE while the db
       attribute is RED evidence is-12.29%

Evidence of a match with <<car>>,<<colour>>
   --> BD51 ABC,BLUE, weight is 12.29%
```

```
Evidence of a match with <<car>>,<<manufacturer>>
   --> CE21 FGH, CITRON, weight is 1.68%

Evidence of a match with <<car>>,<<model>>
   --> CE21 FGH,2CV, weight is 6.15%

Match with BD51 ABC, evidence is 12.29%
Match with CE21 FGH, evidence is 75.42%

Best match was CE21 FGH at 75.42%

Found match with more than 50% likelihood CE21 FGH, evidence:
75.42%
```

7.4 Text Matching Patterns in the EMR

We now turn to the third extension made to the initial version of ESTEST, in order to overcome the need for the user to reconfigure the JAPE[6] rules from scratch on each cycle.

In the initial version of ESTEST, the annotations found on the previous IE process run are reset when the IE process is re-run. If this were not done, it would not be possible to take into account any change in configuration that would result in amending or removing annotations found on the previous run. However, as described in Chapter 6, the need for the user to amend, on each iteration of ESTEST, the stub JAPE rules which include text matching patterns is a limitation that would be a barrier to its practical use by end users. We have therefore extended the EMR to associate patterns with concepts. These patterns are used to generate, automatically, the IE rules to be applied on each ESTEST iteration. There are three kinds of pattern which can be specified stored in the EMR:

[6] GATE's *Java Annotation Patterns Engine* (JAPE) provides finite state transduction over annotations based on regular expressions. The left-hand-side of JAPE rules consist of an annotation pattern that may contain regular expression operators. The right-hand-side consists of annotation manipulation statements, typically by creating an annotation over the matched text which describes its type. JAPE rules are grouped into a *grammar* for processing as a discrete step in an IE *pipeline*, and for each grammar one of a number of alternative control styles is specified to determine the order rules should fire, and from where in the text processing should resume.

- A `macro` is a pattern used as a shortcut for specifying textual alternatives. It is not linked to any specific concept in the global schema and there are no annotations associated with it. Macros are defined by a `name` and a `value`, which specifies the alternative text strings (which can themselves contain macros) that will match this macro.
- A `schemaElementMatch` is a pattern linked to a schema element, and annotations found in the text are treated as discovered instances of the schema element in the text. Like macros, these have a name and a list of text strings, which match an instance of this schema element. Patterns additionally require the name of the related schema element in the global schema to be specified.
- To identify schema elements to act as sources of named entities, a special case is allowed where a `macro` is defined with no `value` — in this case the `name` is assumed to be a schema element name and the extent of that object is used as the source of named entity instances (in retrospect, it would have been clearer to have supported an alternative kind of pattern other than `macro`, as in this special case annotations in the text are found and a JAPE rule is created as well as a JAPE macro).

For example, to define a number of alternative words used for street lighting, given the following `macro`:

```
<macro>
   <name>model</name>
   <value>lamppost-OR-streetlight </value>
</macro>
```

ESTEST will include the following JAPE macro in the configuration file used for its IE processing:

```
Macro: lamppost
(
  {Token.string == "lamppost"}  |
  {Token.string == "streetlight"}
)
```

Other patterns can now refer to this macro by referring to (LAMPPOST) in their specification. An example of the special case of a named entity macro, used to define the schema element `model` from the crime example as a named entity source, is:

```
<macro>
   <name>model</name>
</macro>
```

This will generate the following JAPE macro and rule:

```
Macro: MODEL
({Lookup.minorType == model})

Rule: model
(
(MODEL)
)
:model -->
   :model.model = {kind ="model", rule = "model",
                   idAnnotationType="model"}
```

Additionally, an entry will be created in the configuration for the ESTEST `SchemaGazetteer` component to link these annotations to the schema element in the global schema:

```
<ne_source>
   <schema>OpIntelautzx</schema>
   <object>model</object>
   <type>extent</type>
</ne_source>
```

Finally, the `schemaElementMatch` below:

```
<schemaElementMatch>
   <name>car</name>
   <schema_object_name>car</schema_object_name>
   <value>(COLOUR)?--(MANUFACTURER)?--(MODEL)--(CAR_REG)</value>
</schemaElementMatch>
```

will generate the following JAPE rule:

```
Rule: car
(
(COLOUR)? (SPACE)? (MANUFACTURER)? (SPACE)? (MODEL) (SPACE)?
(CAR_REG)
):car -->
   :car.car = {kind = "car", rule = "known_car0"}
```

In Section 7.5 below, we describe use of EMR patterns in the crime example, together with the use of automatic extraction of values from the text.

7.5 Automatic Extraction of Values from Text

A central task in IE is named entity recognition, which at its simplest involves identifying proper names in text by matching against lists of known entity values, although patterns can also be defined to recognise entities. In GATE, list-matching named entity recognition is performed by gazetteer components, while pattern matching named entity recognition is performed by JAPE. However, as we noted in Section 6.2.7, the facilities offered by IE systems, such as GATE, are restrictive when it comes to automatic further processing of extracted annotations in order to store them. The string the annotation covers will usually not be the string that should be used as the entity identifier. In many cases, the annotation will contain that value as a substring, and the substring may also be associated with another annotation type. For example, if a car annotation covers the string “dark blue Mondeo registration SGR 4RT”, then just the registration mark “SGR 4RT” should be used as the identifier for the car instance. This restriction arises from IE systems typically being used in isolation, without consideration for how their results can be automatically processed beyond displaying the annotated text, other than by code specifically written for each application.

As the right hand side of a JAPE rule can invoke Java code, it would be possible to write code to extract the value from the text and assert an annotation to be stored. However, writing Java for each rule is not suitable either for the interactive and iterative approach of ESTEST, or for its intended eventual use by end-users. Therefore, we have developed an annotation post-processor that automatically identifies annotations of interest from either lookup or pattern-matching named entity recognition, and stores these results automatically in the ESTEST repository, extending the extents of the corresponding elements of the virtual global schema.

These new facilities have been provided as follows:

1) It has been necessary to develop a new control style to automatically process annotations. As mentioned above, JAPE grammars can have one of four control styles: i) `brill` means that should more than one rule match from the current position in the text, then each rule will only fire once over the text covered by the longest match; therefore a second match for a rule covering a smaller portion of the text will be missed; ii) `first` means the first match found fires, allowing the rules' ordering to be used to decide their priority; iii) `appelt` provides a set of criteria to use in deciding which rule to fire, such as giving preference to the match which covers the most text; finally iv) `all` fires all matching rules so that rules that match substrings will fire as many times as they match.

Our pattern processor needs to be able to find all matching annotation types in order to find values to use as identifiers, but use of the `brill` style proved too restrictive as the matching recommences after the longest match and may miss matches in the later part of the text. The `all` style does find the complete set of matches but when there are rules with optional parts, then these result in the rule firing twice; for example, given the pattern "(COLOUR)? CAR" and the text "red car", two matches for the car annotation would be found: "red car" and "car". To overcome this limitation, we have developed a new style: the JAPE grammars use the `all` style, however our pattern processor removes annotations covering substrings of the text covered by other annotations of the same type; thus, in the example above, "car" would be deleted leaving the longest match "red car". As future work, it would be useful to implement this as a new style within the GATE Jape processor as it seems to be of general use.

2) A `schemaElementMatch` can optionally specify the name of another `schemaElementMatch` to use as its identifier; for example, taking the example from Section 7.5, it would be possible now to store the car registration number as the identifier of a car by specifying such an `id_name` in the pattern definition:

```
<schemaElementMatch>
   <name>car</name>
   <schema_object_name>car</schema_object_name>
   <value>(COLOUR)?--(MANUFACTURER)?--(MODEL)--(CAR_REG)</value>
    <id_name>car_reg</id_name>
</schemaElementMatch>
```

The JAPE rule associated with this pattern would now include the `idAnnotationType` feature which would tell ESTEST to find the value of the `car_reg` annotation within the string covered by the `car` annotation:

```
Rule: car
(
(COLOUR)? (SPACE)? (MANUFACTURER)? (SPACE)? (MODEL) (SPACE)?
(CAR_REG)
):car -->
   :car.car = {kind = "car", rule = "known_car0",
               idAnnotationType="car_reg"}
```

3) The `idAnnotationType` feature above enables schema elements that are sources for named entity recognition to be used as identifiers; for example, if a car with a registration mark that is already known was mentioned in the text, the rule would fire. But in order to be able to identify cars not already known, it is necessary to be able to specify a text pattern to match and associate this pattern with a schema element.

For this purpose, a new pattern type, `value_def`, allows a sequence of characters, numerals and punctuation to be specified, e.g. for registration marks:

```
<value_def>
        <name>REGISTRATION_MARK</name>
        <schema_object_name>car_reg</schema_object_name>
        <value_def_part>
            <type>String</type>
            <length>2</length>
        </value_def_part>
        <value_def_part>
            <type>Integer</type>
            <length>2</length>
        </value_def_part>
        <value_def_part>
            <type>Space</type>
            <length>1</length>
        </value_def_part>
         <value_def_part>
            <type>String</type>
            <length>3</length>
         </value_def_part>
   </value_def>
```

This results in the following JAPE macro and rule being created:

```
Macro: REGISTRATION_MARK
(
    ({Token.kind == word, Token.length == "2"})
     ({Token.kind == number, Token.length == "2"})
     ((SPACE))
     ({Token.kind == word, Token.length == "3"})
 )

Rule: REGISTRATION_MARK
(
  (REGISTRATION_MARK)
)
 :REGISTRATION_MARK -->
        :REGISTRATION_MARK.car_reg = {kind = "car_reg", rule =
                "REGISTRATION_MARK", estestStore="yes",
                idAnnotationType="car_reg"}
```

Now when a pattern such as "AA11 AAA" is found in the text, this will create an annotation linked to the `car_reg` schema element. Combining this with the car rule will result in new cars being identified, and stored identified by their registration. In the crime example, this can be seen in the first operation intelligence string processed which contains a reference to a car that is present in `CarsDB`:

```
'GEORGE BUSH HAS A NEW YELLOW CAR REGISTRATION LO78 HYS. IT IS A
FORD MONDEO.'
```

This instance of `car` is identified and stored, including the previously unknown fact that it is yellow:

```
Storing template: <<car>>/LO78 HYS

  Storing template attribute edge
    <<attribute,car,car_reg>> [1,LO78 HYS]
  Storing template attribute edge
    <<attribute,car,colour>>[1,YELLOW]
  Storing template attribute edge
    <<attribute,car,manufacturer>>[1,FORD]
  Storing template attribute edge
    <<attribute,car,model>>[1,MONDEO]
```

Similarly, in the second text processed there is a car mentioned, the registration mark of which was not previously in `CarsDB`:

```
'TONY BLAIR SEEN COMING OUT OF THE PERSEVERANCE PUBLIC HOUSE
DRIVES OFF IN A GREEN FORD PUMA UY22 QWC.'
```

This is stored in the same way, in this case though the HDM edge `<<attribute ,car,car_reg>> [2,UY22 QWC]` will be new:

```
Storing template: <<car>>/UY22 QWC
  Storing template attribute edge
    <<attribute,car,car_reg>> [2,UY22 QWC]
  Storing template attribute edge
    <<attribute,car,colour>>[2,GREEN]
  Storing template attribute edge
    <<attribute,car,manufacturer>>[2,FORD]
  Storing template attribute edge <<attribute,car,model>>[2,PUMA]
```

7.6 Discussion

In this chapter we have described several extensions made to the initial ESTEST system which demonstrate the potential for extending its functionality in a number of research directions:

1) As well as making use of schema element names in schema matching, ESTEST also makes use of textual metadata, such as JDBC remarks and comments in XML, to suggest correspondences. We are aware of no other data integration system that is able to exploit such textual metadata.

2) ESTEST is able to perform coreference resolution by merging templates matching the text. It is then able to compare these with the known structured data and decide if they are new instances or if they represent new attributes about existing instances. We are aware of no other system that combines approaches for resolving duplicate references in both text and structured data. While our attempt at making use of GATE's `OrthoMatcher` component proved ineffective, we believe that there is potential for making use of pronominal coreference resolution for annotation types that refer to people.

3) A further extension has been provided that extracts values from the text strings covered by annotations, for example storing car registration marks to represent instances of cars. When combined with the template merging extension of 2), this provides a method of automatically storing the results of the IE process. Other than the KIM system which relies on an ontology of everything, we are aware of no other IE system which provides general facilities for further processing of the annotations produced. As future work it would be possible to enhance the ESTEST data model to include identifiers for record-based source data models e.g. the primary keys defined in relational tables. In this way, annotations found for the primary key attributes could be automatically used as the `id_name` for the pattern related to the schema element representing the table.

4) Finally, in order to overcome the limitation of requiring the user to recode JAPE rules from stubs on each ESTEST iteration, which was the case in the first version of ESTEST, patterns associated with schema elements can now be stored in the EMR. These patterns have been designed in such a way that they could be created by the end user from the workbench that we describe as possible future work in Chapter 8.

Chapter 8

Conclusions and Future Work

8.1 Summary and Principal Achievements

We now give a brief summary of the thesis and then list the main research contributions made. In Chapter 1 we described the application class we seek to address, namely that of partially structured data. In these applications data is often stored as a combination of structured data and free text. We assert that this reliance on text is for two main reasons: 1) the queries required become known over time, and often after the data is captured, and 2) there are limitations in the support of dynamically evolving schemas in conventional DBMSs. We outlined our approach, which is based on combining data integration and IE techniques, making use of the virtual global schema constructed by data integration to assist in the configuration of an IE process, and then using IE to semi-automatically extract new structured data from text which is automatically integrated within the global schema. Our ESTEST prototype has been developed to demonstrate this approach and to provide a test-bed for further research.

Chapter 2 gives an overview of: the facilities provided by DBMSs for exploiting text; semi-structured data; data on the web; and data integration systems, in particular focusing on schema matching approaches based on textual schema element names. We also describe how conventional record-based database systems have limited support for the dynamic evolution of schemas, and we suggest that graph-based data models are a more appropriate foundation for ESTEST.

An overview of IE is given in Chapter 3, including its place in the wider field of Natural Language Processing, and a description of the tasks undertaken by classic IE systems, such as those which were built for the MUC competitions. A description of the facilities offered by the GATE IE system is given, including its language engineering design, with its goal of reuse and extensibility of common IE components making it a good choice for constructing the IE process in our prototype system. Recent developments in IE which are related to, and have happened in parallel to, our research, are then described, especially semantic annotation which, rather than assigning entity types to annotations, links them to a concept in an ontology.

Chapter 4 reviews the AutoMed data integration system, in particular its BAV approach to data integration, which is well-suited to the dynamic schema extension capabilities required by the ESTEST system. We describe the extensions we made to AutoMed in order to support the development of ESTEST, in particular:

- In the class of applications that ESTEST targets, ontologies are an increasingly common data source. Therefore, we added RDF/ RDFS as a data model supported by AutoMed.
- ESTEST requires a repository for the data extracted by its IE process. At the time of ESTEST's development, the AutoMed query processing facilities were still to be developed, therefore we developed a store for native HDM data to meet this need and also to investigate the practicalities of HDM as a data model.
- In order to be able to access heterogeneous data sources, we contributed to the development of the AutoMed wrapper architecture in general, and specifically built wrappers for RDF / RDFS and our native HDM store.

In Chapter 5 we describe the design of ESTEST, which iterates through a number of steps: integrating available data sources to construct a global schema and metadata repository; using these to semi-automatically configure an IE process; executing the IE process over the textual part of the data; automatically integrating the extracted information into the global schema; supporting queries encompassing the new data;

and then including new structured data sources. ESTEST currently runs from scripts, but has been designed with the intention that the input it requires and commands to its components could equally be supplied by the front-end that we discuss in Section 8.3.

Chapter 6 shows the system in use in the Road Traffic Accident domain and begins with an overview of the characteristics of the data and query requirements of such applications. We give an example of ESTEST in use, combining an RDF / RDFS ontology of the causes of accidents with a relational database of accident reports (the complete output from ESTEST is listed in Appendix B). In Section 6.3 we present an evaluation of ESTEST processing real RTA reports and show that the results, measured in terms of recall and precision, are in line with a vanilla IE system, while ESTEST is also able to combine additional evidence from the structured data with that from the extracted data.

Several extensions made to the initial version of ESTEST are described in Chapter 7 and are illustrated by reference to an example from the crime application domain. In this example, a structured database of cars known to the police is integrated with operational intelligence gathered by police officers on patrol (the complete output from ESTEST running this example is given in Appendix C). The first of the extensions to ESTEST enables textual descriptions of schema elements, automatically obtained from the metadata of the data sources and subsequently processed by an IE process, to be used to provide evidence for candidate correspondences between schema elements. We also explore the potential for making use of WordNet semantic distance metrics for this purpose. Our second extension exploits the similarity between the task of detecting duplicates in databases with that of coreference resolution in IE. Our attempt to reuse standard nominal coreference components was not successful, and instead we have developed an alternative method of detecting coreference in text based on the creation of templates. The resulting templates are then compared to the already known structured data, and the extracted information is either merged with that or added as new data. Our third extension enables patterns representing occurrences of schema elements found in text to be stored in the ESTEST metadata repository, and these are used to automatically generate the necessary input to the GATE process, in particular the JAPE rules. Our fourth extension makes use of schema information to decide which value to extract from text in order to represent an annotation, for example to represent a car by its registration mark. When these last three extensions are taken together, they provide a generally applicable method of automatically processing annotations extracted from text.

The extended version of ESTEST makes the following principal research contributions:

- Data Integration systems have not previously been able to integrate information that is stored as unstructured text with other structured and semi-structured data sources.
- IE is the task of finding pre-defined entities in text. For the first time, ESTEST provides IE with a general application-independent way of defining those entities by building a virtual global schema which combines available data sources, regardless of their data model. This global schema, and the associated metadata repository, is used to assist in the semi-automatic configuration of the IE process.
- The schema-matching component of ESTEST is novel in that it is the first, to our knowledge, to make use of not only schema element names and abbreviations, but also the textual descriptions found in the metadata of many commonly used data models.
- We have combined, for the first time, approaches to deal with the detection of duplicate entity references in both free text and structured data, including a novel approach to co-reference detection in text, based on creating templates from annotations found in text.

- ESTEST has demonstrated a new method of automatically processing the annotations found as a result of IE processing. A mechanism for specifying the correct identifiers for annotations can be combined with the templates created and compared to the structured data. Together, these provide a general-purpose method for processing annotations.

A number of challenges were encountered during the work described in this thesis:

- As it was our intention to contribute to new techniques combining IE and data integration for the first time, we made use where possible of existing research software, namely AutoMed and GATE. While both of these are good examples of research groups producing useful open-source software, in using both there were bugs to find and overcome, and backwards-compatibility issues with the frequent new versions required significant effort, which became a distraction from our research effort, detracting from the time available for our research effort.
- It was necessary to develop a prototype system supporting our whole end-to-end approach before being able to focus our investigations on further particular research areas. After the development of a first version of ESTEST, and its validation in the RTA domain, we used this version as a testbed to concentrate on further research issues arising from limitations in specific phases of the approach.
- The lack of any other competing system for partially structured data created difficulties in evaluating our approach. Therefore, we have demonstrated that ESTEST can operate as effectively as a "vanilla" IE system but also that it can support queries that an IE system would not. In Section 8.3 below we discuss the steps required for a complete evaluation of our approach.

An outline of our approach was published in [Williams, D. and Poulovassilis, A. 2003], the design of ESTEST from Chapter 5 was published in [Williams, D. 2005; Williams, D. and Poulovassilis, A. 2004], the RTA example and experimental results from Chapter 6 were published in [Williams, D. and Poulovassilis, A. 2006], and the extensions from Chapter 7 were published in [Williams, D. and Poulovassilis, A. 2008].

8.2 General Applicability of our Approach

In this thesis we have described a specific prototype system demonstrating our approach, developed using facilities provided by an existing data integration system (AutoMed) and IE system (GATE). We demonstrated the potential of using our ESTEST prototype via examples from the road traffic accident and crime domains — although to prove the value to end-users of the system an evaluation is required following development of an end-user workbench as described in Section 8.3.

However, our approach is not dependent on these specific AutoMed and GATE systems, nor do we believe its potential is restricted to the road traffic accident and crime domains as we now discuss:

8.2.1 Other Application domains

While conducting the research for this thesis, we interviewed domain experts from a varied range of applications that use a combination of structured data and text. A common theme of these applications is that they have experts whose main role is to meet information needs on behalf of a wider user-base, and it is such information-providers who we envisage as the eventual end-users of ESTEST. Our discussions with such domain experts were used to provide anecdotal evidence to support of our assumptions about the reasons for data being stored as text, and how such data is used:

1) In the Road Traffic Accident domain, described more fully in Section 6.1, the schema has evolved through its use over decades, yet new queries still arise and have to be answered, often by reference to the text. Information experts at the UK Department of Transport specialise in answering such queries, in response to questions raised by government department or elected representatives. Changes to

the structured part of the data are the subject of Government review and so happen infrequently. In addition to discussions with experts at the UCL Centre for Transport Studies, we also discussed the commercially available software available in this domain with developers at the UK's leading vendor.

2) The Resource Information Service [RIS] is a charity providing the LINK database about services for homeless people used by over 2000 staff at 75 homelessness organisations. Managing and using textual data about these services provided by these organisations to RIS is problematic, and the ability to progressively merge this data into the structured database over time is seen as an important feature that cannot be provided by the technology currently being used. The RIS LINK project manager provided us with the details of the IT director at one of their client organisations, and we held discussions with them on the use of the data provided.

3) The British Sub Aqua Club collates information on all scuba-diving accidents within and off the coast of the UK. A standard form combines structured information such as the depths and length of decompressions stops, contributing factors such as low visibility, and the diving equipment involved. A textual description of the accident is also captured. Statistics of these accidents are used to inform training and safe diving procedures. Discussions held with the BSAC staff member responsible for collecting and using these statistics revealed that although the text is captured on the form as given by the person involved in the accident, when entered into the system a set of rules is used in order to make scanning for keywords more successful. For example, the various terms for the buoyancy control device, such as "BCD", "Stab Jacket", "Stability Jacket", are all translated into a single term. When new questions arise, the text is scanned for keywords to answer these queries. For example, a recent rise in the number of people who drown when they could have reached the surface had they jettisoned their weight belt was identified, and it is believed to be an overreaction to the recent emphasis on the dangers of decompression due to rapid assents. The BSAC staff member responsible for the system used suggested that

standardising the text from the accident reports was only possible because one person was, just, able to manage the number of reports involved and could ensure the text was coded in a sufficiently consistent way to be of later use.

4) In Bioinformatics, the Swiss-Prot database [Bairoch, A., Boeckmann, B. et al. 2004] has a predefined flat-file format which includes a field for comments. As new requirements arise, this field is used rather than undertaking the large amount of effort that would be required to change the schema, and over time this text becomes increasingly important.

We are also aware of a number of similar applications in the finance sector, for example the production and analysis of investment banking research materials, the documentation associated with setting up new clients, and the monitoring of news feeds such as that from Reuters.

Therefore, we believe that the range of applications that our approach could be evaluated against are both varied and numerous. The applications we have examined have confirmed our assertion that the reason for the dependence on the use of text is due to the need to capture information for as yet unknown queries and the difficulty of dynamically evolving the schemas of conventional databases.

8.2.2 Using other Data Integration and Information Extraction Software

ESTEST was developed using facilities provided for data integration by AutoMed, and for IE by GATE. However, while the characteristics of these systems make them particularly suitable for use in ESTEST, our approach would be able to be supported by other data integration and IE systems as well.

The main benefit of systems based on LAV data integration, for example IM [Levy, A., Rajaraman, A. et al. 1996] and Agora [Manolescu, I., Florescu, D. et al. 2001], is that it is straightforward to add a new data source — provided the virtual global schema does not need to change as a result. However, it is a characteristic of the application classes that we have targeted that the global schema does change, as a result of new data sources being added to support new query requirements. Although AutoMed is a BAV system, ESTEST only makes use of its GAV features, in particular the `add`, `extend` and `rename` steps, in building pathways from source schemas to the global schema. Therefore, other GAV systems such as TSIMMIS [Chawathe, S.S., Garcia-Molina, H. et al. 1994], InterViso [Templeton, M., Henley, H. et al. 1995] and GARLIC [Roth, M.T. and Schwarz, P. 1997] and IBM's Websphere [Websphere], are more readily applicable to the needs of ESTEST then LAV systems. As GLAV subsumes GAV, GLAV systems such as coDB [Franconi, E., Kuper, G.M. et al. 2004] would also be applicable.

With regards to IE, GATE is the leading academic language-engineering platform available and currently includes over 150 components developed both by the GATE team at Sheffield University and contributed by collaborators around the world. However, there are other systems which could also be used to provide the IE processing requirements of ESTEST, principally IBM'S open-source UIMA project [Ferrucci, D. and Lally, A. 2004] for developing unstructured information processing software.

8.3 Future Work

ESTEST is the first implementation of our approach to more effective support of partially structured data applications, and it has potential impact in a variety of active research areas including heterogeneous data integration and the semantic web. It can also serve as a test-bed for further investigation in a range of specific research directions that have been mentioned throughout the thesis, for example the use of semantic-distance metrics in natural language ontologies for schema matching, and enhancing our template-based coreference detection method. However, for the immediate future, we see the most benefit from the following next steps:

1) Developing an end-user workbench for ESTEST, which will require the design of a novel user interface.

2) Undertaking a full evaluation of our approach, for which the end-user workbench would be a necessary prerequisite.

3) Refactoring the ESTEST code in order to allow for its more straightforward future extensibility.

ESTEST Workbench. Throughout the development of ESTEST, we have aimed to develop the code with the future development of a user interface in mind. This includes features such as the steps in the current script-based system which allow for changes to settings that simulate user input. The heuristics used throughout, for example type information to suggest possible text sources and sources of named entities, would be of use in such a workbench. The EMR patterns associated with schema elements used to create JAPE rules were designed in such a way as to be able to be specified through a GUI interface by the end user. While WordNet semantic distance is not likely to be able to be fine-grained enough on its own to provide mappings between concepts, it could be used to intelligently order suggestions presented to the user through the front-end.

The development of such a workbench would not merely involve implementation but will require original research effort, since providing such functionality to the end-user would be novel. While both AutoMed and GATE have GUI front-ends, neither is aimed at application end-users, the GATE GUI being intended for linguists who are at least IT literate, and the AutoMed GUI for application developers. The workbench we envisage would be aimed at the end-user of the data and would enable them to make use of the textual data to answer new queries and to evolve the schema of the structured part of the data as their requirements evolve, for example by selecting text which represents an instance of a new entity and then creating a new schema element and linking it into the global schema.

Graphical user interfaces for the crime domain are an active area of both research and commercial activity. The graphical query language developed in [Smith, M.N. and King, P.J.H. 2004] offers pointers for providing a visual representation and querying of graph-based data models, while the commercial product for extracting information from text [Xanalys] does offer the crime investigator an interface for interrogating textual data, but without the capability of extending the schema or the grammar used in the underlying IE process.

Evaluating ESTEST. As ESTEST provides facilities which are not currently provided in any other system, an appropriate evaluation would require a comparison by an end-user over time between ESTEST and whichever workaround is currently used when new query or data requirements arise. With our envisaged end-user workbench, it would be possible to support an application, targeted at a domain such as those we discuss in Section 8.2.1 above, with the aim of supporting the end-user in their comparison of using ESTEST to meet their information needs with the combination of whichever methods they use at present.

Refactoring ESTEST. Finally, there would be benefits to refactoring the current prototype ESTEST code. Appendix A gives an overview of the technical aspects of the ESTEST software. There are a number of areas in the current code and data model which could be simplified and consolidated, particularly in the data structures used at various stages to represent and make use of annotations. Also, neither the latest version of AutoMed nor GATE are being used at present, and there are benefits to each, particularly concerning performance and scalability in the latest AutoMed query processor. Therefore, an upgrade to these latest versions should be included in the refactoring.

As a final observation, we expect that further possibilities of synergies between data integration and IE techniques are likely to arise as a result of a full evaluation of the current ESTEST system, and that the system will be able to provide a sound foundation for the further investigation of these. In particular, our approach to combining text and structured duplicate detection techniques can be further developed by experimenting with the effectiveness of using proximity and more elaborate merging algorithms. Also, our end-to-end method of associating patterns with schema elements, using these to automatically perform the later steps of configuring and processing text by IE, with the results being stored automatically, provides new opportunities for end-user IE tools that do not require programmers or linguists to configure them. Finally, as more comprehensive models for specifying annotations are emerging, for example [Laprun, C., Fiscus, J. et al. 1999], it should be possible to produce representations that are capable of describing entities both as elements in a virtual global schema and as they appear within free text.

Bibliography

Adelberg, Brad (1998). NoDoSE: a tool for semi-automatically extracting structured and semistructured data from text documents. Proceedings of the 1998 ACM SIGMOD international conference on Management of data. Seattle, Washington, United States. ACM Press. pp 283-294. ISBN:0-89791-995-5.

Agichtein, E., Gravano, L., Pavel, J., Sokolova, V. and Voskoboynik, A. (2001). "Snowball: A Prototype System for Extracting Relations from Large Text Collections." SIGMOD RECORD, vol 30, p 612. Association for Computing Machinery.

Amardeilh, F. and Francart, T. (2004). A Semantic Web Portal with HLT Capabilities. Actes du colloque, Veille Stratgique Scientifique et Technologique (VSST2004). Toulouse, France. Vol 2. pp 481-492.

Appelt, Douglas E. (1999). "Introduction to Information Extraction." AI Communications, vol 12 (3), pp 161-172. IOS Press. ISSN:0921-7126.

Autonomy (Website). Autonomy Corporation. http://www.autonomy.com/

Bagga, Amit (1998). Evaluation of Coreferences and Coreference Resolution Systems. First Language Resource and Evaluation Conference (LREC'98). pp 563-566. http://citeseer.ist.psu.edu/306857.html

Bairoch, A, Boeckmann, B, Ferro, S and Gasteiger, E (2004). "Swiss-Prot: Juggling between evolution and stability." Briefings in Bioinformatics, vol 5 (1), pp 39-55. Oxford University Press. ISSN:1467-5463.

Basili, R., Pazienza, M. and Zanzotto, F. (2002). Learning IE patterns: a terminology extraction perspective. Workshop of Event Modelling for Multilingual

Document Linking at LREC 2002.
http://citeseer.ist.psu.edu/basili02learning.html

Beech, D., Malhotra, A. and Rys, M. (1999). A formal data model and algebra for XML, Communication to the W3C.
http://citeseer.ist.psu.edu/beech99formal.html

Berners-Lee, T., Fielding, R. and Masinter, L. (1998). Uniform Resource Identifiers (URI): Generic Syntax, The Internet Engineering Task Force.
http://www.ietf.org/rfc/rfc2396.txt

Berners-Lee, Tim (1999). Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor, Harper San Francisco.
ISBN:1402842937

Bitton, Dina and DeWitt, David J. (1983). "Duplicate Record Elimination in Large Data Files." ACM Transactions on Database Systems, vol 8 (2), pp 255-265.
ISSN:0362-5915.

Boguraev, B., Garigliano, R. and Tait, J (1995). "Editorial." Natural Language Engineering, vol 1 (1), pp 1-7. Cambridge University Press. ISSN:1351-3249.

Bontcheva, K, Tablan, V., Maynard, D and Cunningham, H (2004). "Evolving GATE to Meet New Challenges in Language Engineering." Natural Language Engineering, vol 10 (3/4), pp 349-373. Great Britain. ISSN:1351-3249.

Bontcheva, K. and Cunningham, H. (2003). The Semantic Web: A New Opportunity and Challenge for Human Language Technology. Workshop on Human Language Technology for the Semantic Web and Web Services at ISWC 2003. Florida, USA. http://www.citeseer.ist.psu.edu/642640.html

Boyd, M., McBrien, P.J. and Tong, N. (2002). "The AutoMed schema integration repository." LECTURE NOTES IN COMPUTER SCIENCE, vol 2405 - BNCOD 2002, pp 42-45. Springer-Verlag. ISSN:0302-9743.

Brickley, D and Guha, R.V (2004). RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, W3C. http://www.w3.org/TR/rdf-schema/

Brin, S. (1999). Extracting Patterns and Relations from the World Wide Web. WebDB '98: Selected papers from the International Workshop on The World Wide Web and Databases. Lecture Notes In Computer Science. A. O. Mendelzon P. Atzeni, and G. Mecca (Eds). Springer Verlag. Vol 1590. pp 172-183. ISBN:3-540-65890-4.

Broekstra, J., Kampman, A. and Harmelen, F. van (2002). Sesame: An Architecture for Storing and Querying RDF Data and Schema Information. ISWC 2002: First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002. Proceedings. Lecture Notes in Computer Science. Springer Berlin / Heidelberg. Vol 2342. p 54.

Budanitsky, Alexander and Hirst, Graeme (2001). Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures. Workshop on WordNet and Other Lexical Resources, Second meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-2001). Pittsburgh, USA. http://www.citeseer.ist.psu.edu/budanitsky01semantic.html

Califf, M. E. and Mooney, R. J. (1999). Relational Learning of Pattern-Match Rules for Information Extraction. Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99). Orlando, Florida, USA. pp 328-334.

Chau, Michael, Xu, Jennifer J. and Chen, Hsinchun (2002). Extracting Meaningful Entities from Police Narrative Reports. National Conference for Digital Government Research. Los Angeles, CA, USA. Digital Government Research Center. pp 1-5.

Chawathe, Sudarshan S., Garcia-Molina, Hector, Hammer, Joachim, Ireland, Kelly, Papakonstantinou, Yannis, Ullman, Jeffrey D. and Widom, Jennifer (1994). The TSIMMIS Project: Integration of Heterogeneous Information Sources. Proceedings of the 10th Meeting of the Information Processing Society of Japan.

Chu, Eric, Baid, Akanksha, Chen, Ting, Doan, AnHai and Naughton, Jeffrey (2007). A relational approach to incrementally extracting and querying structure in unstructured data. VLDB '07: Proceedings of the 33rd international conference on very large data bases. Vienna, Austria. VLDB Endowment. pp 1045-1056. ISBN:978-1-59593-649-3.

Codd, E. F. (1970). "A relational model of data for large shared data banks." Communications of the ACM, vol 13 (6), pp 377-387. http://doi.acm.org/10.1145/362384.362685 ISSN:0001-0782.

Cohen, W. W. (1998). "Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Textual Similarity." SIGMOD RECORD, vol 27 (2), pp 201-212. ACM.

Cowie, Jim and Lehnert, Wendy (1996). "Information Extraction." Communications of the ACM, vol 39 (1), pp 80-91. ACM Press. ISSN:0001-0782.

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

Cunningham, H., Maynard, D., Bontcheva, K. and Tablan, V. (2002). GATE: A framework and graphical development environment for robust NLP tools and applications. 40th Anniversary Meeting of the Association for Computational Linguistics.

Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V. and Ursu, C. (2002). The GATE User Guide, University of Sheffield. http://www.gate.ac.uk/sale/tao/index.html

DARPA (Website). Defence Advanced Research Projects Agency. http://www.darpa.mil/

DfT (1999). Instructions for the Completion of Road Accident Report Form STATS19. UK Government Department for Transport. www.collisionreporting.gov.uk/Stats/stats20.pdf

Doan, AnHai, Domingos, Pedro and Levy, Alon Y. (2000). Learning Source Description for Data Integration. WebDB. pp 81-86. http://citeseer.ist.psu.edu/doan00learning.html

Doddington, George, Mitchell, Alexis, Przybocki, Mark, Ramshaw, Lance, Strassel, Stephanie and Weischedel, Ralph (2004). Automatic Content Extraction (ACE) program - task definition and performance measures. Fourth International Conference on Language Resources and Evaluation (LREC). Lisbon, Portugal.

Domingos, P. and Pazzani, M. (1997). "On the optimality of the simple Bayesian classifier under zero-one loss." Machine Learning, vol 29 (2-3), pp 103-130. Kluwer Academic Publishers.Hingham, MA, USA. ISSN:0885-6125.

Elemagarmind, A, Ipeirotis, P and Varykios, V (2007). "Duplicate Record Detection: A Survey." IEEE transactions on knowledge and data engineering, vol 19 (1), IEEE Computer Society.

Fan, H. and Poulovassilis., A. (2004). Schema evolution in data warehousing environments - a schema transformation-based approach. Proceedings of ER'04. Lecture Notes in Computer Science. Vol 3288. pp 639-653. ISBN:978-3-540-23723-5. www.dcs.bbk.ac.uk/~ap/pubs/er04.pdf

Fellbaum, C (Ed.) (1998). WordNet An Electronic Lexical Database, MIT Press. ISBN:0-262-06197-X

Fellegi, I.P. and Sunter, A. B. (1969). "A Theory for Record Linkage." Journal of the American Statistical Association, vol 64 (328), pp 1183-1210. ISSN:0162-1459. http://www.jstor.org/view/01621459/di985900/98p0492d/0

Fensel, Dieter, Harmelen, Frank van, Horrocks, Ian, McGuinness, Deborah L. and Patel-Schneider, Peter F. (2001). "OIL: An Ontology Infrastructure for the Semantic Web." IEEE Intelligent Systems, vol 16 (2), pp 38-45.

Ferrucci, David and Lally, Adam (2004). "UIMA: an architecectural approach to unstructured information processing in the corporate research environment."

Natural Language Engineering, vol 10, pp 327-348. Cambridge University Press.Cambridge.

Florescu, Daniela, Levy, Alon and Mendelzon, Alberto (1998). "Database techniques for the World-Wide Web: a survey." SIGMOD RECORD, vol 27, pp 59-74. ACM Press. ISSN:0163-5808.

Franconi, Enrico, Kuper, Gabriel M., Lopatenko, Andrei and Zaihrayeu, Ilya (2004). Queries and Updates in the coDB Peer to Peer Database System. Proceedings of the Thirtieth International Conference on Very Large Data Bases. Toronto, Canada. Morgan Kaufmann. pp 1277-1280. ISBN:0-12-088469-0. http://www.vldb.org/conf/2004/DEMP7.PDF

Freitag, D. and Kushmerick, N. (2000). Boosted wrapper induction. National Conference on Artificial Intelligence (AAAI). Vol 17. pp 577-583. http://citeseer.ist.psu.edu/article/freitag00boosted.html

Friedman, Marc, Levy, Alon Y. and Millstein, Todd D. (1999). Navigational Plans for Data Integration. IJCAI-99 Workshop on Intelligent Information Integration. Stockholm, Sweden.

Frost, R.A. (1982). "Binary-Relational Storage Structures." The Computer Journal, vol 25 (3), pp 358-367. Heyden & Son Ltd. ISSN:0010-4620.

Gazdar, Gerald (1996). Paradigm merger in natural language processing. Computing Tomorrow: Future Research Directions in Computer Science. Robin Milner and Ian Wand, Cambridge University Press: 88-109.

Gazdar, Gerald and Mellish, Chris (1989). Natural Language Processing in LISP, Addison Wesley. ISBN:0-201-17825-7

Ge, Niyu, Hale, John and Charniak, Eugene (1998). A statistical approach to anaphora resolution. Sixth Workshop on Very Large Corpra. Quebec, Canada.

Graca, Joao, Mamede, Nuno J. and Pereira, Joao D. (2004). A Run-Time Shared Repository for NLP Tools, Spoken Language Systems Lab, INESC-ID, Portugal.

Grishman, Ralph (1998). TIPSTER Text Architecture Design, New York University. www-nlpir.nist.gov/related_projects/tipster/docs/arch31.doc

Grishman, Ralph and Sundheim, Beth (1996). Message Understanding Conference-6: a brief history. 16th Conference on Computational Linguistics. ACM. pp 466-471.

Gruber, Tom (1993). "A translation approach to portable ontologies." Knowledge Acquisition, vol 5 (2), pp 199-220.

Guarino, Nicola (1998). Formal Ontology and Information Systems. FOIS'98. Trento, Italy. IOS Press. pp 3-15. ISBN:9051993994. http://osm.cs.byu.edu/CS652s04/Gua98Formal.pdf

Harris, S. and Gibbins, N. (2003). 3store: Efficient bulk RDF storage. 1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03). http://citeseer.ist.psu.edu/harris03store.html

Harrison, Ann (1997). The Story of the Blob (email thread). http://www.ibphoenix.com/main.nfs?a=ibphoenix&page=ibp_blob_history

Hernandez, M. A. and Stolfo, S. J. (1998). "Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem." Data Mining and Knowledge Discovery, vol 2 (1), pp 9-37.

Heydecker, Ben (2005). Personal Communication to Dean Williams.

Hirst, Graeme (1981). Anaphora in natural language understanding. Berlin, Springer Berlin / Heidelberg. ISBN:0387108580

Hobbs, Jerry R. (1976). "Pronoun Resolution." SIGART Bull., vol 61, p 28. ACM Press.New York, NY, USA. ISSN:0163-5719. http://portal.acm.org/citation.cfm?id=1045283.1045292

Hobbs, Jerry R., Appelt, Douglas, Bear, John, Israel, David, Kameyama, Megumi, Stickel, Mark and Tyson, Mabry (1996). FASTUS: Extracting Information from Natural-Language Texts. Finite State Devices for Natural Language Processing. E. Roche and Y. Schabes.

Horrocks, Ian (2005). OWL: A Description Logic Based Ontology Language. Principles and Practice of Constraint Programming - CP 2005. Lecture Notes in Computer Science. Sitges, Spain. Springer. Vol 3709**.** pp 5-8.

Horrocks, Ian, Patel-Schneider, Peter F. and Harmelen, Frank van (2002). Reviewing the Design of DAML+OIL: An Ontology Language for the Semantic Web. AAAI/IAA. Edmonton, Alberta, Canada. AAAI Press.

Humphreys, K., Gaizauskas, R., Azzam, S., Huyck, C., Mitchell, B., Cunningham, H. and Wilks, Y. (1998). Description of the LaSIE-II system as used for MUC-7. Proceedings MUC-7. University of Sheffield. http://citeseer.ist.psu.edu/article/humphreys98university.html

Jasper, Edgar (2002). Global Query Processing in the AutoMed Heterogeneous Database Environment. Advances in Databases: 19th British National Conference on Databases, BNCOD 19. Lecture Notes in Computer Science. Sheffield, UK. Springer Berlin / Heidelberg. Vol 2405**.** pp 46-49. http://citeseer.ist.psu.edu/jasper02global.html

JWNL (Website). Java WordNet Library. http://www.sourceforge.net/projects/jwordnet

Kameyama, Megumi (1997). Recognizing Referential Links: An Information Extraction Perspective, Technical Report, AI Center, SRI International. http://citeseer.ist.psu.edu/ameyama97recognizing.html

Kashyap, V and Sheth, A (1996). "Semantic and schematic similarities between database objects: a context-based approach." VLDB Journal, vol 5 (4), pp 276-304.

Kent, William (1979). "Limitations of Record-Based Information Models." Transactions on Database Systems (TODS), vol 4 (1), pp 107-131. ACM. ISSN:0362-5915. http://doi.acm.org/10.1145/320064.320070

King, P.J.H. and Small, C. (1991). "Default Databases and Incomplete Information." The Computer Journal: Special issue on Information Systems, vol 34 (3), pp

239-244. ISSN:0010-4620.

http://comjnl.oxfordjournals.org/cgi/reprint/34/3/239

King, Peter, Derakhshan, Mir, Poulovassilis, Alexandra and Small, Carol (1990). TriStarp - An Investigation into the Implementation and Exploitation of Binary Relational Storage Structures pp 64-84.

King, Peter and Poulovassilis, Alexandra (2000). Enhancing database technology to better manage and exploit Partially Structured Data, Technical Report BBKCS-00-14, Birkbeck College, University of London. http://www.dcs.bbk.ac.uk/research/techreps/2000/bbkcs-00-14.pdf

Laprun, Cristophe, Fiscus, Jonathan, Garofolo, Jonh and Pajot, Sylvain (1999). Recent improvements to the Atlas architecture. National Intitute of Standards and Technology, US Government National Institute of Standards and Technology. http://www.itl.nist.gov/iaui/894.01/atlas/download/hlt2002-atlas.pdf

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

Lenzerini, Maurizio (2002). Data Integration: A Theorectical Perspective. PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM Press. pp 247-258. ISBN:1-58113-507-6. http://www.csd.uoc.gr/~hy562/Papers/lenzerini02.pdf

Levy, Alon, Rajaraman, Anand and Ordille, Joann (1996). Querying Heterogeneous Information Sources Using Source Descriptions. VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc. pp 251-262. ISBN:1-55860-382-4.

Manolescu, Ioana, Florescu, Daniela and Kossmann, Donald (2001). Answering XML Queries on Heterogeneous Data Sources. VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases. Morgan Kaufmann

Publishers Inc. pp 241--250. ISBN:1-55860-804-4. http://citeseer.ist.psu.edu/manolescu01answering.html

Marsh, Elaine (1998). TIPSTER Information Extraction Evaluation: The MUC-7 Workshop. Navy Center for Applied Research in Artificial Intelligence.

Maynard, D., Bontcheva, K. and Cunningham, H. (2003). Towards a semantic extraction of named entities. Recent Advances in Natural Language Processing. Bulgaria.

McBride, Brian (2002). "Jena: A Semantic Web Toolkit." IEEE Internet Computing, vol 6 (6), pp 55-59. IEEE Educational Activities Department.Piscataway, NJ, USA. ISSN:1089-7801.

McBride, Brian and Hayes, Patrick (2002). RDF Semantics, W3C Working Draft. http://www.w3.org/TR/2002/WD-rdf-mt-20021112

McBrien, P.J. and Poulovassilis, A. (1999). A Uniform Approach to Inter-Model Transformations. CAiSE'99. Lecture Notes in Computer Science. Springer. Vol 1626. pp 333-348.

McBrien, P.J. and Poulovassilis, A. (2001). A Semantic Approach to Integrating XML and Structured Data Sources. CAiSE'01. Lecture Notes in Computer Science. Springer. Vol 2068. pp 330-345.

McBrien, P.J. and Poulovassilis, A. (2002). Schema Evolution in Heterogeneous Database Architectures, A Schema Transformation Approach. CAiSE'02. Lecture Notes in Computer Science. Vol 2348. pp 484-499.

McBrien, P.J. and Poulovassilis, A. (2003). "Data integration by bi-directional schema transformation rules." Proceedings 19th International Conference on Data Engineering,ICDE'03, pp 227-238. IEEE Computer Society.Los Alamitos, CA, USA. ISSN:1063-6382. http://citeseer.ist.psu.edu/mcbrien03data.html

McEnery, Tony and Wilson, Andrew (1996). Corpus Lingusitics, Edingburgh University Press. ISBN:0748608087

McGuinness, Deborah L., Fikes, Richard, Stein, Lynn Andrea and Hendler, James A. (2003). DAML-ONT: An Ontology Language for the Semantic Web. Spinning the Semantic Web. Dieter Fensel, James A. Hendler, Henry Lieberman and Wolfgang Wahlster, MIT Press.

Milo, T. and Zohar, S. (1998). Using schema matching to simplify heterogeneous data translation. Proc. 24th Int. Conf. Very Large Data Bases, (VLDB). Morgan Kaufmann Publishers Inc. pp 122-133. ISBN:1-55860-566-5. http://citeseer.ist.psu.edu/milo98using.html

Mitkov, Ruslan (1999). Anaphora Resolution: The State of the Art, Wolverhampton: School of Languages and European Studies, University of Wolverhampton. ISBN:1897618034

Mooney, Raymond J. and Nahm, Un Yong (2005). Text Mining with Information Extraction. 4th International MIDP Colloquium. Bloemfontein, South Africa. Van Schaik Pub. South Africa. pp 141-160.

Morton, Thomas (1997). Coreference for NLP Applications. In Proceedings ACL. http://citeseer.ist.psu.edu/morton97coreference.html

Müller, Heiko and Freytag, Johann-Christoph (2003). Problems, Methods, and Challenges in Comprehensive Data Cleansing, Humboldt University Berlin.

Nahm, U.Y. and Mooney, R (2000). Using Information Extraction to Aid the Discovery of Prediction Rules from Text. KDD-2000 Workshop on Text Mining. pp 51-58.

Newcombe, H.B., Kennedy, J.M., Axford, S.J. and James, A.P. (1959). "Automatic Linkage of Vital Records." Science, vol 130, pp 954-959. ISSN:0036-8075.

NIST (Website). National Institute of Standards and Technology. http://www.nist.gov/

Noy, N. F., Sintek, M., Decker, S., Crubezy, M., and, R. W. Fergerson and Musen, M. A. (2001). "Creating Semantic Web Content with Protege-2000." IEEE Intelligent Systems, vol 2 (16), pp 60-71.

Popov, B., Kiryakov, A., Ognyanoff, D., Manov, D., Kirilov, A. and Goranov, M. (2003). Towards semantic web information extraction. ISWC.

Popov, Borislav, Kiryakov, Atanas, Kirilov, Angel, Manov, Dimitar, Ognyanoff, Damyan and Goranov, Miroslav (2003). KIM - Semantic Annotation Platform. Proceedings 2nd International Semantic Web Conference (ISWC2003). Lecture Notes in Artificial Intelligence. Florida, USA. Springer-Verlag. Vol 2870. pp 834-849.

Popov, Borislav, Kiryakov, Atanas, Ognyanoff, Damyan, Manov, Dimitar and Kirilov, Angel (2004). "KIM - a semantic platform for information extraction and retrieval." Natural Language Engineering, vol 10 (3-4), pp 375-392. Cambridge University Press.New York, NY, USA. ISSN:1351-3249.

Poulovassilis, A (1992). "The Implementation of FDL, a Functional Database Language." Computer Journal, vol 35 (2), pp 119-128. Oxford University Press.Oxford, UK. ISSN:0010-4620.

Poulovassilis, A (2001). The AutoMed Intermediate Query Language, AutoMed Project Technical Report. http://www.doc.ic.ac.uk/automed/techreports/query_language.ps

Poulovassilis, A. and McBrien, P.J. (1998). "A general formal framework for schema transformation." Data and Knowledge Engineering, vol 28 (1), pp 47-71.

Rahm, Erhard and Bernstein, Philip A. (2001). "A Survey of Approaches to Automatic Schema Matching." VLDB Journal, vol 10, pp 334-350.

Rich, Elaine and Knight, Kebin (1991). Artificial Intelligence, McGraw-Hill, Inc. ISBN:0-07-100894-2

RIS (Website). Resource Information Service. http://www.ris.org.uk/

Roth, M. Tork and Schwarz, P. (1997). Don't scrap it, wrap it! a wrapper architecture for legacy sources. VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases. Athens, Greece. Morgan Kaufmann Publishers Inc. ISBN:1-55860-470-7.

Salton, Gerard and McGill, Michael (1983). Introduction to Modern Information Retrieval. New York, NY, USA, McGraw-Hill. ISBN:0070544840

Sciences, National Academy of (1999). The Rise of Relational Databases. Funding a Revolution: Government Support for Computing Research, National Academy Press.

Smith, M. N. and King, P. J. H. (2004). "A Database Interface for Link Analysis." Journal of Database Management, vol 16 (1), pp 60-74. Hershey, PA; Idea Group Publishing. ISSN:1063-8016.

STATS20 (Website). Instructions for the Completion of Road Accident Report Form STATS19. http://www.dft.gov.uk/stellent/groups/dft _transstats/documents/page/dft_transstats_505596.pdf

Tan, A.H. (1999). Text mining: The state of the art and the challenges. PAKDD 1999 Workshop on Knowledge Discovery from Advanced Databases. pp 65-70.

Templeton, Marjorie, Henley, Herbert, Maros, Edward and Buer, Darrel J. Van (1995). "InterViso: dealing with the complexity of federated database access." The VLDB Journal, vol 4 (2), pp 287-318. Springer-Verlag New York, Inc. http://dx.doi.org/10.1007/BF01237922

TIPSTER (Website). The TIPSTER text programme. http://www-nlpir.nist.gov/related_projects/tipster/overv.htm

Tristarp (Website). Tristarp Project Homepage. www.dcs.bbk.ac.uk/TriStarp

Uren, Victoria, Cimiano, Philipp, Iria, José, Handschuh, Siegfried, Vargas-Vera, Maria, Motta, Enrico and Ciravegna, Fabio (2006). "Semantic annotation for knowledge management: Requirements and a survey of the state of the art." Web Semantics: Science, Services and Agents on the World Wide Web, pp 14-28.

Websphere (Website). IBM Websphere. http://www-306.ibm.com/software/websphere/

WEKA (Website). Weka ML Wrapper. http://www.cs.waikato.ac.nz/ml/weka/

Wilks, Yorick (1975). "An intelligent analyzer and understander of English." Communications of the ACM, vol 18 (5), pp 264-274. ACM Press.New York, NY, USA. ISSN:0001-0782. http://doi.acm.org/10.1145/360762.360770

Williams, Dean (2005). Combining Data Integration and Information Extraction Techniques. BNCOD'05: Workshop on Data Mining and Knowledge Discovery. Sunderland, UK. University of Sunderland. pp 96-101.

Williams, Dean and Poulovassilis, Alexandra (2003). Combining Data Integration with Natural Language Technology for the Semantic Web. Workshop on Human Language Technology for the Semantic Web and Web Services, at ISWC'03. Sanibel Island, Florida, USA. pp 113-117. http://www.dcs.bbk.ac.uk/~dean/ISWCPoster.pdf

Williams, Dean and Poulovassilis, Alexandra (2004). An Example of the ESTEST Approach to Combining Unstructured Text and Structured Data. DEXA '04: Proc. of the Database and Expert Systems Applications, 15th International Workshop on (DEXA'04). Zaragoza, Spain. IEEE Computer Society. pp 191-195. ISBN:0-7695-2195-9.

Williams, Dean and Poulovassilis, Alexandra (2006). Combining Information Extraction and Data Integration in the ESTEST system. ICSOFT 2006. Setubal, Portugal. CCIS 10, Springer-Verlag. pp 279-292. http://www.dcs.bbk.ac.uk/~dean/22945.pdf

Williams, Dean and Poulovassilis, Alexandra (2008). Combining Data Integration and IE Techniques to support Partially Structured Data. 13th International Conference on Applications of Natural Language to Information Systems, NLDB 2008. London, UK. LNCS 5039, Springer-Verlag. ISBN:978-3-540-69857-9.

Williams, Simon (2002). The Associative Model of Data, Lazysoft Limited. ISBN:1903453011

Winkler, W. (1994). Advanced Methods for Record Linkage. Statistical Research Division, US Bureau of the Census. citeseer.ist.psu.edu/winkler94advanced.html

Winkler, William E. (2006). Overview of Record Linkage and Current Research Directions, Research Report 2006-2, Statistical Research Division, US Census Bureau. http://www.census.gov/srd/papers/pdf/rrs2006-02.pdf

Winograd, Terry (1972). Understanding Natural Language. New York, NY, USA, Academic Press. ISBN:0127597506

Wu, J. and Heydecker, B. (1998). "Natural language understanding in road accident data analysis." Advances in Engineering Software, vol 29 (7-9), pp 599-610. Elsevier Science Ltd.Oxford, UK. ISSN:0965-9978. http://dx.doi.org/10.1016/S0965-9978(98)00025-8

Xanalys (Website). Xanalys Indexer. http://www.xanalys.com/pdf/solutions/xanalysindexer.asp

Appendix A

ESTEST Implementation

The ETSEST system is written in Java using Eclipse as the IDE, Ant for building and deployment and CVS for source control. There are 17,000 lines of ESTEST code, organised into 130 classes within 4 packages. Postgres is used to persist the results in the ESTEST Metadata Repository (EMR), and the relevant DDL is included in the AutoMed distribution and the database created as part of the AutoMed repository.

Below we give some notes on the main classes of the ESTEST code as pointers to navigate through the code. We then describe the tables in the EMR data source.

Package	**Description**
`uk.ac.bbk.dcs.automed.hdmstore` (12 classes)	The HDM store and its Wrapper are implemented in this package, which is the only one yet to be integrated into the AutoMed distribution.
`uk.ac.bbk.estest` (63 classes)	Main package for ESTEST code
`uk.ac.bbk.estest.ie` (28 classes)	ESTEST code for performing IE, including new components for GATE
`uk.ac.bbk.estest.test` (27 classes)	Test classes not used in the normal running of the system.

For each package, the main classes are as follows:

uk.ac.bbk.dcs.automed.hdmstore	
Class	**Description**
`HdmStore`	Main class which allows for the HDM tables to be created, provides the API for populating and querying the HDM Store.
`HdmWrapperFactory`	Creates and returns an `HdmWrapper`. Also passed a wrapper will create an AutoMed schema.
`HdmWrapper`	Passed IQL queries in AutoMed's ASG representation, will perform retrieve, insert & delete queries.
`LowLevelWrapperHdm`	Initial version of the HDM wrapper implemented while

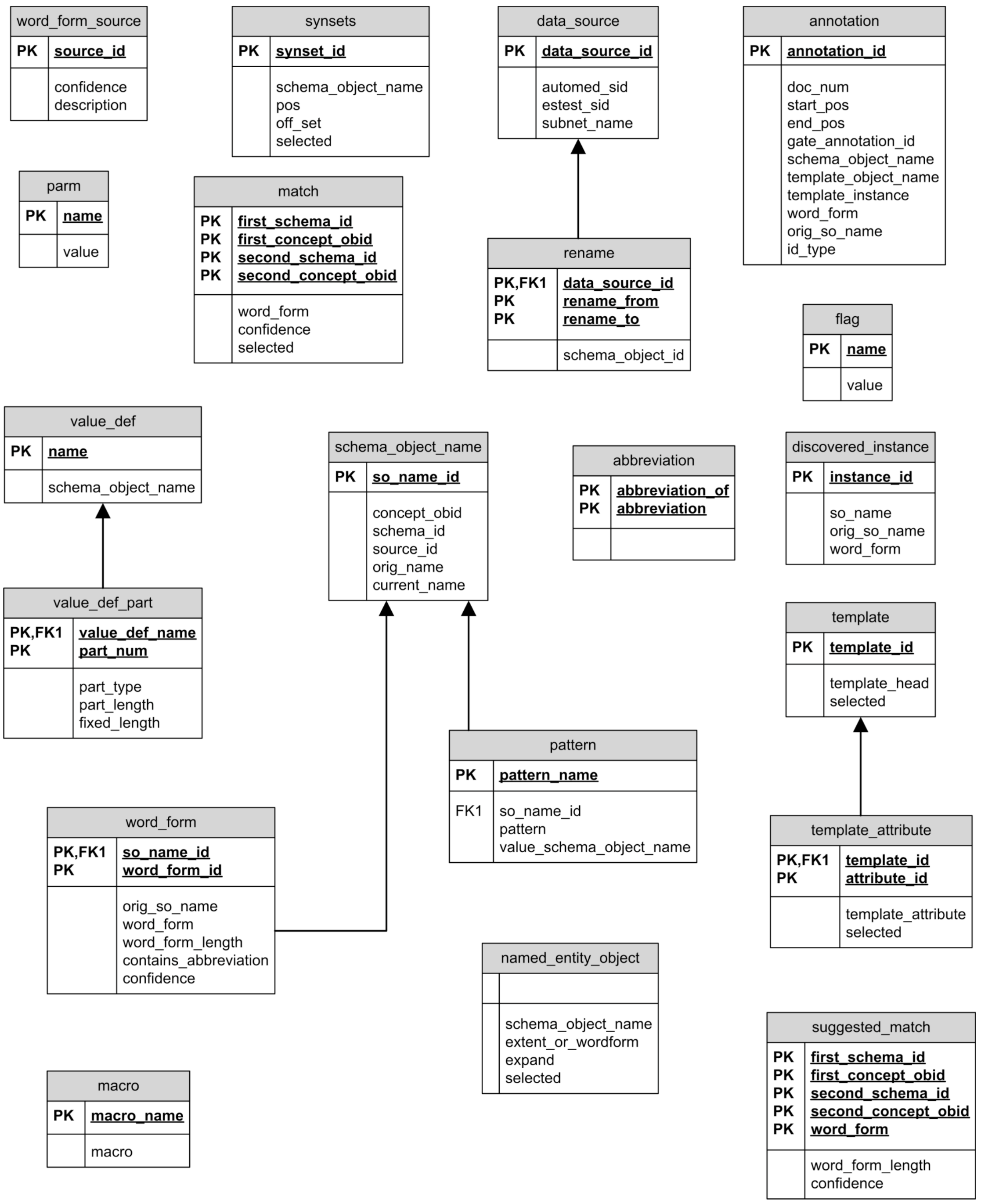
	working on the combined Wrapper / Wrapper Factory architecture. The `HdmWrapper` class converts results from this class into the standard ASG format.

uk.ac.bbk.estest	
Class	**Description**
`Estest`	Called passing the name of a script to run, loads that and executes each step in turn.
`EstestConstants`	Interface used to define constants used throughout the system e.g. default file names.
`EstestMetaDataRepository`	All database access is done through this class.
`Script`	Defines the steps ESTEST will execute as a collection of `ScriptStep`
`Integrator`	Controls the integration of data sources, creates the HDM store, finds correspondences and creates the virtual global schema.
`ConfigIE`	Generates configuration files for GATE components.
`ParameterDefintionReader`	Reads an XML file containing settings and configration details for ESTEST phases.
`ResultIntegrator`	Processes IE annotations to automatically store the results in the HDM store.
`Parameter`	Definition of a setting e.g. named entity defintion.
`EstestOntologyWrapperFactory`	Factory to create an ontology wrapper and to add EDM modelling for ontologies.
`EstestOntologyWrapper`	ESTEST Wrapper over an AutoMed wrapper such as the `RdfWrapper`
`RdfWrapper`	AutoMed wrapper for RDF / RDFS data sources.
`RdfWapperfactory`	Factory to create an RDF wrapper.
`DataSource`	Abstract class defining data sources to be integrated.
`OntologyDataSource`	Implementation of `DataSource` for Ontologies.
`RelationalDataSource`	Implementation of `DataSource` for relational RDMBs.
`WordNet`	Wrapper over the Java WordNet library. Also implements distance metric and and ESTEST WordNet API
`Abbreviation`	An abbreviation for a word form in the domain
`CoRefs`	Collection of coreference matches found in a document.
`Match`	Match between two concepts in the global schema

Templates	Processes templates found in the text including building, merging and persisting them.

uk.ac.bbk.estest.ie	
Class	**Description**
InformationExtractor	Builds a GATE pipeline according to configuration files produced by ConfigIE. Runs pipeline of documents and extracts annotations of interest.
SchemaGazetter	Gazetteer component configured from the virtual global schema. Able to obtain named entity lists from the extent of a schema element or from the associated word forms in the EMR.
SchemaNameTokeniser	Tokeniser for schema element names. Recognises common naming conventions e.g. accNumber, account-number
SchemaObjectTextIEProcessor	IE processor for schema element metadata
NamedEntityDefintion	Schema element with an associated list of word forms to match in the text.
AnnotationDetail	Details of a GATE annotation
Pattern	Implementation of an EMR pattern which is used to find references to a specific schema element in text.
Macro	Alternative word sequences for use in other patterns
ValueDef	Pattern specifying a value to be used as an identifier e.g. a car registration mark.

The tables comprising the ETEST Metadata Repository (EMR) are as follows:



Appendix B

ESTEST Output from the

RTA Example

```
Loading class uk.ac.ic.doc.automed.wrappers.TransactSQLWrapper
Loading class uk.ac.ic.doc.automed.wrappers.PostgresWrapper
Loading class uk.ac.ic.doc.automed.wrappers.OracleWrapper
Loading class uk.ac.ic.doc.automed.wrappers.YattaWrapper
Loading class uk.ac.bbk.dcs.automed.xml.wrappers.DOMWrapper
Loading class uk.ac.bbk.dcs.automed.xml.wrappers.SAXWrapper
Loading class uk.ac.bbk.dcs.automed.hdmstore.HdmWrapper
Loading class uk.ac.bbk.estest.RdfWrapper
Loading class uk.ac.bbk.estest.EstestOntologyWrapper
Verbose debugging switched on

---------------------------------------------------------------

ESTEST running from a script.
Using:C:\estest\bin\config\demoScri
 pt.xml
Building Estest modelling language

================================================================
```

```
   DETAILS OF LOADED ESTEST SCRIPT
================================================================

ScriptStep 1: type=Parameters uri C:\estest\bin\config\parmsAnim
 alsAbbrevs.xml

ScriptStep 2: type=Integrate uri =C:\estest\bin\config\dsdx.xml

ScriptStep 3: type=Query

ScriptStep 4: type=Config

ScriptStep 5: type=Parameters uri C:\estest\bin\config\parmsAnim
 alInRoad1.xml

ScriptStep 6: type=ConfigOutput

ScriptStep 7: type=IE

ScriptStep 8: type=Query

ScriptStep 9: type=Parameters uri C:\estest\bin\config\parmsAnim
 alsConfig2.xml

ScriptStep 10: type=ConfigOutput

ScriptStep 11: type=IE

ScriptStep 12: type=Query

================================================================
   PARAMETERS STEP (1)
================================================================

Parameters to be loaded:
  Abbreviation of: accident, is: acc
  Abbreviation of: vehicle, is: veh
  Abbreviation of: description, is: desc

================================================================
   INTEGRATE STEP (2)
================================================================

Loading datasourced from definition
at:C:\estest\bin\config\dsdx. xml
```

```
Building RDF modelling language
RDF Wrapper Factory creating RDF Model Oriented Schema accOnt
schema
Details of schema: accOnt
  RDF subject          <<subject>>
  RDF predicate        <<predicate>>
  RDF object           <<object>>
  RDF triple           <<triple,subject,predicate,object>>
  RDF uri              <<uri>>
  RDF blank            <<blank>>
  RDF literal          <<literal>>


Data Sources To Be Integrated:
  DS 1 is accDB
  DS 2 is accDBx
  DS 3 is accOnt

Creating the AutoMed Schemas.
  RelationalDataSource is building a wrapper for schema
accDBauto

Created AutoMed schema for accDB
  Details of schema: accDBauto
    sql_390 table        <<vehicle>>
    sql_390 column       <<vehicle,acc_ref>>
    sql_390 column       <<vehicle,veh_no>>
    sql_390 column       <<vehicle,veh_reg_no>>
    sql_390 column       <<vehicle,veh_type>>
    sql_390 primary_key  <pky_vehicle,vehicle,
                            <<vehicle,acc_ref>>
                         <<vehicle,veh_no>>>>
    sql_390 table        <<carriageway_hazards>>
    sql_390 column       <<carriageway_hazards,hazard_id>>
    sql_390 column       <<carriageway_hazards,hazard_desc>>
    sql_390 primary_key  <<pky_carr_hazard,
                          carriageway_hazards,<<
                          carriageway_hazards,hazard_id>>>>
    sql_390 table        <<acc>>
    sql_390 column       <<acc,acc_ref>>
    sql_390 column       <<acc,year>>
    sql_390 column       <<acc,road>>
    sql_390 column       <<acc,road_type>>
    sql_390 column       <<acc,hazard_id>>
    sql_390 column       <<acc,acc_desc>>
```

```
    sql_390 primary_key  <<pky_accident,acc,<<acc,acc_ref>>>>
    sql_390 foreign_key  <fky_vehicle_accident,vehicle,
                           <<vehicle,acc_ref>>,acc,
                           <<acc,acc_ref>>>>
    sql_390 foreign_key  fky_accident_hazard,acc,<<acc,hazard_id
                        ,carriageway_hazards,<<carriageway_hazar
                         s,hazard_id>>>>

RelationalDataSource is building a wrapper for schema accDBxauto
  Created AutoMed schema for accDBx
  Details of schema: accDBxauto
    sql_390 table        <<towns>>
    sql_390 column       <<towns,town>>
    sql_390 primary_key  <<pky_town,towns,<<towns,town>>>>
    sql_390 table        <<roads>>
    sql_390 column       <<roads,road>>
    sql_390 column       <<roads,town>>
    sql_390 primary_key  <<pky_road,roads,<<roads,road>>,
                           <<roads,town>>>>
    sql_390 foreign_key  <fky_accident_hazard,roads,
                           <<roads,town>>,towns,
                           <<towns,town>>>>

Creating the ESTEST Model Schemas.
    Finding foreign keys (for isA relationship).
    Finding tables and columns (for concepts).
    Now delete the relational constructs.....
    The Estest oriented schema for this relational data source
is accDBautzzh
    Now find word forms for each schema element.
    Schema element 'vehicle', word forms are : 'vehicle', 'veh'
    Schema element 'veh_no', word forms are : 'veh', 'vehicle',
      'veh no', 'vehicle no', 'no'
    Schema element 'veh_reg_no', word forms are : 'veh',
      'vehicle', 'veh reg', 'vehicle reg', 'reg', 'veh no',
      'vehicle no', 'veh_reg no', 'vehicle reg no', 'reg no',
      'no'
    Schema element 'veh_type', word forms are : 'veh','vehicle',
      'veh type', 'vehicle type', 'type'
    Schema element 'carriageway_hazards', word forms are :
      'carriage way', 'carriageway hazards', 'hazards'
    Schema element 'hazard_id', word forms are : 'hazard',
      'hazard id', 'id'
    Schema element 'hazard_desc', word forms are : 'hazard',
      'hazard desc', 'desc', 'hazard description', 'description'
```

```
   Schema element 'acc', word forms are : 'acc', 'accident'
   Schema element 'acc_ref', word forms are : 'acc',
    'accident', 'acc ref', 'accident ref', 'ref'
   Schema element 'year', word forms are : 'year'
   Schema element 'road', word forms are : 'road'
   Schema element 'road_type', word forms are : 'road', 'road
     type', 'type'
   Schema element 'acc_desc', word forms are : 'acc',
    'accident','acc desc', 'accident desc', 'desc', 'acc
     description', 'accident description', 'description'
   Storing Data Source info and metadata in EMR.....
 Created ESTEST schema for accDB
 Details of schema: accDBautzzh
   Estest concept       <<vehicle>>
   Estest concept       <<vehicle_veh_no>>
   Estest attribute     <<attribute,vehicle,vehicle_veh_no>>
   Estest concept       <<vehicle_veh_reg_no>>
   Estest attribute     <<attribute,vehicle,
                          vehicle_veh_reg_no>>
   Estest concept       <<vehicle_veh_type>>
   Estest attribute     <<attribute,vehicle,vehicle_veh_type>>
   Estest concept       <<carriageway_hazards>>
   Estest concept       <<carriageway_hazards_hazard_id>>
   Estest attribute     <<attribute,carriageway_hazards,
                          carriageway_hazards_hazard_id>>
   Estest concept       <<carriageway_hazards_hazard_desc>>
   Estest attribute     <attribute,carriageway_hazards,
                          carriageway_hazards_hazard_desc>>
   Estest concept       <<acc>>
   Estest concept       <<acc_acc_ref>>
   Estest attribute     <<attribute,acc,acc_acc_ref>>
   Estest concept       <<acc_year>>
   Estest attribute     <<attribute,acc,acc_year>>
   Estest concept       <<acc_road>>
   Estest attribute     <<attribute,acc,acc_road>>
   Estest concept       <<acc_road_type>>
   Estest attribute     <<attribute,acc,acc_road_type>>
   Estest concept       <<acc_acc_desc>>
   Estest attribute     <<attribute,acc,acc_acc_desc>>
   Estest attribute     <<attribute,vehicle,acc>>
   Estest attribute     <<attribute,acc,carriageway_hazards>>

   Finding foreign keys (for isA relationship).
   Finding tables and columns (for concepts).
   Now delete the relational constructs.....
```

```
    The Estest oriented schema for this relational data source
is accDBxautzd
    Now find word forms for each schema element.
    Schema element 'towns', word forms are : 'towns'
    Schema element 'town', word forms are : 'town'
    Schema element 'roads', word forms are : 'roads'
    Schema element 'road', word forms are : 'road'
    Storing Data Source info and metadata in EMR.....

  Created ESTEST schema for accDBx
  Details of schema: accDBxautzd
    Estest concept       <<towns>>
    Estest concept       <<towns_town>>
    Estest attribute     <<attribute,towns,towns_town>>
    Estest concept       <<roads>>
    Estest concept       <<roads_road>>
    Estest attribute     <<attribute,roads,roads_road>>
    Estest attribute     <<attribute,roads,towns>>

 OntologyDataSource is about to create ESTEST Schema.
  found class: http://www.dcs.bbk.ac.uk/~dean/kb/acc1#tree
  found class: http://www.dcs.bbk.ac.uk/~dean/kb/acc1#inanimate
  found class:
http://www.dcs.bbk.ac.uk/~dean/kb/acc1#obstruction
  found class: http://www.dcs.bbk.ac.uk/~dean/kb/acc1#animal
  found class: http://www.dcs.bbk.ac.uk/~dean/kb/acc1#spillage
  found class: http://www.dcs.bbk.ac.uk/~dean/kb/acc1#accident
  About to check for word forms.
  Schema element 'tree', word forms are : 'tree'
  Schema element 'inanimate', word forms are : 'inanimate'
  Schema element 'obstruction', word forms are : 'obstruction'
  Schema element 'animal', word forms are : 'animal'
  Schema element 'spillage', word forms are : 'spillage'
  Schema element 'accident', word forms are : 'accident', 'acc'
  Schema element 'Resource', word forms are : 'Resource'
  Schema element 'bricks', word forms are : 'bricks'
  Schema element 'cat', word forms are : 'cat'
  Schema element 'fox', word forms are : 'fox'
  Schema element 'oak', word forms are : 'oak'
  Created ESTEST schema for accOnt
  Details of schema: accOntEstest
    Estest concept       <<tree>>
    Estest concept       <<inanimate>>
    Estest concept       <<obstruction>>
    Estest concept       <<animal>>
    Estest concept       <<spillage>>
    Estest concept       <<accident>>
    Estest concept       <<Resource>>
    Estest isA           <<isA,accident,Resource>>
    Estest isA           <<isA,spillage,inanimate>>
    Estest isA           <<isA,inanimate,obstruction>>
    Estest isA           <<isA,animal,obstruction>>
    Estest isA           <<isA,tree,inanimate>>
    Estest isA           <<isA,obstruction,Resource>>
    Estest attribute     <<attribute,accident,obstruction>>
    Estest concept       <<bricks>>
    Estest isA           <<isA,bricks,spillage>>
    Estest concept       <<cat>>
    Estest isA           <<isA,cat,animal>>
    Estest concept       <<fox>>
    Estest isA           <<isA,fox,animal>>
    Estest concept       <<oak>>
    Estest isA           <<isA,oak,tree>>

Going to find matches between Schema elements.
The matches are:
  Match with 0.64% confidence on word form ACCIDENT
    Schema 1: accDBautzzh, Concept 1: acc
    Schema 2: accOntEstest, Concept 2: accident
  Match with 0.64% confidence on word form ROAD
    Schema 1: accDBautzzh, Concept 1: acc_road
    Schema 2: accDBxautzd, Concept 2: roads_road
Going to rename matching elemets so they have the same name.
In Integrator getPositionOfSchema looking for sid 89
  checking 89, accDBautzzh
  ........ and its a match
In Integrator getPositionOfSchema looking for sid 105
  checking 89, accDBautzzh
  checking 104, accDBxautzd
  checking 105, accOntEstest
  ........ and its a match
 Renamed :accOntEstest, accident  to acc
In Integrator getPositionOfSchema looking for sid 89
  checking 89, accDBautzzh
  ........ and its a match
In Integrator getPositionOfSchema looking for sid 104
  checking 89, accDBautzzh
  checking 104, accDBxautzd
  ........ and its a match
 Renamed :accDBxautzd, roads_road  to acc_road
```

```
Extend to match schemas.
 accDBautzzh extending to match accDBxautze
  new extended schema is :accDBautzzn
 accDBautzzn extending to match accOntEstest1a
  new extended schema is :accDBautzzzi
 accDBxautze extending to match accDBautzzh
  new extended schema is :accDBxautzzc
 accDBxautzzc extending to match accOntEstest1a
  new extended schema is :accDBxautzzx
 accOntEstest1a extending to match accDBautzzh
  new extended schema is :accOntEstest1y
 accOntEstest1y extending to match accDBxautze
  new extended schema is :accOntEstest1ze
Assert Identity Transformations between the extended schemas.
 Asserting ID transformation between accDBautzzzi & accDBxautzzx
 Asserting ID transformation between accDBautzzzi &
accOntEstest1ze

About to create HDM store copy of global schema.
Building the AutoMed HDM model
Creating HDM Store estest_store
Creating transormation pathway from HDM model to ESTEST model
global schema
Materialising isA relationships.
  Contents of the IsaFunctionList are:
    <<bricks>>        <<bricks>>
    <<spillage>>      <<spillage>> ++ <<bricks>>
    <<cat>>           <<cat>>
    <<animal>>        <<animal>> ++ <<cat>> ++ <<fox>>
    <<fox>>           <<fox>>
    <<oak>>           <<oak>>
    <<tree>>          <<tree>> ++ <<oak>>
    <<acc>>           <<acc>>
    <<Resource>>      <<Resource>> ++ <<acc>> ++ <<obstruction>>
                      ++ <<animal>> ++ <<cat>> ++ <<fox>> ++
                      <<inanimate>> ++ <<spillage>> ++
                      <<bricks>> ++ <<tree>> ++ <<oak>>
    <<inanimate>>     <<inanimate>> ++ <<spillage>> ++
                      <<bricks>> ++ <<tree>> ++ <<oak>>
    <<obstruction>>   <<obstruction>> ++ <<animal>> ++ <<cat>>
                      ++ <fox>> ++ <<inanimate>> ++
                      <<spillage>> ++ <<bricks>> ++ <<tree>> ++
                      <<oak>>
  Integrator loadDef attempting ident with HDM Store
Global Schema is complete, schema name is: accDBautzzzi
```

```
  Details of schema: accDBautzzzi
    Estest concept       <<vehicle>>
    Estest concept       <<vehicle_veh_no>>
    Estest attribute     <<attribute,vehicle,vehicle_veh_no>>
    Estest concept       <<vehicle_veh_reg_no>>
    Estest attribute
<<attribute,vehicle,vehicle_veh_reg_no>>
    Estest concept       <<vehicle_veh_type>>
    Estest attribute     <<attribute,vehicle,vehicle_veh_type>>
    Estest concept       <<carriageway_hazards>>
    Estest concept       <<carriageway_hazards_hazard_id>>
    Estest attribute
<<attribute,carriageway_hazards,carriage
                         way_hazards_hazard_id>>
    Estest concept       <<carriageway_hazards_hazard_desc>>
    Estest attribute
<<attribute,carriageway_hazards,carriage
                         way_hazards_hazard_desc>>
    Estest concept       <<acc>>
    Estest concept       <<acc_acc_ref>>
    Estest attribute     <<attribute,acc,acc_acc_ref>>
    Estest concept       <<acc_year>>
    Estest attribute     <<attribute,acc,acc_year>>
    Estest concept       <<acc_road>>
    Estest attribute     <<attribute,acc,acc_road>>
    Estest concept       <<acc_road_type>>
    Estest attribute     <<attribute,acc,acc_road_type>>
    Estest concept       <<acc_acc_desc>>
    Estest attribute     <<attribute,acc,acc_acc_desc>>
    Estest attribute     <<attribute,vehicle,acc>>
    Estest attribute     <<attribute,acc,carriageway_hazards>>
    Estest concept       <<towns>>
    Estest concept       <<towns_town>>
    Estest attribute     <<attribute,towns,towns_town>>
    Estest concept       <<roads>>
    Estest attribute     <<attribute,roads,towns>>
    Estest attribute     <<attribute,roads,acc_road>>
    Estest concept       <<tree>>
    Estest concept       <<inanimate>>
    Estest concept       <<obstruction>>
    Estest concept       <<animal>>
    Estest concept       <<spillage>>
    Estest concept       <<Resource>>
    Estest isA           <<isA,spillage,inanimate>>
    Estest isA           <<isA,inanimate,obstruction>>
```

```
    Estest isA           <<isA,animal,obstruction>>
    Estest isA           <<isA,tree,inanimate>>
    Estest isA           <<isA,obstruction,Resource>>
    Estest concept       <<bricks>>
    Estest isA           <<isA,bricks,spillage>>
    Estest concept       <<cat>>
    Estest isA           <<isA,cat,animal>>
    Estest concept       <<fox>>
    Estest isA           <<isA,fox,animal>>
    Estest concept       <<oak>>
    Estest isA           <<isA,oak,tree>>
    Estest isA           <<isA,acc,Resource>>
    Estest attribute     <<attribute,acc,obstruction>>

================================================================
   QUERY STEP (3)
================================================================

About to run IQL query. Schema:accDBautzzzi,
query:<<attribute,ac
 c,obstruction>>
Connecting to HdmStore for schema: estest_store
  The query was <<attribute,acc,obstruction>>
  Results: []

================================================================
   CONFIGIE STEP (4)
================================================================

Generating Suggestions for Named Entity.
Suggested possible NE List is:
  Possible Extent NE object - Data Source: 1, Schema
    Object<<acc_acc_ref>>
  Possible Extent NE object - Data Source: 1, Schema
    Object<<acc_road_type>>
  Possible Extent NE object - Data Source: 2, Schema
    Object<<towns_town>>
  Possible Extent NE object - Data Source: 2, Schema
    Object<<roads_road>>
Identifing Text Sources.
Finding Templates.
  Template: <<roads>>
    Attribute: <<towns>>
    Attribute: <<acc_road>>
  Template: <<acc>>
```

```
    Attribute: <<acc_acc_ref>>
    Attribute: <<acc_year>>
    Attribute: <<acc_road>>
    Attribute: <<acc_road_type>>
    Attribute: <<acc_acc_desc>>
    Attribute: <<carriageway_hazards>>
    Attribute: <<obstruction>>
  Template: <<vehicle>>
    Attribute: <<vehicle_veh_no>>
    Attribute: <<vehicle_veh_reg_no>>
    Attribute: <<vehicle_veh_type>>
    Attribute: <<acc>>
  Template: <<carriageway_hazards>>
    Attribute: <<carriageway_hazards_hazard_id>>
    Attribute: <<carriageway_hazards_hazard_desc>>
  Template: <<towns>>
    Attribute: <<towns_town>>
ESTEST is set NOT to wait for user confirmation of results.

================================================================

   PARAMETERS STEP (5)
================================================================

Parameters to be loaded:
  Named Entity Parameter: animal is wordform based and schema
    expansion is selected
  Named Entity Parameter: acc_road is extent based and no
    expansion is selected
  Named Entity Parameter: obstruction is wordform based and
    schema expansion is selected

================================================================
   CONFIG OUTPUT GENERATION STEP (6)
================================================================

Expanding the selected Named Entity schema elements.
Expanding word forms from schema for <<animal>>
Expanding word forms from schema for <<obstruction>>
  OBSTRUCTION, OBSTRUCTION, INANIMATE, INANIMATE, SPILLAGE
  SPILLAGE, BRICKS, BRICKS, TREE, TREE, OAK, OAK, ANIMAL
Generating the Information Extraction JAPE input file
  Macro: acc_acc_ref
   ({Lookup.minorType == acc_acc_ref})
```

```
Rule: acc_acc_ref
(
(acc_acc_ref)
)
:acc_acc_ref -->
 :acc_acc_ref.acc_acc_ref = {kind ="acc_acc_ref", rule =
    "acc_acc_ref"}


Macro: acc_road_type
 ({Lookup.minorType == acc_road_type})


Rule: acc_road_type
(
(acc_road_type)
)
:acc_road_type -->
 :acc_road_type.acc_road_type = {kind ="acc_road_type", rule =
     "acc_road_type"}


Macro: towns_town
 ({Lookup.minorType == towns_town})


Rule: towns_town
(
(towns_town)
)
:towns_town -->
 :towns_town.towns_town = {kind ="towns_town", rule =
    "towns_town"}


Macro: roads_road
 ({Lookup.minorType == roads_road})


Rule: roads_road
(
(roads_road)
)
:roads_road -->
```

```
    :roads_road.roads_road = {kind ="roads_road", rule =
       "roads_road"}


  Macro: animal
    ({Lookup.minorType == animal})


  Rule: animal
  (
  (animal)
  )
  :animal -->
   :animal.animal = {kind ="animal", rule = "animal"}


  Macro: acc_road
    ({Lookup.minorType == acc_road})


  Rule: acc_road
  (
  (acc_road)
  )
  :acc_road -->
   :acc_road.acc_road = {kind ="acc_road", rule = "acc_road"}


  Macro: obstruction
    ({Lookup.minorType == obstruction})


  Rule: obstruction
  (
  (obstruction)
  )
  :obstruction -->
   :obstruction.obstruction = {kind ="obstruction", rule =
    "obstruction"}


Created IE input file ie.jape

================================================================
   INFORMATION EXTRACTION STEP (7)
```

```
================================================================

Initialising Gate using Gate Home :C:\Program Files\GATE 3.0
Using C:\Program Files\GATE 3.0 as GATE home
Using C:\Program Files\GATE 3.0\plugins as installed plug-ins
directory.
Using C:\Program Files\GATE 3.0\gate.xml as site configuration
file.
Using C:\Documents and Settings\dean\gate.xml as user
configuration file
CREOLE plugin loaded: file:/C:/Program Files/GATE-
3.1b1/plugins/ANNIE/
Registering Creole directories:
 file:/C:/estest
CREOLE plugin loaded: file:/C:/estest/
Creating Default Tokeniser Gate Processing Resource.
Creating Sentence Splitter Gate Processing Resource.
Creating Database Gazetteer Gate Processing Resource.
  Configuring Database Gazetteer using file:/C:/estest/dbGaz.xml
  Named Entity source to be loaded is accDBautzzzi:obstruction
  Named Entity source to be loaded is accDBautzzzi:acc_road
  Loaded 10 values for word-form NE object <<obstruction>>
    About to run IQL query. Schema:accDBautzzzi, query:distinct
    <<acc_road>>
  Loaded 5 values for extent-based NE object <<acc_road>>
  Reading accDBautzzzi:obstruction
  Reading accDBautzzzi:acc_road
Creating Jape Transducer Gate Processing Resource.
JAPE URL: file:/C:/estest/ie.jape
Assembling Components Into Pipeline.
Gate is now initialised and the ESTEST application is built.
No JAPE URI specified - default will be used
Configuring Database Gazetteer using file:/C:/estest/dbGaz.xml
Named Entity source to be loaded is accDBautzzzi:obstruction
Named Entity source to be loaded is accDBautzzzi:acc_road
Loaded 10 values for word-form NE object <<obstruction>>
  About to run IQL query. Schema:accDBautzzzi, query:distinct
    <<acc_road>>
Loaded 5 values for extent-based NE object <<acc_road>>
Reading accDBautzzzi:obstruction
Reading accDBautzzzi:acc_road
About to run IQL query. Schema:accDBautzzzi,
query:<<attribute,ac
 c,acc_acc_desc>>
```

```
Added doc FOX RUNS INTO ABBEY STREET CAUSING V1 TO SWERVE
VIOLENTLY AND LEAVE ROAD OFFSIDE 50M AWAY
template instance is A001234
Added doc A50 WELFORD ROAD LEICESTER,BRIDGE 200 YDS S ROMAN WAY.
 V1 TRAV MOTORWAY M6 FAILS TO STOP AT XRDS AND HITS V2 TRAV
UPPERTON RD V2 THEN HITS V3 PKD ON OS OF UPPERTON RD
template instance is B231562
Added doc ESCAPED KANGAROO JUMPS IN FRONT OF V1
template instance is C051633

----------------------------------------------------------------

Document to be processed by IE : 'FOX RUNS INTO ABBEY STREET
CAUS
 ING V1 TO SWERVE VIOLENTLY AND LEAVE ROAD OFFSIDE 50M AWAY'
Running processing resources over corpus...

      Annotations:
   ......Annotation Set jape
Anotation type : obstruction rule/obstruction kind/obstruction
adding FOX/<<obstruction>>
Anotation type : acc_road rule/acc_road kind/acc_road
adding ABBEY STREET/<<acc_road>>
Annotation Details:
  Schema element = '<<obstruction>>', value = 'FOX' and the ID
    Will be generated.
  Schema element = '<<acc_road>>', value = 'ABBEY STREET' and
the
    ID will be generated.
adding node to HDM Store with generated id <<fox>>
[estestInstanc
 e1]
adding edge <<isA,fox,animal>>[A001234,estestInstance1]
adding edge <<isA,animal,obstruction>>[A001234,estestInstance1]
adding template attribute edge
<<attribute,acc,obstruction>>[A001
 234,estestInstance1]
adding node to HDM Store with generated id <<acc_road>>
[estestIn
 stance2]
adding template attribute edge
<<attribute,acc,acc_road>>[A001234
 ,estestInstance2]

----------------------------------------------------------------
```

```
Document to be processed by IE : 'A50 WELFORD ROAD
LEICESTER,BRIDGE 200 YDS S ROMAN WAY. V1 TRAV MOTORWAY M6 FAILS
TO STOP AT XRD S AND HITS V2 TRAV UPPERTON RD V2 THEN HITS V3
PKD ON OS OF UPPERTON RD'
Running processing resources over corpus...

      Annotations:
   ......Annotation Set jape
Annotation Details:

----------------------------------------------------------------

Document to be processed by IE : 'ESCAPED KANGAROO JUMPS IN
FRONT OF V1'
Running processing resources over corpus...

      Annotations:
   ......Annotation Set jape
Annotation Details:

================================================================
   QUERY STEP (8)
================================================================

About to run IQL query. Schema:accDBautzzzi,
query:<<attribute,ac
 c,obstruction>>
  The query was <<attribute,acc,obstruction>>
  Results: [{A001234 ,estestInstance1 }]

================================================================
   PARAMETERS STEP (9)
================================================================

ParameterDefintionContentHandler & its the end of a end of a syn
 set
offSetString is :1780968
offSet is :1780968
Parameters to be loaded:
  Synset Parameter: animal points to synset  1780968
  Named Entity Parameter: animal is wordform based and word net
e
   xpansion is selected
  Named Entity Parameter: obstruction is wordform based and
schem
   a expansion is selected

================================================================
   CONFIG OUTPUT GENERATION STEP (10)
================================================================

Expanding the selected Named Entity schema elements.
Initialising Word Net with init file: file_properties.xml
 INFO [main] (MessageLog.java:62) - Installing dictionary
net.didion.jwnl.dictionary.FileBackedDictionary@54f169
Expanding word forms from WordNet for <<animal>>, found:
  FEMALE MAMMAL, TUSKER, PROTOTHERIAN, METATHERIAN, PLACENTAL
  PLACENTAL MAMMAL, EUTHERIAN, EUTHERIAN MAMMAL
  FOSSORIAL MAMMAL, MONOTREME, EGG-LAYING MAMMAL, MARSUPIAL
  POUCHED MAMMAL, LIVESTOCK, STOCK, FARM ANIMAL, BULL, COW
  YEARLING, BUCK, DOE, INSECTIVORE, AQUATIC MAMMAL, CARNIVORE
  FISSIPEDIA, AARDVARK, ANT BEAR, ANTEATER, ORYCTEROPUS AFER
  BAT, CHIROPTERAN, LAGOMORPH, GNAWING MAMMAL, RODENT, GNAWER
  GNAWING ANIMAL, UNGULATA, UNGULATE, HOOFED MAMMAL
  UNGUICULATA, UNGUICULATE, UNGUICULATE MAMMAL, HYRAX, CONEY
  CONY, DASSIE, DAS, PACHYDERM, EDENTATE, PANGOLIN
  SCALY ANTEATER, ANTEATER, PRIMATE, TREE SHREW, FLYING LEMUR
  FLYING CAT, COLUGO, PROBOSCIDEAN, PROBOSCIDIAN
  PLANTIGRADE MAMMAL, DIGITIGRADE MAMMAL, NAKED MOLE RAT
  DAMARALAND MOLE RAT, ECHIDNA, SPINY ANTEATER, ANTEATER
  ECHIDNA, SPINY ANTEATER, ANTEATER, PLATYPUS, DUCKBILL
  DUCKBILLED PLATYPUS, DUCK-BILLED PLATYPUS
  ORNITHORHYNCHUS ANATINUS, OPOSSUM, POSSUM, OPOSSUM RAT
  BANDICOOT, KANGAROO, PHALANGER, OPOSSUM, POSSUM, WOMBAT
  DASYURID MARSUPIAL, DASYURID, POUCHED MOLE, MARSUPIAL MOLE
  NOTORYCTUS TYPHLOPS, STAG, MOLE, SHREW, SHREWMOUSE, HEDGEHOG
  ERINACEUS EUROPAEUS, ERINACEUS EUROPEAEUS, TENREC, TENDRAC
  OTTER SHREW, POTAMOGALE, POTAMOGALE VELOX, CETACEAN
  CETACEAN MAMMAL, BLOWER, SEA COW, SIRENIAN MAMMAL, SIRENIAN
  PINNIPED MAMMAL, PINNIPED, PINNATIPED, FISSIPED MAMMAL
  FISSIPED, CANINE, CANID, FELINE, FELID, BEAR, VIVERRINE
  VIVERRINE MAMMAL, MUSTELINE MAMMAL, MUSTELID, MUSTELINE
  PROCYONID, FRUIT BAT, MEGABAT, CARNIVOROUS BAT, MICROBAT
  DUPLICIDENTATA, LEPORID, LEPORID MAMMAL, PIKA, MOUSE HARE
  ROCK RABBIT, CONEY, CONY, MOUSE, RAT, MURINE, WATER RAT
  NEW WORLD MOUSE, MUSKRAT, MUSQUASH, ONDATRA ZIBETHICA
  ROUND-TAILED MUSKRAT, FLORIDA WATER RAT, NEOFIBER ALLENI
  COTTON RAT, SIGMODON HISPIDUS, WOOD RAT, WOOD-RAT, HAMSTER
```

```
  GERBIL, GERBILLE, LEMMING, PORCUPINE, HEDGEHOG
  JUMPING MOUSE, JERBOA, DORMOUSE, SQUIRREL, PRAIRIE DOG
  PRAIRIE MARMOT, MARMOT, BEAVER, MOUNTAIN BEAVER, SEWELLEL
  APLODONTIA RUFA, CAVY, MARA, DOLICHOTIS PATAGONUM, CAPYBARA
  CAPIBARA, HYDROCHOERUS HYDROCHAERIS, AGOUTI
  DASYPROCTA AGUTI, PACA, CUNICULUS PACA, MOUNTAIN PACA, COYPU
  NUTRIA, MYOCASTOR COYPUS, CHINCHILLA, CHINCHILLA LANIGER
  MOUNTAIN CHINCHILLA, MOUNTAIN VISCACHA, VISCACHA
  CHINCHILLON, LAGOSTOMUS MAXIMUS, ABROCOME, CHINCHILLA RAT
  RAT CHINCHILLA, MOLE RAT, MOLE RAT, SAND RAT, DINOCERATE
  ODD-TOED UNGULATE, PERISSODACTYL, PERISSODACTYL MAMMAL
  EVEN-TOED UNGULATE, ARTIODACTYL, ARTIODACTYL MAMMAL
  ROCK HYRAX, ROCK RABBIT, PROCAVIA CAPENSIS, ARMADILLO, SLOTH
  TREE SLOTH, MEGATHERIAN, MEGATHERIID, MEGATHERIAN MAMMAL
  MYLODONTID, MYLODON, ANTEATER, NEW WORLD ANTEATER, SIMIAN
  APE, ANTHROPOID, HOMINOID, HOMINID, MONKEY, PROSIMIAN, LEMUR
  TARSIER, PENTAIL, PEN-TAIL, PEN-TAILED TREE SHREW
  CYNOCEPHALUS VARIEGATUS, ELEPHANT, MASTODON, MASTODONT
Expanding word forms from schema for <<obstruction>>
  OBSTRUCTION, OBSTRUCTION, INANIMATE, SPILLAGE, BRICKS, TREE
  OAK, ANIMAL, CAT, FOX, INANIMATE, INANIMATE, SPILLAGE
  BRICKS, TREE, OAK, SPILLAGE, SPILLAGE, BRICKS, BRICKS
  BRICKS, TREE, TREE, OAK, OAK, OAK, ANIMAL, ANIMAL, CAT, FOX
  FEMALE MAMMAL, TUSKER, PROTOTHERIAN, METATHERIAN, PLACENTAL
  PLACENTAL MAMMAL, EUTHERIAN, EUTHERIAN MAMMAL
  FOSSORIAL MAMMAL, MONOTREME, EGG-LAYING MAMMAL, MARSUPIAL
  POUCHED MAMMAL, LIVESTOCK, STOCK, FARM ANIMAL, BULL, COW
  YEARLING, BUCK, DOE, INSECTIVORE, AQUATIC MAMMAL, CARNIVORE
  FISSIPEDIA, AARDVARK, ANT BEAR, ANTEATER, ORYCTEROPUS AFER
  BAT, CHIROPTERAN, LAGOMORPH, GNAWING MAMMAL, RODENT, GNAWER
  GNAWING ANIMAL, UNGULATA, UNGULATE, HOOFED MAMMAL
  UNGUICULATA, UNGUICULATE, UNGUICULATE MAMMAL, HYRAX, CONEY
  CONY, DASSIE, DAS, PACHYDERM, EDENTATE, PANGOLIN
  SCALY ANTEATER, PRIMATE, TREE SHREW, FLYING LEMUR
  FLYING CAT, COLUGO, PROBOSCIDEAN, PROBOSCIDIAN
  PLANTIGRADE MAMMAL, DIGITIGRADE MAMMAL, NAKED MOLE RAT
  DAMARALAND MOLE RAT, ECHIDNA, SPINY ANTEATER, PLATYPUS
  DUCKBILL, DUCKBILLED PLATYPUS, DUCK-BILLED PLATYPUS
  ORNITHORHYNCHUS ANATINUS, OPOSSUM, POSSUM, OPOSSUM RAT
  BANDICOOT, KANGAROO, PHALANGER, WOMBAT, DASYURID MARSUPIAL
  DASYURID, POUCHED MOLE, MARSUPIAL MOLE, NOTORYCTUS TYPHLOPS
  STAG, MOLE, SHREW, SHREWMOUSE, HEDGEHOG, ERINACEUS EUROPAEUS
  ERINACEUS EUROPEAEUS, TENREC, TENDRAC, OTTER SHREW
  POTAMOGALE, POTAMOGALE VELOX, CETACEAN, CETACEAN MAMMAL
  BLOWER, SEA COW, SIRENIAN MAMMAL, SIRENIAN, PINNIPED MAMMAL
  PINNIPED, PINNATIPED, FISSIPED MAMMAL, FISSIPED, CANINE
  CANID, FELINE, FELID, BEAR, VIVERRINE, VIVERRINE MAMMAL
  MUSTELINE MAMMAL, MUSTELID, MUSTELINE, PROCYONID, FRUIT BAT
  MEGABAT, CARNIVOROUS BAT, MICROBAT, DUPLICIDENTATA, LEPORID
  LEPORID MAMMAL, PIKA, MOUSE HARE, ROCK RABBIT, MOUSE, RAT
  MURINE, WATER RAT, NEW WORLD MOUSE, MUSKRAT, MUSQUASH
  ONDATRA ZIBETHICA, ROUND-TAILED MUSKRAT, FLORIDA WATER RAT
  NEOFIBER ALLENI, COTTON RAT, SIGMODON HISPIDUS, WOOD RAT
  WOOD-RAT, HAMSTER, GERBIL, GERBILLE, LEMMING, PORCUPINE
  JUMPING MOUSE, JERBOA, DORMOUSE, SQUIRREL, PRAIRIE DOG
  PRAIRIE MARMOT, MARMOT, BEAVER, MOUNTAIN BEAVER, SEWELLEL
  APLODONTIA RUFA, CAVY, MARA, DOLICHOTIS PATAGONUM, CAPYBARA
  CAPIBARA, HYDROCHOERUS HYDROCHAERIS, AGOUTI
  DASYPROCTA AGUTI, PACA, CUNICULUS PACA, MOUNTAIN PACA, COYPU
  NUTRIA, MYOCASTOR COYPUS, CHINCHILLA, CHINCHILLA LANIGER
  MOUNTAIN CHINCHILLA, MOUNTAIN VISCACHA, VISCACHA
  CHINCHILLON, LAGOSTOMUS MAXIMUS, ABROCOME, CHINCHILLA RAT
  RAT CHINCHILLA, MOLE RAT, SAND RAT, DINOCERATE
  ODD-TOED UNGULATE, PERISSODACTYL, PERISSODACTYL MAMMAL
  EVEN-TOED UNGULATE, ARTIODACTYL, ARTIODACTYL MAMMAL
  ROCK HYRAX, PROCAVIA CAPENSIS, ARMADILLO, SLOTH, TREE SLOTH
  MEGATHERIAN, MEGATHERIID, MEGATHERIAN MAMMAL, MYLODONTID
  MYLODON, NEW WORLD ANTEATER, SIMIAN, APE, ANTHROPOID
  HOMINOID, HOMINID, MONKEY, PROSIMIAN, LEMUR, TARSIER
  PENTAIL, PEN-TAIL, PEN-TAILED TREE SHREW
  CYNOCEPHALUS VARIEGATUS, ELEPHANT, MASTODON, MASTODONT
Generating the Information Extraction JAPE input file
  Macro: acc_acc_ref
    ({Lookup.minorType == acc_acc_ref})


  Rule: acc_acc_ref
  (
  (acc_acc_ref)
  )
  :acc_acc_ref -->
    :acc_acc_ref.acc_acc_ref = {kind ="acc_acc_ref", rule =
     "acc_acc_ref"}


  Macro: acc_road_type
    ({Lookup.minorType == acc_road_type})


  Rule: acc_road_type
```

```
  (
  (acc_road_type)
  )
  :acc_road_type -->
    :acc_road_type.acc_road_type = {kind ="acc_road_type", rule =
      "acc_road_type"}


  Macro: towns_town
    ({Lookup.minorType == towns_town})


  Rule: towns_town
  (
  (towns_town)
  )
  :towns_town -->
    :towns_town.towns_town = {kind ="towns_town", rule =
     "towns_town"}


  Macro: roads_road
    ({Lookup.minorType == roads_road})


  Rule: roads_road
  (
  (roads_road)
  )
  :roads_road -->
    :roads_road.roads_road = {kind ="roads_road", rule =
     "roads_road"}


  Macro: acc_road
    ({Lookup.minorType == acc_road})


  Rule: acc_road
  (
  (acc_road)
  )
  :acc_road -->
    :acc_road.acc_road = {kind ="acc_road", rule = "acc_road"}
  Macro: animal
    ({Lookup.minorType == animal})


  Rule: animal
  (
  (animal)
  )
  :animal -->
   :animal.animal = {kind ="animal", rule = "animal"}


  Macro: obstruction
    ({Lookup.minorType == obstruction})


  Rule: obstruction
  (
  (obstruction)
  )
  :obstruction -->
   :obstruction.obstruction = {kind ="obstruction", rule =
     "obstruction"}


Created IE input file ie.jape

=================================================================
  INFORMATION EXTRACTION STEP (11)
=================================================================

No JAPE URI specified - default will be used
Configuring Database Gazetteer using file:/C:/estest/dbGaz.xml
Named Entity source to be loaded is accDBautzzzi:obstruction
Named Entity source to be loaded is accDBautzzzi:acc_road
Loaded 227 values for word-form NE object <<obstruction>>
  About to run IQL query. Schema:accDBautzzzi, query:distinct
   <<acc_road>>
Loaded 5 values for extent-based NE object <<acc_road>>
Reading accDBautzzzi:obstruction
Reading accDBautzzzi:acc_road
About to run IQL query. Schema:accDBautzzzi,
query:<<attribute,acc,acc_acc_desc>>
```

```
Added doc FOX RUNS INTO ABBEY STREET CAUSING V1 TO SWERVE
VIOLENTLY AND LEAVE ROAD OFFSIDE 50M AWAY
template instance is A001234
Added doc A50 WELFORD ROAD LEICESTER,BRIDGE 200 YDS S ROMAN WAY.
 V1 TRAV MOTORWAY M6 FAILS TO STOP AT XRDS AND HITS V2 TRAV
UPPERTON RD V2 THEN HITS V3 PKD ON OS OF UPPERTON RD
template instance is B231562
Added doc ESCAPED KANGAROO JUMPS IN FRONT OF V1
template instance is C051633
-----------------------------------------------------------------
-

Document to be processed by IE : 'FOX RUNS INTO ABBEY STREET
CAUSING V1 TO SWERVE VIOLENTLY AND LEAVE ROAD OFFSIDE 50M AWAY'
Running processing resources over corpus...

      Annotations:
   ......Annotation Set jape
Anotation type : obstruction rule/obstruction kind/obstruction
adding FOX/<<obstruction>>
Anotation type : acc_road rule/acc_road kind/acc_road
adding ABBEY STREET/<<acc_road>>
Annotation Details:
  Schema element = '<<obstruction>>', value = 'FOX' and the ID
   Will be generated.
  Schema element = '<<acc_road>>', value = 'ABBEY STREET' and
the ID will be generated.
adding node to HDM Store with generated id <<fox>>
[estestInstance3]
adding edge <<isA,fox,animal>>[A001234,estestInstance3]
adding edge <<isA,animal,obstruction>>[A001234,estestInstance3]
adding template attribute edge
<<attribute,acc,obstruction>>[A001234,estestInstance3]
adding node to HDM Store with generated id <<acc_road>>
[estestInstance4]
adding template attribute edge
<<attribute,acc,acc_road>>[A001234,estestInstance4]

-----------------------------------------------------------------

Document to be processed by IE : 'A50 WELFORD ROAD
LEICESTER,BRIDGE 200 YDS S ROMAN WAY. V1 TRAV MOTORWAY M6 FAILS
TO STOP AT XRDS AND HITS V2 TRAV UPPERTON RD V2 THEN HITS V3 PKD
ON OS OF UPPERTON RD'
Running processing resources over corpus...
```

```
      Annotations:
   ......Annotation Set jape
Annotation Details:

----------------------------------------------------------------

Document to be processed by IE : 'ESCAPED KANGAROO JUMPS IN
FRONT OF V1'
Running processing resources over corpus...

      Annotations:
   ......Annotation Set jape
Anotation type : obstruction rule/obstruction kind/obstruction
adding KANGAROO/<<obstruction>>
Annotation Details:
  Schema element = '<<obstruction>>', value = 'KANGAROO' and the
   ID will be generated.
adding node to HDM Store with generated id <<animal>>
[estestInstance5]
```

```
adding edge <<isA,animal,obstruction>>[C051633,estestInstance5]
adding template attribute edge
<<attribute,acc,obstruction>>[C051633,estestInstance5]

================================================================

   QUERY STEP (12)
================================================================

About to run IQL query. Schema:accDBautzzzi,
query:<<attribute,ac
 c,obstruction>>
  The query was <<attribute,acc,obstruction>>
  Results: [{A001234 ,estestInstance3 },{C051633
,estestInstance5}]
Closing debug log file.
```

Appendix C

ESTEST Output from the Crime Example

This appendix includes the input files used to configure ESTEST, the automatically produced input used to configure GATE, and the resulting output from ESTEST.

C.1 Input Provided to Configure ESTEST

Three input files are provided to configure ESTEST. While these files are currently created by hand, they have been designed with a GUI in mind, as described in Chapter 8. The first input file is a script which i) integrates the datasources, ii) loads the patterns used to generate JAPE rules, iii) creates the GATE configuration files, iv) generates those files, and v) runs the IE process.

```
<?xml version="1.0"?>

<!DOCTYPE script SYSTEM "script.dtd">
<!-- script for ESTEST to perform -->
<script>
    <step>
            <name>Integrate</name>
            <uri>C:\estest\bin\config\dsdCrime.xml</uri>
    </step>
     <step>
            <name>Parameters</name>

    <uri>C:\estest\bin\config\parmsCrimePatterns.xml</uri>
     </step>
     <step>
            <name>Config</name>
     </step>
    <step>
          <name>ConfigOutput</name>
    </step>
    <step>
          <name>IE</name>
    </step>
</script>
```

The second input file is a data source definition file, which in our example defines the two relational datasources `OpIntel` and `CarsDB`, and the RDF/S datasource `CrimeOnt`:

```
<?xml version="1.0"?>

<!DOCTYPE dsd SYSTEM "dsd.dtd">
<!-- datasource defintions for the data source definitions -->
<dsd>
   <relational_ds>
          <name>OpIntel</name>
          <driver>org.postgresql.Driver</driver>
          <url>jdbc:postgresql://193.61.29.5/OpIntel</url>
          <username>dean</username>
          <password>dean</password>
   </relational_ds>
   <relational_ds>
          <name>Cars</name>
          <driver>org.postgresql.Driver</driver>
          <url>jdbc:postgresql://193.61.29.5/CarsDB</url>
          <username>dean</username>
          <password>dean</password>
   </relational_ds>
   <ontology_ds>
          <name>CrimeOnt</name>
          <rdf_url>

file:C:/estest/bin/config/CrimeOntRdfEmpty.xml</rdf_url>
          <rdfs_url>
         file:C:/estest/bin/config/CrimeOntRdfs.xml</rdfs_url>
   </ontology_ds>
</dsd>
```

The third input file contains the patterns which are stored in the EMR and are used to generate JAPE rules for processing the unstructured textual data:

```
<?xml version="1.0"?>

<!DOCTYPE parameters SYSTEM "parms.dtd">
<!-- configuration file to be run before IE in crime example -->
<parameters>
    <macro>
        <name>model</name>
    </macro>
    <macro>
        <name>colour</name>
    </macro>
    <macro>
        <name>manufacturer</name>
    </macro>
    <macro>
        <name>car_reg</name>
    </macro>
    <value_def>
        <name>REGISTRATION_MARK</name>
        <schema_object_name>car_reg</schema_object_name>
        <value_def_part>
            <type>String</type>
            <length>2</length>
        </value_def_part>
        <value_def_part>
            <type>Integer</type>
            <length>2</length>
        </value_def_part>
        <value_def_part>
            <type>Space</type>
            <length>1</length>
        </value_def_part>
         <value_def_part>
            <type>String</type>
            <length>3</length>
        </value_def_part>
   </value_def>
   <pattern>
                  <name>known_car</name>
                  <schema_object_name>car</schema_object_name>
                  <value>(COLOUR)?--(MANUFACTURER)?--(MODEL)—
                      (CAR_REG)</value>
                  <id_name>car_reg</id_name>
   </pattern>
   <pattern>
                  <name>new_car</name>
                  <schema_object_name>car</schema_object_name>
                  <value>(COLOUR)?--(MANUFACTURER)?--(MODEL)?--
CAR?—
                      REGISTRATION?--(REGISTRATION_MARK)</value>
                  <id_name>car_reg</id_name>
   </pattern>
   <pattern>
                  <name>car_no_reg</name>
                  <schema_object_name>car</schema_object_name>
                  <value>(COLOUR)?--(MANUFACTURER)?--
(MODEL)</value>
               <id_name>car_reg</id_name>
   </pattern>
</parameters>
```

C.2 Automatically Created GATE Configuration Files

The rules used by ESTEST in schema matching, as described in Section 7.2, are automatically created from the global schema:

```
Phase:     EstestJape
Options: control = brill
```

```

Macro: op_intel
({Lookup.minorType == op_intel})

Rule: op_intel
(
(op_intel)
)
:op_intel -->
   :op_intel.op_intel = {kind ="op_intel", rule = "op_intel"}

Macro: op_intel_intel
({Lookup.minorType == op_intel_intel})

Rule: op_intel_intel
(
(op_intel_intel)
)
:op_intel_intel -->
   :op_intel_intel.op_intel_intel = {kind ="op_intel_intel",
rule = "op_intel_intel"}

Macro: op_intel_pc
({Lookup.minorType == op_intel_pc})

Rule: op_intel_pc
(
(op_intel_pc)
)
:op_intel_pc -->
   :op_intel_pc.op_intel_pc = {kind ="op_intel_pc", rule =
"op_intel_pc"}

Macro: op_intel_report_id
({Lookup.minorType == op_intel_report_id})

Rule: op_intel_report_id
(
(op_intel_report_id)
)
:op_intel_report_id -->
   :op_intel_report_id.op_intel_report_id = {kind
="op_intel_report_id", rule = "op_intel_report_id"}

Macro: car
```

```
({Lookup.minorType == car})

Rule: car
(
(car)
)
:car -->
   :car.car = {kind ="car", rule = "car"}

Macro: car_reg
({Lookup.minorType == car_reg})

Rule: car_reg
(
(car_reg)
)
:car_reg -->
   :car_reg.car_reg = {kind ="car_reg", rule = "car_reg"}

Macro: colour
({Lookup.minorType == colour})

Rule: colour
(
(colour)
)
:colour -->
   :colour.colour = {kind ="colour", rule = "colour"}

Macro: colour_colour
({Lookup.minorType == colour_colour})

Rule: colour_colour
(
(colour_colour)
)
:colour_colour -->
   :colour_colour.colour_colour = {kind ="colour_colour", rule =
"colour_colour"}

Macro: manufacturer
({Lookup.minorType == manufacturer})

Rule: manufacturer
(
```

```
(manufacturer)
)
:manufacturer -->
   :manufacturer.manufacturer = {kind ="manufacturer", rule =
"manufacturer"}

Macro: manufacturer_manufacturer
({Lookup.minorType == manufacturer_manufacturer})

Rule: manufacturer_manufacturer
(
(manufacturer_manufacturer)
)
:manufacturer_manufacturer -->
   :manufacturer_manufacturer.manufacturer_manufacturer = {kind
="manufacturer_manufacturer", rule =
"manufacturer_manufacturer"}

Macro: model
({Lookup.minorType == model})

Rule: model
(
(model)
)
:model -->
   :model.model = {kind ="model", rule = "model"}

Macro: model_model
({Lookup.minorType == model_model})

Rule: model_model
(
(model_model)
)
:model_model -->
   :model_model.model_model = {kind ="model_model", rule =
"model_model"}

Macro: Resource
({Lookup.minorType == Resource})

Rule: Resource
(
(Resource)
```

```
)
:Resource -->
   :Resource.Resource = {kind ="Resource", rule = "Resource"}

Macro: opIntel
({Lookup.minorType == opIntel})

Rule: opIntel
(
(opIntel)
)
:opIntel -->
   :opIntel.opIntel = {kind ="opIntel", rule = "opIntel"}

Macro: pub
({Lookup.minorType == pub})

Rule: pub
(
(pub)
)
:pub -->
   :pub.pub = {kind ="pub", rule = "pub"}

Macro: vehicle
({Lookup.minorType == vehicle})

Rule: vehicle
(
(vehicle)
)
:vehicle -->
   :vehicle.vehicle = {kind ="vehicle", rule = "vehicle"}
```

The text matching patterns stored in the EMR are used by ESTEST to automatically produce the configuration for the IE process; firstly, the configuration file for the `SchemaGazetteer` component:

```
<?xml version="1.0"?>
<!DOCTYPE SchemaGazetteer  SYSTEM "dbGaz.dtd">
<!-- named entity source defs for the SchemaGazetteer  -->
<SchemaGazetteer >
   <ne_source>
          <schema>OpIntelautzx</schema>
          <object>model</object>
          <type>extent</type>
   </ne_source>
   <ne_source>
          <schema>OpIntelautzx</schema>
          <object>colour</object>
          <type>extent</type>
   </ne_source>
   <ne_source>
          <schema>OpIntelautzx</schema>
          <object>manufacturer</object>
          <type>extent</type>
   </ne_source>
   <ne_source>
          <schema>OpIntelautzx</schema>
          <object>car_reg</object>
          <type>extent</type>
   </ne_source>
   <ne_source>
          <schema>OpIntelautzx</schema>
          <object>car</object>
          <type>extent</type>
   </ne_source>
   <ne_source>
          <schema>OpIntelautzx</schema>
          <object>model</object>
          <type>extent</type>
   </ne_source>
   <ne_source>
          <schema>OpIntelautzx</schema>
          <object>colour</object>
          <type>extent</type>
   </ne_source>
   <ne_source>
          <schema>OpIntelautzx</schema>
          <object>manufacturer</object>
          <type>extent</type>
   </ne_source>
   <ne_source>
```

```
          <schema>OpIntelautzx</schema>
          <object>car_reg</object>
          <type>extent</type>
   </ne_source>
</SchemaGazetteer >
```

Secondly, the automatically produced JAPE rules for processing the text:

```
Phase:     EstestJape
Options: control = all


Macro: SP
 (
  ({SpaceToken.kind == space, SpaceToken.length == "1"}
  ({SpaceToken.kind == control, SpaceToken.length == "1"})?) |
  ({SpaceToken.kind == control, SpaceToken.length == "1"}
  ({SpaceToken.kind == space, SpaceToken.length == "1"})?)
  )

Macro: SPACE
(
 ({Token.string == ","})?
 (SP)+
)

Macro: MODEL
({Lookup.minorType == model})



Rule: model
(
(MODEL)
)
:model -->
   :model.model = {kind ="model", rule = "model",
       idAnnotationType="model"}

Macro: COLOUR
({Lookup.minorType == colour})



Rule: colour
(
(COLOUR)
)
:colour -->
   :colour.colour = {kind ="colour", rule = "colour",
       idAnnotationType="colour"}

Macro: MANUFACTURER
({Lookup.minorType == manufacturer})



Rule: manufacturer
(
(MANUFACTURER)
)
:manufacturer -->
   :manufacturer.manufacturer = {kind ="manufacturer", rule =
      "manufacturer", idAnnotationType="manufacturer"}

Macro: CAR_REG
({Lookup.minorType == car_reg})



Rule: car_reg
(
(CAR_REG)
)
:car_reg -->
   :car_reg.car_reg = {kind ="car_reg", rule = "car_reg",
      idAnnotationType="car_reg"}

Macro: lamppost
(
  {Token.string == "lamppost"}  |
  {Token.string == "streetlight"}
)

Macro: APPROX
```

```
(
  {Token.string == "APPROX"}  |
  {Token.string == "APPROXIMATELY"}  |
  {Token.string == "ABOUT"}
)

Macro: REGISTRATION_MARK
(
    ({Token.kind == word, Token.length == "2"})
     ({Token.kind == number, Token.length == "2"})
     ((SPACE))
     ({Token.kind == word, Token.length == "3"})
 )

Rule: REGISTRATION_MARK
(
  (REGISTRATION_MARK)
)
 :REGISTRATION_MARK -->
   :REGISTRATION_MARK.car_reg = {kind = "car_reg", rule =
     "REGISTRATION_MARK", estestStore="yes",
     idAnnotationType="car_reg"}


Rule: car
(
(COLOUR)? (SPACE)? (MANUFACTURER)? (SPACE)? (MODEL) (SPACE)?
(CAR_REG)
):car -->
   :car.car = {kind = "car", rule = "known_car0",
       idAnnotationType="car_reg"}

Rule: car
(
(COLOUR)? (SPACE)? (MANUFACTURER)? (SPACE)? (MODEL)? (SPACE)?
({Token.string == "CAR"})? (SPACE)? ({Token.string ==
"REGISTRATION"})? (SPACE)? (REGISTRATION_MARK)
):car -->
   :car.car = {kind = "car", rule = "new_car0",
      idAnnotationType="car_reg"}

Rule: car
(
(COLOUR)? (SPACE)? (MANUFACTURER)? (SPACE)? (MODEL)
):car -->
   :car.car = {kind = "car", rule = "car_no_reg0",
     idAnnotationType="car_reg"}
```

C.3 Output from ESTEST

```
Loading class uk.ac.ic.doc.automed.wrappers.TransactSQLWrapper
Loading class uk.ac.ic.doc.automed.wrappers.PostgresWrapper
Loading class uk.ac.ic.doc.automed.wrappers.OracleWrapper
Loading class uk.ac.ic.doc.automed.wrappers.YattaWrapper
Loading class uk.ac.bbk.dcs.automed.xml.wrappers.DOMWrapper
Loading class uk.ac.bbk.dcs.automed.xml.wrappers.SAXWrapper
Loading class uk.ac.bbk.dcs.automed.hdmstore.HdmWrapper
Loading class uk.ac.bbk.estest.RdfWrapper
Loading class uk.ac.bbk.estest.EstestOntologyWrapper

----------------------------------------------------------------

ESTEST running from a script.
Using:C:\estest\bin\config\crimeScript.xml
Building Estest modelling language
================================================================
   DETAILS OF LOADED ESTEST SCRIPT
================================================================

ScriptStep 1: type=Integrate uri
=C:\estest\bin\config\dsdCrime.xml

ScriptStep 2: type=Parameters uri=C:\estest\bin\config\parmsCrim
 ePatterns.xml

ScriptStep 3: type=Config

ScriptStep 4: type=ConfigOutput

ScriptStep 5: type=IE


================================================================
   INTEGRATE STEP (1)
```

```
================================================================

Loading datasourced from definition
at:C:\estest\bin\config\dsdCrime.xml
Building RDF modelling language
RDF Wrapper Factory creating RDF Model Oriented Schema CrimeOnt
schema
Details of schema: CrimeOnt
  RDF subject           <<subject>>
  RDF predicate         <<predicate>>
  RDF object            <<object>>
  RDF triple            <<triple,subject,predicate,object>>
  RDF uri               <<uri>>
  RDF blank             <<blank>>
  RDF literal           <<literal>>


Data Sources To Be Integrated:
  DS 1 is OpIntel
  DS 2 is Cars
  DS 3 is CrimeOnt
Creating the AutoMed Schemas.
  RelationalDataSource is building a wrapper for schema
OpIntelauto
  Created AutoMed schema for OpIntel
  Details of schema: OpIntelauto
    sql_390 table        <<op_intel>>
    sql_390 column       <<op_intel,report_id>>
    sql_390 column       <<op_intel,pc>>
    sql_390 column       <<op_intel,intel>>
    sql_390 primary_key
                         <<pky_op_intel,op_intel,
                         <<op_intel,reportid>>>
  RelationalDataSource is building a wrapper for schema Carsauto
  Created AutoMed schema for Cars
  Details of schema: Carsauto
    sql_390 table        <<colour>>
    sql_390 column       <<colour,colour>>
    sql_390 primary_key  <<pky_colour,colour,<<colour,colour>>>>
    sql_390 table        <<car>>
    sql_390 column       <<car,reg>>
    sql_390 column       <<car,manufacturer>>
    sql_390 column       <<car,model>>
    sql_390 column       <<car,colour>>
    sql_390 primary_key  <<pky_car,car,<<car,reg>>>>
```

```
    sql_390 table        <<model>>
    sql_390 column       <<model,model>>
    sql_390 primary_key  <<pky_model,model,<<model,model>>>>
    sql_390 table        <<manufacturer>>
    sql_390 column       <<manufacturer,manufacturer>>
    sql_390 primary_key  <<pky_manufacturer,manufacturer,
                         <<manufacturer,manufacturer>>>>
    sql_390 foreign_key  <<fky_car_colour,car,
                         <<car,colour>>,
                         colour,<<colour,colour>>>>
    sql_390 foreign_key  <<fky_car_manufacturer,car,
                         <<car,manufacturer>>,manufacturer,
                         <<manufacturer,manufacturer>>>>
    sql_390 foreign_key  <<fky_car_model,car,<<car,model>>
                         ,model,<<model,model>>>>

Creating the ESTEST Model Schemas.
    Finding foreign keys (for isA relationship).
    Finding tables and columns (for concepts).
    Now delete the relational constructs.....
    The Estest oriented schema for this relational data source
is OpIntelautza
    Now find word forms for each schema element by processing
using SchemaNameTokeniser component.
    Creating schema name tokeniser to process names of schema
objects.
Using C:\Program Files\GATE 3.0 as GATE home
Using C:\Program Files\GATE 3.0\plugins as installed plug-ins
directory.
Using C:\Program Files\GATE 3.0\gate.xml as site configuration
file.
Using C:\Documents and Settings\dean\gate.xml as user
configuration file
CREOLE plugin loaded: file:/C:/Program Files/GATE-
3.1b1/plugins/ANNIE/
CREOLE plugin loaded: file:/C:/estest/
    Storing Data Source info and metadata in EMR.....
  Created ESTEST schema for OpIntel
  Details of schema: OpIntelautza
    Estest concept       <<op_intel>>
    Estest concept       <<op_intel_report_id>>
    Estest attribute     <<attribute,op_intel,
                         op_intel_report_id>>
    Estest concept       <<op_intel_pc>>
    Estest attribute     <<attribute,op_intel,op_intel_pc>>
    Estest concept       <<op_intel_intel>>
    Estest attribute     <<attribute,op_intel,op_intel_intel>>


    Finding foreign keys (for isA relationship).
    Finding tables and columns (for concepts).
    Now delete the relational constructs.....
    The Estest oriented schema for this relational data source
is Carsautzv
    Now find word forms for each schema element by processing
using SchemaNameTokeniser component.
    Storing Data Source info and metadata in EMR.....
  Created ESTEST schema for Cars
  Details of schema: Carsautzv
    Estest concept       <<colour>>
    Estest concept       <<colour_colour>>
    Estest attribute     <<attribute,colour,colour_colour>>
    Estest concept       <<car>>
    Estest concept       <<car_reg>>
    Estest attribute     <<attribute,car,car_reg>>
    Estest concept       <<model>>
    Estest concept       <<model_model>>
    Estest attribute     <<attribute,model,model_model>>
    Estest concept       <<manufacturer>>
    Estest concept       <<manufacturer_manufacturer>>
    Estest attribute     <<attribute,manufacturer,
                         manufacturer_manfacturer>>
    Estest attribute     <<attribute,car,colour>>
    Estest attribute     <<attribute,car,manufacturer>>
    Estest attribute     <<attribute,car,model>>


  OntologyDataSource is about to create ESTEST Schema.
  found class: http://www.dcs.bbk.ac.uk/~dean/kb/acc1#pub
  found class: http://www.dcs.bbk.ac.uk/~dean/kb/acc1#opIntel
  found class: http://www.dcs.bbk.ac.uk/~dean/kb/acc1#vehicle
  About to check for word forms.
  Created ESTEST schema for CrimeOnt
  Details of schema: CrimeOntEstest
    Estest concept       <<pub>>
    Estest concept       <<opIntel>>
    Estest concept       <<vehicle>>
    Estest concept       <<Resource>>
    Estest isA           <<isA,opIntel,Resource>>
    Estest isA           <<isA,vehicle,Resource>>
```

```
    Estest isA           <<isA,pub,Resource>>
    Estest attribute     <<attribute,opIntel,pub>>
    Estest attribute     <<attribute,opIntel,vehicle>>


Going to process text metadata for additional schema element
matching evidence.

================================================================
   CREATING GATE PIPELINE FOR SCHEMA MATCHING.
================================================================

Creating Default Tokeniser Gate processing resource.
Creating Sentence-Splitter Gate processing resource.
Creating Database-Gazetteer Gate processing resource with all
schema objects as NE sources based on the word forms extracted
from schema names.
  No data source URL provided so loading word form named
entities for all schema elements.
  Loading definition of Named Entity

================================================================
   PROCESSING TEXTUAL METADATA FOR SCHEMA MATCHING
================================================================

Creating Jape rules for processing schema metadata: smie.jape
path: C:\estest\smie.jape
Creating Jape Transducer Gate Processing Resource.
JAPE URL: file:/C:/estest/smie.jape
Assembling Components Into Pipeline.
Gate is now initialised and the ESTEST application is built.

----------------------------------------------------------------

Document to be processed by IE : 'OPERATIONAL INTELLIGENCE
GATHERED BY POLICE OFFICERS ON THEIR PATROLS'
Running processing resources over document...

----------------------------------------------------------------

Document to be processed by IE : 'GENERATED ID FOR A REPORT'
Running processing resources over document...

----------------------------------------------------------------
```

```
Document to be processed by IE : 'POLICE CONSTABLE WHO MADE THE
REPORT'
Running processing resources over document...

-----------------------------------------------------------------

Document to be processed by IE : 'INTELLIGENCE GATHERED'
Running processing resources over document...

-----------------------------------------------------------------

Document to be processed by IE : 'VEHICLE SEEN DURING
OPERATIONAL INTELLIGENCE GATHERING'
Running processing resources over document...
Match between the textual metadata of schema element 84/62, and
the schema element 85/104

-----------------------------------------------------------------

Document to be processed by IE : 'UK REG MARK IE TWO CHAR AREA
CODE, AGE, AND THREE RANDOM LETTERS'
Running processing resources over document...

-----------------------------------------------------------------



Going to find matches between Schema elements.
The matches are:
  Match with 0.5% confidence on word form meta-match
    Schema 1: Carsautzv, Concept 1: car
    Schema 2: CrimeOntEstest, Concept 2: vehicle


Going to rename matching elements so they have the same name.
In Integrator getPositionOfSchema looking for sid 84
  checking 51, OpIntelautza
  checking 84, Carsautzv
  ........ and its a match
In Integrator getPositionOfSchema looking for sid 85
  checking 51, OpIntelautza
  checking 84, Carsautzv
  checking 85, CrimeOntEstest
  ........ and its a match
 Renamed :CrimeOntEstest, vehicle  to car
```

```
Extend to match schemas.
 OpIntelautza extending to match Carsautzv
  new extended schema is :OpIntelautzp
 OpIntelautzp extending to match CrimeOntEstest1a
  new extended schema is :OpIntelautzx
 Carsautzv extending to match OpIntelautza
  new extended schema is :Carsautzzc
 Carsautzzc extending to match CrimeOntEstest1a
  new extended schema is :Carsautzzk
 CrimeOntEstest1a extending to match OpIntelautza
  new extended schema is :CrimeOntEstest1h
 CrimeOntEstest1h extending to match Carsautzv
  new extended schema is :CrimeOntEstest1v
Assert Identity Transformations between the extended schemas.
 Asserting ID transformation between OpIntelautzx & Carsautzzk
 Asserting ID transformation between OpIntelautzx &
CrimeOntEstest1v
About to create HDM store copy of global schema.
Building the AutoMed HDM model
Creating HDM Store estest_store
Creating transormation pathway from HDM model to ESTEST model
global schema
Materialising isA relationships.
  Contents of the IsaFunctionList are:
    <<opIntel>>       <<opIntel>>
    <<Resource>>      <<Resource>> ++ <<opIntel>> ++ <<pub>> ++
                      <<car>>
    <<pub>>           <<pub>>
    <<car>>           <<car>>
  Integrator loadDef attempting ident with HDM Store


  Global Schema is complete, schema name is: OpIntelautzx
  Details of schema: OpIntelautzx
    Estest concept       <<op_intel>>
    Estest concept       <<op_intel_report_id>>
    Estest attribute     <<attribute,op_intel,
                         op_intel_report_id>>
    Estest concept       <<op_intel_pc>>
    Estest attribute     <<attribute,op_intel,op_intel_pc>>
    Estest concept       <<op_intel_intel>>
    Estest attribute     <<attribute,op_intel,op_intel_intel>>
    Estest concept       <<colour>>
    Estest concept       <<colour_colour>>
    Estest attribute     <<attribute,colour,colour_colour>>
    Estest concept       <<car>>
    Estest concept       <<car_reg>>
    Estest attribute     <<attribute,car,car_reg>>
    Estest concept       <<model>>
    Estest concept       <<model_model>>
    Estest attribute     <<attribute,model,model_model>>
    Estest concept       <<manufacturer>>
    Estest concept       <<manufacturer_manufacturer>>
    Estest attribute     <<attribute,manufacturer,
                         manufacturer_manfacturer>>
    Estest attribute     <<attribute,car,colour>>
    Estest attribute     <<attribute,car,manufacturer>>
    Estest attribute     <<attribute,car,model>>
    Estest concept       <<pub>>
    Estest concept       <<opIntel>>
    Estest concept       <<Resource>>
    Estest isA           <<isA,opIntel,Resource>>
    Estest isA           <<isA,pub,Resource>>
    Estest attribute     <<attribute,opIntel,pub>>
    Estest isA           <<isA,car,Resource>>
    Estest attribute     <<attribute,opIntel,car>>



================================================================
   PARAMETERS STEP (2)
================================================================

Parameters to be loaded:
  Macro Parameter: model, and this is a stub to be generated as
a NE source.

  Macro Parameter: colour, and this is a stub to be generated as
a NE source.
  Macro Parameter: manufacturer, and this is a stub to be
generated as a NE source.
  Macro Parameter: car_reg, and this is a stub to be generated
as a NE source.
  ValueDefParameter, Name: REGISTRATION_MARK
  Pattern Parameter: car Pattern:  (COLOUR)?--(MANUFACTURER)?--
(MODEL)--(CAR_REG), id:car_reg
  Pattern Parameter: car Pattern:  (COLOUR)?--(MANUFACTURER)?--
(MODEL)?--CAR?--REGISTRATION?--(REGISTRATION_MARK), id:car_reg
  Pattern Parameter: car Pattern:  (COLOUR)?--(MANUFACTURER)?--
(MODEL), id:car_reg
```

```

================================================================
   CONFIGIE STEP (3)
================================================================

Generating Suggestions for Named Entity.
Suggested possible NE List is:
  Possible Extent NE object - Data Source: 1, Schema
element<<op_intel_pc>>
  Possible Extent NE object - Data Source: 2, Schema
element<<colour_colour>>
  Possible Extent NE object - Data Source: 2, Schema
element<<car_reg>>
  Possible Extent NE object - Data Source: 2, Schema
element<<model_model>>
  Possible Extent NE object - Data Source: 2, Schema
element<<manufacturer_manufacturer>>
Identifying Text Sources.
Finding Templates.
Removing templates with only one attribute...

Templates Are:
  Template: <<opIntel>>
    Attribute: <<pub>>
    Attribute: <<car>>
  Template: <<op_intel>>
    Attribute: <<op_intel_report_id>>
    Attribute: <<op_intel_pc>>
    Attribute: <<op_intel_intel>>
  Template: <<car>>
    Attribute: <<car_reg>>
    Attribute: <<colour>>
    Attribute: <<manufacturer>>
    Attribute: <<model>>
Generating aliases for Orthomatcher.
Generating the alias file for named co-ref detection
   file :alias.lst path: orthomatcher\alias.lst
about to delete alias.lst path: orthomatcher\alias.lst
about to create alias.lst path: C:\estest\orthomatcher\alias.lst
Created IE input file alias.lst
ESTEST is set NOT to wait for user confirmation of results.

================================================================
   CONFIG OUTPUT GENERATION STEP (4)
================================================================
Expanding the selected Named Entity schema elements.
Jape rules are being generated from EMR patterns.
Jape Generator - getting the macros definitions.
Jape Generator - getting the value pattern definitions.
Jape Generator - getting the regular patterns definitions.
Generating the Information Extraction JAPE input file from EMR
patterns.
Created IE input file ie.jape
Generating the DBGaz.xml configuration file for the DB Gazeteer
Component
   file :dbGaz.xml path: dbGaz.xml
about to delete dbGaz.xml path: dbGaz.xml
  Created DB Gaz Config file dbGaz.xml

================================================================
   INFORMATION EXTRACTION STEP (5)
================================================================

Initialising Gate using Gate Home :C:\Program Files\GATE-3.1
Using C:\Program Files\GATE 3.0 as GATE home
Using C:\Program Files\GATE 3.0\plugins as installed plug-ins
directory.
Using C:\Program Files\GATE 3.0\gate.xml as site configuration
file.
Registering Creole directories:
Creating Default Tokeniser Gate Processing Resource.
Creating POS Tagger Gate Processing Resource.
Creating Sentence Splitter Gate Processing Resource.
Creating ANNIE Transducer Gate Processing Resource.
Creating Default Gazetteer Gate Processing Resource.
Creating Database Gazetteer Gate Processing Resource.
Creating Jape Transducer Gate Processing Resource.
JAPE URL: file:/C:/estest/ie.jape
Creating Orthomatcher Gate Processing Resource, configuring
using ESTEST generated aliases.
Creating Pronominal Co-referencer Gate Processing Resource.
Assembling Components Into Pipeline.
Gate is now initialised and the ESTEST application is built.
No JAPE URI specified - default will be used
Configuring Database Gazetteer using file:/C:/estest/dbGaz.xml
Loading definition of Named Entity
  Connecting to HdmStore for schema: estest_store
Loaded 11 values for extent-based NE object <<model>>
Loaded 22 values for extent-based NE object <<colour>>
```

```
Loaded 3 values for extent-based NE object <<manufacturer>>
Loaded 142 values for extent-based NE object <<car_reg>>
Loaded 142 values for extent-based NE object <<car>>
Loaded 11 values for extent-based NE object <<model>>
Loaded 22 values for extent-based NE object <<colour>>
Loaded 3 values for extent-based NE object <<manufacturer>>
Loaded 142 values for extent-based NE object <<car_reg>>

----------------------------------------------------------------

Document to be processed by IE : 'GEORGE BUSH HAS A NEW YELLOW
CAR REGISTRATION LO78 HYS. IT IS A FORD MONDEO.'
Attempting pronominal coreference detection.

AnnotationImpl: id=62; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[62, 63, 64,
60, 61]}; start=NodeImpl: id=15; offset=45; end=NodeImpl: id=20;
offset=54

AnnotationImpl: id=15; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=15; offset=45;
end=NodeImpl: id=16; offset=46

AnnotationImpl: id=30; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=6, string=MONDEO};
start=NodeImpl: id= 30; offset=69; end=NodeImpl: id=31;
offset=75

AnnotationImpl: id=43; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=2; offset=7; end=NodeImpl:
id=3; offset=11

AnnotationImpl: id=16; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=2, string=LO}; start=NodeImpl:
id=16; offset=46; end=NodeImpl: id=17; offset=48

AnnotationImpl: id=31; type=Token; features={string=., length=1,
 kind=punctuation, category=.}; start=NodeImpl: id=31;
offset=75; end=NodeImpl: id=32; offset=76

AnnotationImpl: id=48; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[50, 48, 49]};
start=NodeImpl: id=10; offset=22; end=NodeImpl: id=20; offset=54
```

```
AnnotationImpl: id=59; type=car; features
={idAnnotationType=car_reg, rule=new_car0, kind=car};
start=NodeImpl: id=14; offset=33;end=NodeImpl: id=20; offset=54

AnnotationImpl: id=32; type=Lookup; features
={majorType=OpIntelautzx, minorType=colour}; start=NodeImpl:
id=10; offset=22; end=NodeImpl: id=11; offset=28

AnnotationImpl: id=34; type=Lookup; features
={majorType=OpIntelautzx, minorType=car}; start=NodeImpl: id=16;
offset=46; end=NodeImpl: id=20; offset=54

AnnotationImpl: id=73; type=car; features
={idAnnotationType=car_reg, rule=car_no_reg0, kind=car};
start=NodeImpl: id=30; offset=69; end=NodeImpl: id=31; offset=75

AnnotationImpl: id=41; type=Identifier; features=
{rule1=Identifier1, rule2=IdentifierFinal}; start=NodeImpl:
id=16; offset=46; end=NodeImpl: id=18; offset=50

AnnotationImpl: id=70; type=car; features
={idAnnotationType=car_reg, rule=car_no_reg0, kind=car};
start=NodeImpl: id=28; offset=64; end=NodeImpl: id=31; offset=75

AnnotationImpl: id=60; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[62, 63, 64,
60, 61]}; start=NodeImpl: id=15; offset=45; end=NodeImpl: id=20;
offset=54

AnnotationImpl: id=61; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[62, 63, 64,
60, 61]}; start=NodeImpl: id=15; offset=45; end=NodeImpl: id=20;
offset=54

AnnotationImpl: id=1; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=1; offset=6;
end=NodeImpl: id=2; offset=7

AnnotationImpl: id=29; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=29; offset=68;
end=NodeImpl: id=30; offset=69

AnnotationImpl: id=14; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=12, string=REGISTRATION};
start=NodeImpl: id=14; offset=33; end=NodeImpl: id=15; offset=45
```

```
AnnotationImpl: id=68; type=car; features
={idAnnotationType=car_reg, rule=car_no_reg0, kind=car};
start=NodeImpl: id=27; offset=63; end=NodeImpl: id=31; offset=75

AnnotationImpl: id=47; type=colour; features
={idAnnotationType=colour, rule=colour, kind=colour};
start=NodeImpl: id=10; offset=22; end=NodeImpl: id=11; offset=28

AnnotationImpl: id=58; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[58, 55, 57,
56]}; start=NodeImpl: id=13; offset=32; end=NodeImpl: id=20;
offset=54

AnnotationImpl: id=63; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[62, 63, 64,
60, 61]}; start=NodeImpl: id=15; offset=45; end=NodeImpl: id=20;
offset=54

AnnotationImpl: id=18; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=18; offset=50;
end=NodeImpl: id=19; offset=51

AnnotationImpl: id=3; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=3; offset=11;
end=NodeImpl: id=4; offset=12

AnnotationImpl: id=66; type=car; features
={idAnnotationType=car_reg, rule=new_car0, kind=car};
start=NodeImpl: id=16; offset=46; end=NodeImpl: id=20; offset=54

AnnotationImpl: id=12; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=3, string=CAR}; start=NodeImpl:
id=12; offset=29; end=NodeImpl: id=13; offset=32

AnnotationImpl: id=27; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=27; offset=63;
end=NodeImpl : id=28; offset=64

AnnotationImpl: id=44; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=8; offset=18; end=NodeImpl:
id=9; offset=21

AnnotationImpl: id=64; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[62, 63, 64,
60, 61]}; start=NodeImpl: id=15; offset=45; end=NodeImpl: id=20;
offset=54

AnnotationImpl: id=17; type=Token; features={string=78,
length=2, kind=number, category=CD}; start=NodeImpl: id=17;
offset=48; end=NodeImpl: id=18; offset=50

AnnotationImpl: id=2; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=4, string=BUSH}; start=NodeImpl:
id=2; offset=7; end=NodeImpl: id=3; offset=11

AnnotationImpl: id=13; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=13; offset=32;
end=NodeImpl: id=14; offset=33

AnnotationImpl: id=28; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=4, string=FORD}; start=NodeImpl:
id=28; offset=64; end=NodeImpl: id=29; offset=68

AnnotationImpl: id=38; type=Sentence; features={};
start=NodeImpl: id=0; offset=0; end=NodeImpl: id=21; offset=55

AnnotationImpl: id=57; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[58, 55, 57,
56]}; start=NodeImpl: id=13; offset=32; end=NodeImpl: id=20;
offset=54

AnnotationImpl: id=37; type=Split; features={kind=internal};
start=NodeImpl: id=20; offset=54; end=NodeImpl: id=21; offset=55

AnnotationImpl: id=51; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[51, 53, 52]};
start=NodeImpl: id=11; offset=28; end=NodeImpl: id=20; offset=54

AnnotationImpl: id=8; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=3, string=NEW}; start=NodeImpl:
id=8; offset=18; end=NodeImpl: id=9; offset=21

AnnotationImpl: id=23; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=23; offset=58;
end=NodeImpl: id=24; offset=59

AnnotationImpl: id=72; type=car; features={kind=car,
rule=car_no_reg0, idAnnotationType=car_reg, matches=[72, 71]};
start=NodeImpl: id=29; offset=68; end=NodeImpl: id=31; offset=75
```

```
AnnotationImpl: id=35; type=Lookup; features
={majorType=OpIntelautzx, minorType=manufacturer};
start=NodeImpl: id=28; offset=64; end=NodeImpl: id=29; offset=68

AnnotationImpl: id=7; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=7; offset=17;
end=NodeImpl: id=8; offset=18

AnnotationImpl: id=71; type=car; features={kind=car,
rule=car_no_reg0, idAnnotationType=car_reg, matches=[72, 71]};
start=NodeImpl: id=29; offset=68; end=NodeImpl: id=31; offset=75

AnnotationImpl: id=54; type=car; features
={idAnnotationType=car_reg, rule=new_car0, kind=car};
start=NodeImpl: id=12; offset=29; end=NodeImpl: id=20; offset=54

AnnotationImpl: id=49; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[50, 48, 49]};
start=NodeImpl: id=10; offset=22; end=NodeImpl: id=20; offset=54

AnnotationImpl: id=22; type=Token; features={category=PRP,
kind=word, orth=allCaps, length=2, string=IT}; start=NodeImpl:
id=22; offset=56; end=NodeImpl: id=23; offset=58

AnnotationImpl: id=53; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[51, 53, 52]};
start=NodeImpl: id=11; offset=28; end=NodeImpl: id=20; offset=54

AnnotationImpl: id=9; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=9; offset=21;
end=NodeImpl: id=10; offset=22

AnnotationImpl: id=65; type=car_reg; features
={idAnnotationType=car_reg, rule=car_reg, kind=car_reg};
start=NodeImpl: id=16; offset=46; end=NodeImpl: id=20; offset=54

AnnotationImpl: id=21; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=21; offset=55;
end=NodeImpl: id=22; offset=56

AnnotationImpl: id=33; type=Lookup; features
={majorType=OpIntelautzx, minorType=car_reg}; start=NodeImpl:
id=16; offset=46; end=NodeImpl: id=20; offset=54
```

```
AnnotationImpl: id=6; type=Token; features={category=DT,
kind=word, orth=upperInitial, length=1, string=A};
start=NodeImpl: id=6; offset=16; end=NodeImpl: id=7; offset=17

AnnotationImpl: id=52; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[51, 53, 52]};
start=NodeImpl: id=11; offset=28; end=NodeImpl: id=20; offset=54

AnnotationImpl: id=24; type=Token; features={category=VBZ,
kind=word, orth=allCaps, length=2, string=IS}; start=NodeImpl:
id=24; offset=59; end=NodeImpl: id=25; offset=61

AnnotationImpl: id=42; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=0; offset=0; end=NodeImpl:
id=1; offset=6

AnnotationImpl: id=4; type=Token; features={category=VBZ,
kind=word, orth=allCaps, length=3, string=HAS}; start=NodeImpl:
id=4; offset=12; end=NodeImpl: id=5; offset=15

AnnotationImpl: id=19; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=3, string=HYS}; start=NodeImpl:
id=19; offset=51; end=NodeImpl: id=20; offset=54

AnnotationImpl: id=36; type=Lookup; features
={majorType=OpIntelautzx, minorType=model}; start=NodeImpl:
id=30; offset=69; end=NodeImpl: id=31; offset=75

AnnotationImpl: id=55; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[58, 55, 57,
56]}; start=NodeImpl: id=13; offset=32; end=NodeImpl: id=20;
offset=54

AnnotationImpl: id=26; type=Token; features={category=DT,
kind=word, orth=upperInitial, length=1, string=A};
start=NodeImpl: id=26; offset=62; end=NodeImpl: id=27; offset=63

AnnotationImpl: id=11; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=11; offset=28;
end=NodeImpl: id=12; offset=29

AnnotationImpl: id=67; type=car_reg; features={kind=car_reg,
rule=REGISTRATION_MARK, estestStore=yes,
IdAnnotationType=car_reg}; start=NodeImpl: id=16; offset=46;
end=NodeImpl: id=20; offset=54
```

```

AnnotationImpl: id=50; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[50, 48, 49]};
start=NodeImpl: id=10; offset=22; end=NodeImpl: id=20; offset=54

AnnotationImpl: id=39; type=Sentence; features={};
start=NodeImpl: id=22; offset=56; end=NodeImpl: id=32; offset=76

AnnotationImpl: id=45; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=12; offset=29; end=NodeImpl:
id=13; offset=32

AnnotationImpl: id=74; type=model; features
={idAnnotationType=model, rule=model, kind=model};
start=NodeImpl: id=30; offset=69; end=NodeImpl: id=31; offset=75

AnnotationImpl: id=20; type=Token; features={string=., length=1,
 kind=punctuation, category=.}; start=NodeImpl: id=20;
offset=54; end=NodeImpl: id=21; offset=55

AnnotationImpl: id=46; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=14; offset=33; end=NodeImpl:
id=15; offset=45

AnnotationImpl: id=25; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=25; offset=61;
end=NodeImpl: id=26; offset=62

AnnotationImpl: id=56; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[58, 55, 57,
56]}; start=NodeImpl: id=13; offset=32; end=NodeImpl: id=20;
offset=54

AnnotationImpl: id=10; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=6, string=YELLOW};
start=NodeImpl: id=10; offset=22; end=NodeImpl: id=11; offset=28

AnnotationImpl: id=5; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=5; offset=15;
end=NodeImpl: id=6; offset=16

AnnotationImpl: id=69; type=manufacturer; features
={idAnnotationType=manufacturer, rule=manufacturer,
kind=manufacturer}; start=NodeImpl: id=28; offset=64;
end=NodeImpl: id=29; offset=68
```

```
AnnotationImpl: id=0; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=6, string=GEORGE};
start=NodeImpl: id=0; offset=0; end=NodeImpl: id=1; offset=6


The total concept counts in the DB:
  Concept: car_reg, count: 142
  Concept: op_intel, count: 3
  Concept: manufacturer, count: 3
  Concept: colour_colour, count: 22
  Concept: colour, count: 22
  Concept: model_model, count: 11
  Concept: Resource, count: 0
  Concept: pub, count: 0
  Concept: op_intel_pc, count: 1
  Concept: op_intel_report_id, count: 3
  Concept: car, count: 142
  Concept: opIntel, count: 0
  Concept: op_intel_intel, count: 3
  Concept: model, count: 11
  Concept: manufacturer_manufacturer, count: 3


Checking for annotations that are subsumed by another of the
same type.
  Replacing subsumed annotation 62 with 48
  Replacing subsumed annotation 62 with 59
  Replacing subsumed annotation 62 with 58
  Replacing subsumed annotation 66 with 62
  Replacing subsumed annotation 62 with 51
  Replacing subsumed annotation 62 with 54
  Replacing subsumed annotation 59 with 48
  Replacing subsumed annotation 58 with 48
  Replacing subsumed annotation 66 with 48
  Replacing subsumed annotation 51 with 48
  Replacing subsumed annotation 54 with 48
  Replacing subsumed annotation 59 with 58
  Replacing subsumed annotation 66 with 59
  Replacing subsumed annotation 59 with 51
  Replacing subsumed annotation 59 with 54
  Replacing subsumed annotation 73 with 70
  Replacing subsumed annotation 73 with 68
  Replacing subsumed annotation 73 with 72
  Replacing subsumed annotation 70 with 68
```

```
  Replacing subsumed annotation 72 with 70
  Replacing subsumed annotation 72 with 68
  Replacing subsumed annotation 66 with 58
  Replacing subsumed annotation 58 with 51
  Replacing subsumed annotation 58 with 54
  Replacing subsumed annotation 66 with 51
  Replacing subsumed annotation 66 with 54
  Replacing subsumed annotation 54 with 51


    Annotations of interest
      Annotation Details:
      Schema element = '<<car>>', value = 'YELLOW CAR
REGISTRATION LO78 HYS', Gate ID is 48 and the ID is the value of
the related car_reg
      Schema element = '<<car>>', value = ' FORD MONDEO', Gate
ID is 68 and the ID is the value of the related car_reg
      Schema element = '<<colour>>', value = 'YELLOW', Gate ID
is 47 and the ID is the value of the related colour
      Schema element = '<<car_reg>>', value = 'LO78 HYS', Gate
ID is 65 and the ID is the value of the related car_reg
      Schema element = '<<model>>', value = 'MONDEO', Gate ID is
74 and the ID is the value of the related model
      Schema element = '<<manufacturer>>', value = 'FORD', Gate
ID is 69 and the ID is the value of the related manufacturer
    Adding annotation <<car>> [LO78 HYS]
    Could not find an id annotation so generating one, id:
<<car>> [estestInstance2]
    Adding annotation <<car>> [estestInstance2]
    Adding annotation <<colour>> [YELLOW]
    Adding annotation <<car_reg>> [LO78 HYS]
    Adding annotation <<model>> [MONDEO]
    Adding annotation <<manufacturer>> [FORD]
    Removing templates with only one attribute...
    Template Instances Are:
      <<car>>,estestInstance1
      <<car>>,estestInstance2
    In buildTemplateInstances <<car>>,estestInstance1, colour
    In buildTemplateInstances <<car>>,estestInstance2, model
    2 templates found
    Templates are:
      Template: 1 -- <<car>>, Instance ID: estestInstance1
        Attribute: <<car_reg>>, Instance ID: LO78 HYS
        Attribute: <<colour>>, Instance ID: YELLOW
      Template: 2 -- <<car>>, Instance ID: estestInstance2
        Attribute: <<manufacturer>>, Instance ID: FORD
        Attribute: <<model>>, Instance ID: MONDEO


    merging template1 into3
    Set idAnnotationType to <<car_reg>>for template <<car>>
    Replacing templateInstanceId null with estestInstance1
    Comparing <<car_reg>> to <<car_reg>>
    Merging templates has found a new annotation of the template
id type <<car_reg>>, using this as the id of the template, id is
LO78 HYS
    Comparing <<colour>> to <<car_reg>>
    Changing template instance id from estestInstance1, to LO78
HYS
    merging template2 into3
    Comparing <<manufacturer>> to <<car_reg>>
    Comparing <<model>> to <<car_reg>>
    mergeTemplates() is returning 1 merged templates
    1 templates after merging
    Templates are:
      Template: 3 -- <<car>>, Instance ID: LO78 HYS
        Attribute: <<car_reg>>, Instance ID: LO78 HYS
        Attribute: <<colour>>, Instance ID: YELLOW
        Attribute: <<manufacturer>>, Instance ID: FORD
        Attribute: <<model>>, Instance ID: MONDEO




    Checking templates for matches
      Checking template to see if it should be added or
merged....
        Template: 3 -- <<car>>, Instance ID: LO78 HYS
        Attribute: <<car_reg>>, Instance ID: LO78 HYS
        Attribute: <<colour>>, Instance ID: YELLOW
        Attribute: <<manufacturer>>, Instance ID: FORD
        Attribute: <<model>>, Instance ID: MONDEO




      Evidence of a match with<<car>>,<<car_reg>> --> LO78
HYS,LO78 HYS, weight is 79.78%
```

```
      Evidence of a match with<<car>>,<<manufacturer>> --> BD51
ABC, FORD, weight is 1.69%
      Evidence of a match with<<car>>,<<manufacturer>> --> IK83
OKE, FORD, weight is 1.69%
      Evidence of a match with<<car>>,<<manufacturer>> --> LO78
HYS,FORD, weight is 1.69%
      Evidence of a match with<<car>>,<<model>> --> LO78
HYS,MONDEO, weight is 6.18%
        Match with BD51 ABC, evidence is 1.69%
        Match with LO78 HYS, evidence is 87.64%
        Match with IK83 OKE, evidence is 1.69%
      Best match was LO78 HYS at 87.64%
      Found match with more than 50% likelyhood LO78 HYS,
evidence: 87.64%


    Storing Templates.
      Storing template: <<car>>/LO78 HYS
        Storing template attribute edge
<<attribute,car,car_reg>>[1,LO78 HYS]
        Storing template attribute edge
<<attribute,car,colour>>[1,YELLOW]
        Storing template attribute edge
<<attribute,car,manufacturer>>[1,FORD]
        Storing template attribute edge
<<attribute,car,model>>[1,MONDEO]
      .... and there were none.

----------------------------------------------------------------


    Document to be processed by IE : 'TONY BLAIR SEEN COMING OUT
OF THE PERSEVERANCE PUBLIC HOUSE DRIVES OFF IN A GREEN FORD PUMA
UY22 QWC.'
    Attempting pronominal coreference detection.
    Pronominal Corefs is empty.

AnnotationImpl: id=62; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[62, 60, 61]};
start=NodeImpl: id=28; offset=76; end=NodeImpl: id=38;
offset=100

    AnnotationImpl: id=30; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=4, string=FORD}; start=NodeImpl:
id=30; offset=82; end=NodeImpl: id=31; offset=86
```

```
    AnnotationImpl: id=31; type=SpaceToken;
features={kind=space, length=1, string= }; start=NodeImpl:
id=31; offset=86; end=NodeImpl: id=32; offset=87

    AnnotationImpl: id=86; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[87, 86, 85,
88, 89]}; start=NodeImpl: id=33; offset=91; end=NodeImpl: id=38;
offset=100

    AnnotationImpl: id=59; type=car; features
={idAnnotationType=car_reg, rule=car_no_reg0, kind=car};
start=NodeImpl: id=28; offset=76; end=NodeImpl: id=33; offset=91

    AnnotationImpl: id=34; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=2, string=UY}; start=NodeImpl:
id=34; offset=92; end=NodeImpl: id=35; offset=94

    AnnotationImpl: id=32; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=4, string=PUMA}; start=NodeImpl:
id=32; offset=87; end=NodeImpl: id=33; offset=91

    AnnotationImpl: id=60; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[62, 60, 61]};
start=NodeImpl: id=28; offset=76; end=NodeImpl: id=38;
offset=100

    AnnotationImpl: id=85; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[87, 86, 85,
88, 89]}; start=NodeImpl: id=33; offset=91; end=NodeImpl: id=38;
offset=100

    AnnotationImpl: id=1; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=1; offset=4;
end=NodeImpl: id=2; offset=5

    AnnotationImpl: id=61; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[62, 60, 61]};
start=NodeImpl: id=28; offset=76; end=NodeImpl: id=38;
offset=100

    AnnotationImpl: id=29; type=SpaceToken; features
={kind=space, length=1, string= }; start=NodeImpl: id=29;
offset=81; end=NodeImpl: id=30; offset=82
```

```
    AnnotationImpl: id=68; type=car; features
={idAnnotationType=car_reg, rule=car_no_reg0, kind=car};
start=NodeImpl: id=30; offset=82; end=NodeImpl: id=33; offset=91

    AnnotationImpl: id=58; type=colour;
features={idAnnotationType=colour, rule=colour, kind=colour};
start=NodeImpl: id=28; offset=76; end=NodeImpl: id=29; offset=81

    AnnotationImpl: id=82; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[83, 82, 84]};
start=NodeImpl: id=32; offset=87; end=NodeImpl: id=38;
offset=100

    AnnotationImpl: id=63; type=car; features
={idAnnotationType=car_reg, rule=car_no_reg0, kind=car};
start=NodeImpl: id=29; offset=81; end=NodeImpl: id=33; offset=91

    AnnotationImpl: id=3; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=3; offset=10;
end=NodeImpl: id=4; offset=11

    AnnotationImpl: id=81; type=model; features
={idAnnotationType=model, rule=model, kind=model};
start=NodeImpl: id=32; offset=87; end=NodeImpl: id=33; offset=91

    AnnotationImpl: id=27; type=SpaceToken; features
={kind=space, length=1, string= }; start=NodeImpl: id=27;
offset=75; end=NodeImpl: id=28; offset=76

    AnnotationImpl: id=64; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[65, 66, 64]};
start=NodeImpl: id=29; offset=81; end=NodeImpl: id=38;
offset=100

    AnnotationImpl: id=2; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=5, string=BLAIR};
start=NodeImpl: id=2; offset=5; end=NodeImpl: id=3; offset=10

    AnnotationImpl: id=38; type=Token; features={string=.,
length=1, kind=punctuation, category=.}; start=NodeImpl: id=38;
offset=100; end=NodeImpl: id=39; offset=101

    AnnotationImpl: id=28; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=5, string=GREEN};
start=NodeImpl:id=28; offset=76; end=NodeImpl: id=29; offset=81
```

```
    AnnotationImpl: id=57; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=37; offset=97; end=NodeImpl:
id=38; offset=100

    AnnotationImpl: id=23; type=SpaceToken;
features={kind=space, length=1, string= }; start=NodeImpl:
id=23; offset=70; end=NodeImpl: id=24; offset=71

    AnnotationImpl: id=87; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[87, 86, 85,
88, 89]}; start=NodeImpl: id=33; offset=91; end=NodeImpl: id=38;
offset=100

    AnnotationImpl: id=35; type=Token; features={string=22,
length=2, kind=number, category=CD}; start=NodeImpl: id=35;
offset=94; end=NodeImpl: id=36; offset=96

    AnnotationImpl: id=7; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=7; offset=22;
end=NodeImpl: id=8; offset=23

    AnnotationImpl: id=54; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=20; offset=60; end=NodeImpl:
id=21; offset=66

    AnnotationImpl: id=53; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=18; offset=54; end=NodeImpl:
id=19; offset=59

    AnnotationImpl: id=65; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[65, 66, 64]};
start=NodeImpl: id=29; offset=81; end=NodeImpl: id=38;
offset=100

    AnnotationImpl: id=33; type=SpaceToken; features
={kind=space, length=1, string= }; start=NodeImpl: id=33;
offset=91; end=NodeImpl: id=34; offset=92

    AnnotationImpl: id=6; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=6, string=COMING};
start=NodeImpl: id=6; offset=16; end=NodeImpl: id=7; offset=22
```

```
    AnnotationImpl: id=24; type=Token; features={category=IN,
kind=word, orth=allCaps, length=2, string=IN}; start=NodeImpl:
id=24; offset=71; end=NodeImpl: id=25; offset=73

    AnnotationImpl: id=88; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[87, 86, 85,
88, 89]}; start=NodeImpl: id=33; offset=91; end=NodeImpl: id=38;
offset=100

    AnnotationImpl: id=4; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=4, string=SEEN}; start=NodeImpl:
id=4; offset=11; end=NodeImpl: id=5; offset=15

    AnnotationImpl: id=36; type=SpaceToken;
features={kind=space, length=1, string= }; start=NodeImpl:
id=36; offset=96; end=NodeImpl: id=37; offset=97

    AnnotationImpl: id=26; type=Token; features={category=DT,
kind=word, orth=upperInitial, length=1, string=A};
start=NodeImpl:id=26; offset=74; end=NodeImpl: id=27; offset=75

    AnnotationImpl: id=55; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=22; offset=67; end=NodeImpl:
id=23; offset=70

    AnnotationImpl: id=67; type=manufacturer;
features={idAnnotationType=manufacturer, rule=manufacturer,
kind=manufacturer}; start=NodeImpl: id=30; offset=82;
end=NodeImpl: id=31; offset=86


    AnnotationImpl: id=76; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[79, 78, 75,
76, 77, 74 ]}; start=NodeImpl: id=31; offset=86; end=NodeImpl:
id=38; offset=100

    AnnotationImpl: id=45; type=Identifier; features =
{rule1=Identifier1, rule2=IdentifierFinal}; start=NodeImpl:
id=34; offset=92; end=NodeImpl: id=36; offset=96

    AnnotationImpl: id=74; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[79, 78, 75,
76, 77, 74 ]}; start=NodeImpl: id=31; offset=86; end=NodeImpl:
id=38; offset=100
```

```
    AnnotationImpl: id=46; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=0; offset=0; end=NodeImpl:
id=1; offset=4

    AnnotationImpl: id=25; type=SpaceToken; features
={kind=space, length=1, string= }; start=NodeImpl: id=25;
offset=73; end=NodeImpl: id=26; offset=74

    AnnotationImpl: id=56; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=34; offset=92; end=NodeImpl:
id=35; offset=94

    AnnotationImpl: id=5; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=5; offset=15; end=NodeIm
pl: id=6; offset=16

    AnnotationImpl: id=15; type=SpaceToken; features
={kind=space, length=1, string= }; start=NodeImpl: id=15;
offset=46; end=NodeImpl: id=16; offset=47

    AnnotationImpl: id=79; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[79, 78, 75,
76, 77, 74 ]}; start=NodeImpl: id=31; offset=86; end=NodeImpl:
id=38; offset=100

    AnnotationImpl: id=43; type=Sentence; features={};
start=NodeImpl: id=0; offset=0; end=NodeImpl: id=39; offset=101

    AnnotationImpl: id=16; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=6, string=PUBLIC};
start=NodeImpl:id=16; offset=47; end=NodeImpl: id=17; offset=53

    AnnotationImpl: id=48; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=4; offset=11; end=NodeImpl:
id=5; offset=15

    AnnotationImpl: id=80; type=car; features
={idAnnotationType=car_reg, rule=car_no_reg0, kind=car};
start=NodeImpl: id=32; offset=87; end=NodeImpl: id=33; offset=91

    AnnotationImpl: id=73; type=car; features={kind=car,
rule=car_no_reg0, idAnnotationType=car_reg, matches=[72, 73]};
start=NodeImpl: id=31; offset=86; end=NodeImpl: id=33; offset=91
```

```
    AnnotationImpl: id=41; type=Lookup; features
={majorType=OpIntelautzx, minorType=model}; start=NodeImpl:
id=32; offset=87; end=NodeImpl: id=33; offset=91

    AnnotationImpl: id=70; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[71, 70, 69]};
start=NodeImpl: id=30; offset=82; end=NodeImpl: id=38;
offset=100

    AnnotationImpl: id=14; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=12, string=PERSEVERANCE};
start=NodeImpl: id=14; offset=34; end=NodeImpl: id=15; offset=46

    AnnotationImpl: id=83; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[83, 82, 84]};
start=NodeImpl: id=32; offset=87; end=NodeImpl: id=38;
offset=100

    AnnotationImpl: id=47; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=2; offset=5; end=NodeImpl:
id=3; offset=10

    AnnotationImpl: id=75; type=car; features={kind=car, rule
=new_car0, idAnnotationType=car_reg, matches=[79, 78, 75, 76,
77, 74 ]}; start=NodeImpl: id=31; offset=86; end=NodeImpl:
id=38; offset=100

    AnnotationImpl: id=18; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=5, string=HOUSE};
start=NodeImpl:id=18; offset=54; end=NodeImpl: id=19; offset=59

    AnnotationImpl: id=66; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[65, 66, 64]};
start=NodeImpl: id=29; offset=81; end=NodeImpl: id=38;
offset=100

    AnnotationImpl: id=12; type=Token; features={category=DT,
kind=word, orth=allCaps, length=3, string=THE}; start=NodeImpl:
id=12; offset=30; end=NodeImpl: id=13; offset=33

    AnnotationImpl: id=17; type=SpaceToken;
features={kind=space, length=1, string= }; start=NodeImpl:
id=17; offset=53; end=NodeImpl: id=18; offset=54
```

```
    AnnotationImpl: id=13; type=SpaceToken; features
={kind=space, length=1, string= }; start=NodeImpl: id=13;
offset=33; end=NodeImpl: id=14; offset=34

    AnnotationImpl: id=77; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[79, 78, 75,
76, 77, 74 ]}; start=NodeImpl: id=31; offset=86; end=NodeImpl:
id=38; offset=100

    AnnotationImpl: id=84; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[83, 82, 84]};
start=NodeImpl: id=32; offset=87; end=NodeImpl: id=38;
offset=100

    AnnotationImpl: id=37; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=3, string=QWC}; start=NodeImpl:
id=37; offset=97; end=NodeImpl: id=38; offset=100

    AnnotationImpl: id=51; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=14; offset=34; end=NodeImpl:
id=15; offset=46

    AnnotationImpl: id=8; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=3, string=OUT}; start=NodeImpl:
id=8; offset=23; end=NodeImpl: id=9; offset=26

    AnnotationImpl: id=40; type=Lookup; features
={majorType=OpIntelautzx, minorType=manufacturer};
start=NodeImpl: id=30; offset=82; end=NodeImpl: id=31; offset=86

    AnnotationImpl: id=72; type=car; features={kind=car,
rule=car_no_reg0, idAnnotationType=car_reg, matches=[72, 73]};
start=NodeImpl: id=31; offset=86; end=NodeImpl: id=33; offset=91

    AnnotationImpl: id=49; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=6; offset=16; end=NodeImpl:
id=7; offset=22

    AnnotationImpl: id=71; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[71, 70, 69]};
start=NodeImpl: id=30; offset=82; end=NodeImpl: id=38;
offset=100
```

```
    AnnotationImpl: id=22; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=3, string=OFF}; start=NodeImpl:
id=22; offset=67; end=NodeImpl: id=23; offset=70

    AnnotationImpl: id=9; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=9; offset=26;
end=NodeImpl: id=10; offset=27

    AnnotationImpl: id=21; type=SpaceToken;
features={kind=space, length=1, string= }; start=NodeImpl:
id=21; offset=66; end=NodeImpl: id=22; offset=67

    AnnotationImpl: id=78; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[79, 78, 75,
76, 77, 74 ]}; start=NodeImpl: id=31; offset=86; end=NodeImpl:
id=38; offset=100

    AnnotationImpl: id=52; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=16; offset=47; end=NodeImpl:
id=17; offset=53

    AnnotationImpl: id=42; type=Split; features={kind=internal};
start=NodeImpl: id=38; offset=100; end=NodeImpl: id=39; offset=1
01

    AnnotationImpl: id=90; type=car; features
={idAnnotationType=car_reg, rule=new_car0, kind=car};
start=NodeImpl: id=34; offset=92; end=NodeImpl: id=38;
offset=100

    AnnotationImpl: id=19; type=SpaceToken; features
={kind=space, length=1, string= }; start=NodeImpl: id=19;
offset=59; end=NodeImpl: id=20; offset=60

    AnnotationImpl: id=11; type=SpaceToken;
features={kind=space, length=1, string= }; start=NodeImpl:
id=11; offset=29; end=NodeImpl: id=12; offset=30

    AnnotationImpl: id=50; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=8; offset=23; end=NodeImpl:
id=9; offset=26

    AnnotationImpl: id=91; type=car_reg; features={kind=car_reg,
rue=REGISTRATION_MARK, estestStore=yes,
idAnnotationType=car_reg}; start=NodeImpl: id=34; offset=92;
end=NodeImpl: id=38; offset=100

    AnnotationImpl: id=39; type=Lookup; features
={majorType=OpIntelautzx, minorType=colour}; start=NodeImpl:
id=28; offset=76; end=NodeImpl: id=29; offset=81

    AnnotationImpl: id=89; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[87, 86, 85,
88, 89]}; start=NodeImpl: id=33; offset=91; end=NodeImpl: id=38;
offset=100

    AnnotationImpl: id=20; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=6, string=DRIVES};
start=NodeImpl:id=20; offset=60; end=NodeImpl: id=21; offset=66

    AnnotationImpl: id=10; type=Token; features={category=IN,
kind=word, orth=allCaps, length=2, string=OF}; start=NodeImpl:
id=10; offset=27; end=NodeImpl: id=11; offset=29

    AnnotationImpl: id=0; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=4, string=TONY}; start=NodeImpl:
id=0; offset=0; end=NodeImpl: id=1; offset=4

    AnnotationImpl: id=69; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[71, 70, 69]};
start=NodeImpl: id=30; offset=82; end=NodeImpl: id=38;
offset=100

   The total concept counts in the DB:
      Concept: car_reg, count: 142
      Concept: op_intel, count: 3
      Concept: manufacturer, count: 3
      Concept: colour_colour, count: 22
      Concept: colour, count: 22
      Concept: model_model, count: 11
      Concept: Resource, count: 0
      Concept: pub, count: 0
      Concept: op_intel_pc, count: 1
      Concept: op_intel_report_id, count: 3
      Concept: car, count: 142
      Concept: opIntel, count: 0
      Concept: op_intel_intel, count: 3
      Concept: model, count: 11
      Concept: manufacturer_manufacturer, count: 3
```

```
    Checking for annotations that are subsumed by another of the
same type.
      Replacing subsumed annotation 86 with 62
      Replacing subsumed annotation 59 with 62
      Replacing subsumed annotation 68 with 62
      Replacing subsumed annotation 82 with 62
      Replacing subsumed annotation 63 with 62
      Replacing subsumed annotation 64 with 62
      Replacing subsumed annotation 76 with 62
      Replacing subsumed annotation 80 with 62
      Replacing subsumed annotation 73 with 62
      Replacing subsumed annotation 70 with 62
      Replacing subsumed annotation 90 with 62
      Replacing subsumed annotation 86 with 82
      Replacing subsumed annotation 86 with 64
      Replacing subsumed annotation 86 with 76
      Replacing subsumed annotation 86 with 70
      Replacing subsumed annotation 90 with 86
      Replacing subsumed annotation 68 with 59
      Replacing subsumed annotation 63 with 59
      Replacing subsumed annotation 80 with 59
      Replacing subsumed annotation 73 with 59
      Replacing subsumed annotation 68 with 63
      Replacing subsumed annotation 68 with 64
      Replacing subsumed annotation 80 with 68
      Replacing subsumed annotation 73 with 68
      Replacing subsumed annotation 68 with 70
      Replacing subsumed annotation 82 with 64
      Replacing subsumed annotation 82 with 76
      Replacing subsumed annotation 80 with 82
      Replacing subsumed annotation 82 with 70
      Replacing subsumed annotation 90 with 82
      Replacing subsumed annotation 63 with 64
      Replacing subsumed annotation 80 with 63
      Replacing subsumed annotation 73 with 63
      Replacing subsumed annotation 76 with 64
      Replacing subsumed annotation 80 with 64
      Replacing subsumed annotation 73 with 64
      Replacing subsumed annotation 70 with 64
      Replacing subsumed annotation 90 with 64
      Replacing subsumed annotation 80 with 76
      Replacing subsumed annotation 73 with 76
      Replacing subsumed annotation 76 with 70
      Replacing subsumed annotation 90 with 76
      Replacing subsumed annotation 80 with 73
      Replacing subsumed annotation 80 with 70
      Replacing subsumed annotation 73 with 70
      Replacing subsumed annotation 90 with 70


        Annotations of interest
        Annotation Details:
        Schema element = '<<car>>', value = 'GREEN FORD PUMA
UY22 QWC', Gate ID is 62 and the ID is the value of the related
car_reg
        Schema element = '<<colour>>', value = 'GREEN', Gate ID
is 58 and the ID is the value of the related colour
        Schema element = '<<model>>', value = 'PUMA', Gate ID is
81 and the ID is the value of the related model
        Schema element = '<<manufacturer>>', value = 'FORD',
Gate ID is 67 and the ID is the value of the related
manufacturer
        Schema element = '<<car_reg>>', value = 'UY22 QWC', Gate
ID is 91 and the ID is the value of the related car_reg
      Adding annotation <<car>> [UY22 QWC]
      Adding annotation <<colour>> [GREEN]
      Adding annotation <<model>> [PUMA]
      Adding annotation <<manufacturer>> [FORD]
      Adding annotation <<car_reg>> [UY22 QWC]
      Removing templates with only one attribute...
      Template Instances Are:
        <<car>>,estestInstance8
      In buildTemplateInstances <<car>>,estestInstance8, colour
      1 templates found
      Templates are:
        Template: 4 -- <<car>>, Instance ID: estestInstance8
        Attribute: <<car_reg>>, Instance ID: UY22 QWC
        Attribute: <<colour>>, Instance ID: GREEN
        Attribute: <<manufacturer>>, Instance ID: FORD
        Attribute: <<model>>, Instance ID: PUMA


    merging template4 into5
    Set idAnnotationType to <<car_reg>>for template <<car>>
    Replacing templateInstanceId null with estestInstance8
    Comparing <<car_reg>> to <<car_reg>>
```

```
    Merging templates has found a new annotation of the template
id type <<car_reg>>, using this as the id of the template, id is
UY22 QWC
    Comparing <<colour>> to <<car_reg>>
    Comparing <<manufacturer>> to <<car_reg>>
    Comparing <<model>> to <<car_reg>>
    Changing template instance id from estestInstance8, to UY22
QWC
    mergeTemplates() is returning 1 merged templates
    1 templates after merging
    Templates are:
      Template: 5 -- <<car>>, Instance ID: UY22 QWC
        Attribute: <<car_reg>>, Instance ID: UY22 QWC
        Attribute: <<colour>>, Instance ID: GREEN
        Attribute: <<manufacturer>>, Instance ID: FORD
        Attribute: <<model>>, Instance ID: PUMA




    Checking templates for matches
      Checking template to see if it should be added or
merged....
        Template: 5 -- <<car>>, Instance ID: UY22 QWC
        Attribute: <<car_reg>>, Instance ID: UY22 QWC
        Attribute: <<colour>>, Instance ID: GREEN
        Attribute: <<manufacturer>>, Instance ID: FORD
        Attribute: <<model>>, Instance ID: PUMA




      Evidence of a match with<<car>>,<<colour>> --> IK83
OKE,GREEN, weight is 12.36%
      Evidence of a match with<<car>>,<<manufacturer>> --> LO78
HYS,FORD, weight is 1.69%
      Evidence of a match with<<car>>,<<manufacturer>> --> BD51
ABC,FORD, weight is 1.69%
      Evidence of a match with<<car>>,<<manufacturer>> --> IK83
OKE,FORD, weight is 1.69%
      Evidence of a match with<<car>>,<<manufacturer>> --> LO78
HYS,FORD, weight is 1.69%
      Evidence of a match with<<car>>,<<model>> --> BD51
ABC,PUMA, weight is 6.18%
        Match with BD51 ABC, evidence is 7.87%
        Match with LO78 HYS, evidence is 3.37%
        Match with IK83 OKE, evidence is 14.04%
      Best match was IK83 OKE at 14.04%


    Storing Templates.
      Storing template: <<car>>/UY22 QWC
        Storing template attribute edge
<<attribute,car,car_reg>>[2,UY22 QWC]
        Storing template attribute edge
<<attribute,car,colour>>[2,GREEN]
        Storing template attribute edge
<<attribute,car,manufacturer>>[2,FORD]
        Storing template attribute edge
<<attribute,car,model>>[2,PUMA]
      .... and there were none.

----------------------------------------------------------------

    Document to be processed by IE : 'NICHOLAS SARKOZY NOW
DRIVING BLUE CITRON 2CV CE21 FGH.'
    Attempting pronominal coreference detection.
    Pronominal Corefs is empty.

    AnnotationImpl: id=62; type=model;
features={idAnnotationType=model, rule=model, kind=model};
start=NodeImpl: id=12; offset=41; end=NodeImpl: id=14; offset=44

    AnnotationImpl: id=15; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=2, string=CE}; start=NodeImpl:
id=15; offset=45; end=NodeImpl: id=16; offset=47

    AnnotationImpl: id=30; type=Identifier; features
={rule1=Identifier1, rule2=IdentifierFinal}; start=NodeImpl:
id=15; offset=45; end=NodeImpl: id=17; offset=49

    AnnotationImpl: id=43; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[43, 41, 42,
44]}; start=NodeImpl: id=9; offset=33; end=NodeImpl: id=19;
offset=53

    AnnotationImpl: id=16; type=Token; features={string=21,
length=2, kind=number, category=CD}; start=NodeImpl: id=16;
offset=47; end=NodeImpl: id=17; offset=49
```

```
    AnnotationImpl: id=31; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=0; offset=0; end=NodeImpl:
id=1; offset=8

    AnnotationImpl: id=48; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[48, 49, 47,
50]}; start=NodeImpl: id=10; offset=34; end=NodeImpl: id=19;
offset=53

    AnnotationImpl: id=59; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[54, 59, 53,
60, 58, 55, 57, 56]}; start=NodeImpl: id=11; offset=40;
end=NodeImpl: id=19; offset=53

    AnnotationImpl: id=32; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=2; offset=9; end=NodeImpl:
id=3; offset=16

    AnnotationImpl: id=34; type=colour; features
={idAnnotationType=colour, rule=colour, kind=colour};
start=NodeImpl: id=8; offset=29; end=NodeImpl: id=9; offset=33

    AnnotationImpl: id=73; type=car; features
={idAnnotationType=car_reg, rule=new_car0, kind=car};
start=NodeImpl: id=15; offset=45; end=NodeImpl: id=19; offset=53

    AnnotationImpl: id=41; type=car; features={kind=car,
rule=known_car0, idAnnotationType=car_reg, matches=[43, 41, 42,
44]}; start=NodeImpl: id=9; offset=33; end=NodeImpl: id=19;
offset=53

    AnnotationImpl: id=70; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[71, 70, 68,
67, 69]}; start=NodeImpl: id=14; offset=44; end=NodeImpl: id=19;
offset=53

    AnnotationImpl: id=60; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[54, 59, 53,
60, 58, 55, 57, 56]}; start=NodeImpl: id=11; offset=40;
end=NodeImpl: id=19; offset=53

    AnnotationImpl: id=61; type=car; features
={idAnnotationType=car_reg, rule=car_no_reg0, kind=car};
start=NodeImpl: id=12; offset=41; end=NodeImpl: id=14; offset=44
```

```
    AnnotationImpl: id=1; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=1; offset=8; end=NodeImp
l: id=2; offset=9

    AnnotationImpl: id=29; type=Identifier; features
={rule1=Identifier1, rule2=IdentifierFinal}; start=NodeImpl:
id=12; offset=41; end=NodeImpl: id=14; offset=44

    AnnotationImpl: id=14; type=SpaceToken; features
={kind=space, length=1, string= }; start=NodeImpl: id=14;
offset=44; end=NodeImpl: id=15; offset=45

    AnnotationImpl: id=68; type=car; features={kind=car, rule
=new_car0, idAnnotationType=car_reg, matches=[71, 70, 68, 67,
69]}; start=NodeImpl: id=14; offset=44; end=NodeImpl: id=19;
offset=53

    AnnotationImpl: id=47; type=car; features={kind=car, rule
=known_car0, idAnnotationType=car_reg, matches=[48, 49, 47,
50]}; start=NodeImpl: id=10; offset=34; end=NodeImpl: id=19;
offset=53


    AnnotationImpl: id=58; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[54, 59, 53,
60, 58, 55, 57, 56]}; start=NodeImpl: id=11; offset=40;
end=NodeImpl: id=19; offset=53

    AnnotationImpl: id=63; type=car; features={kind=car,
rule=known_car0, idAnnotationType=car_reg, matches=[65, 63, 66,
64]}; start=NodeImpl: id=12; offset=41; end=NodeImpl: id=19;
offset=53

    AnnotationImpl: id=18; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=3, string=FGH}; start=NodeImpl:
id=18; offset=50; end=NodeImpl: id=19; offset=53

    AnnotationImpl: id=3; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=3; offset=16; end=NodeIm
pl: id=4; offset=17

    AnnotationImpl: id=66; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[65, 63, 66,
64]}; start=NodeImpl: id=12; offset=41; end=NodeImpl: id=19;
offset=53
```

```

    AnnotationImpl: id=12; type=Token; features={string=2,
length=1, kind=number, category=CD}; start=NodeImpl: id=12;
offset=41; end=NodeImpl: id=13; offset=42

    AnnotationImpl: id=44; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[43, 41, 42,
44]}; start=NodeImpl: id=9; offset=33; end=NodeImpl: id=19;
offset=53

    AnnotationImpl: id=64; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[65, 63, 66,
64]}; start=NodeImpl: id=12; offset=41; end=NodeImpl: id=19;
offset=53

    AnnotationImpl: id=17; type=SpaceToken; features
={kind=space, length=1, string= }; start=NodeImpl: id=17;
offset=49; end=NodeImpl: id=18; offset=50

    AnnotationImpl: id=2; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=7, string=SARKOZY};
start=NodeImpl: id=2; offset=9; end=NodeImpl: id=3; offset=16

    AnnotationImpl: id=13; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=2, string=CV}; start=NodeImpl:
id=13; offset=42; end=NodeImpl: id=14; offset=44

    AnnotationImpl: id=38; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[36, 39, 38,
37]}; start=NodeImpl: id=8; offset=29; end=NodeImpl: id=19;
offset=53

    AnnotationImpl: id=57; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[54, 59, 53,
60, 58, 55, 57, 56]}; start=NodeImpl: id=11; offset=40;
end=NodeImpl: id=19; offset=53

    AnnotationImpl: id=37; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[36, 39, 38,
37]}; start=NodeImpl: id=8; offset=29; end=NodeImpl: id=19;
offset=53

    AnnotationImpl: id=51; type=car; features={kind=car,
rule=car_no_reg0, idAnnotationType=car_reg, matches=[51, 52]};
start=NodeImpl: id=11; offset=40; end=NodeImpl: id=14; offset=44
```

```
    AnnotationImpl: id=8; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=4, string=BLUE}; start=NodeImpl:
id=8; offset=29; end=NodeImpl: id=9; offset=33

    AnnotationImpl: id=23; type=Lookup; features
={majorType=OpIntelautzx, minorType=car_reg}; start=NodeImpl:
id=15; offset=45; end=NodeImpl: id=19; offset=53

    AnnotationImpl: id=40; type=car; features
={idAnnotationType=car_reg, rule=car_no_reg0, kind=car};
start=NodeImpl: id=9; offset=33; end=NodeImpl: id=14; offset=44

    AnnotationImpl: id=72; type=car_reg; features
={idAnnotationType=car_reg, rule=car_reg, kind=car_reg};
start=NodeImpl: id=15; offset=45; end=NodeImpl: id=19; offset=53

    AnnotationImpl: id=35; type=car; features
={idAnnotationType=car_reg, rule=car_no_reg0, kind=car};
start=NodeImpl: id=8; offset=29; end=NodeImpl: id=14; offset=44

    AnnotationImpl: id=7; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=7; offset=28; end=NodeIm
pl: id=8; offset=29

    AnnotationImpl: id=71; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[71, 70, 68,
67, 69]}; start=NodeImpl: id=14; offset=44; end=NodeImpl: id=19;
offset=53

    AnnotationImpl: id=54; type=car; features={kind=car,
rule=known_car0, idAnnotationType=car_reg, matches=[54, 59, 53,
60, 58, 55, 57, 56]}; start=NodeImpl: id=11; offset=40;
end=NodeImpl:id=19; offset=53

    AnnotationImpl: id=49; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[48, 49, 47,
50]}; start=NodeImpl: id=10; offset=34; end=NodeImpl: id=19;
offset=53

    AnnotationImpl: id=22; type=Lookup; features
={majorType=OpIntelautzx, minorType=model}; start=NodeImpl:
id=12; offset=41; end=NodeImpl: id=14; offset=44
```

```
    AnnotationImpl: id=53; type=car; features={kind=car,
rule=known_car0, idAnnotationType=car_reg, matches=[54, 59, 53,
60, 58, 55, 57, 56]}; start=NodeImpl: id=11; offset=40;
end=NodeImpl: id=19; offset=53

    AnnotationImpl: id=9; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=9; offset=33; end=NodeIm
pl: id=10; offset=34

    AnnotationImpl: id=65; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[65, 63, 66,
64]}; start=NodeImpl: id=12; offset=41; end=NodeImpl: id=19;
offset=53

    AnnotationImpl: id=21; type=Lookup; features
={majorType=OpIntelautzx, minorType=manufacturer};
start=NodeImpl: id=10; offset=34; end=NodeImpl: id=11; offset=40

    AnnotationImpl: id=33; type=Unknown; features={kind=PN,
rule=Unknown}; start=NodeImpl: id=6; offset=21; end=NodeImpl:
id=7; offset=28

    AnnotationImpl: id=6; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=7, string=DRIVING};
start=NodeImpl:id=6; offset=21; end=NodeImpl: id=7; offset=28

    AnnotationImpl: id=52; type=car; features={kind=car,
rule=car_no_reg0, idAnnotationType=car_reg, matches=[51, 52]};
start=NodeImpl: id=11; offset=40; end=NodeImpl: id=14; offset=44

    AnnotationImpl: id=24; type=Lookup; features=
{majorType=OpIntelautzx, minorType=car}; start=NodeImpl: id=15;
offset=45; end=NodeImpl: id=19; offset=53

    AnnotationImpl: id=42; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[43, 41, 42,
44]}; start=NodeImpl: id=9; offset=33; end=NodeImpl: id=19;
offset=53

    AnnotationImpl: id=4; type=Token; features={category=RB,
kind=word, orth=allCaps, length=3, string=NOW}; start=NodeImpl:
id=4; offset=17; end=NodeImpl: id=5; offset=20
```

```
    AnnotationImpl: id=19; type=Token; features={string=.,
length=1, kind=punctuation, category=.}; start=NodeImpl: id=19;
offset=53; end=NodeImpl: id=20; offset=54

    AnnotationImpl: id=36; type=car; features={kind=car,
rule=known_car0, idAnnotationType=car_reg, matches=[36, 39, 38,
37]}; start=NodeImpl: id=8; offset=29; end=NodeImpl: id=19;
offset=53

     AnnotationImpl: id=55; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[54, 59, 53,
60, 58, 55, 57, 56]}; start=NodeImpl: id=11; offset=40;
end=NodeImpl: id=19; offset=53

    AnnotationImpl: id=26; type=Sentence; features={};
start=NodeImpl: id=0; offset=0; end=NodeImpl: id=20; offset=54

    AnnotationImpl: id=11; type=SpaceToken; features
={kind=space, length=1, string= }; start=NodeImpl: id=11;
offset=40; end=NodeImpl: id=12; offset=41

    AnnotationImpl: id=67; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[71, 70, 68,
67, 69]}; start=NodeImpl: id=14; offset=44; end=NodeImpl: id=19;
offset=53

    AnnotationImpl: id=50; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[48, 49, 47,
50]}; start=NodeImpl: id=10; offset=34; end=NodeImpl: id=19;
offset=53

    AnnotationImpl: id=39; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[36, 39, 38,
37]}; start=NodeImpl: id=8; offset=29; end=NodeImpl: id=19;
offset=53

    AnnotationImpl: id=45; type=manufacturer; features
={idAnnotationType=manufacturer, rule=manufacturer,
kind=manufacturer}; start=NodeImpl: id=10; offset=34;
end=NodeImpl: id=11; offset=40

    AnnotationImpl: id=74; type=car_reg; features={kind=car_reg,
rule=REGISTRATION_MARK, estestStore=yes, idAnnotationType=car_re
g}; start=NodeImpl: id=15; offset=45; end=NodeImpl: id=19;
offset=53
```

```
   AnnotationImpl: id=20; type=Lookup; features
={majorType=OpIntelautzx, minorType=colour}; start=NodeImpl:
id=8; offset=29; end=NodeImpl: id=9; offset=33

    AnnotationImpl: id=46; type=car; features
={idAnnotationType=car_reg, rule=car_no_reg0, kind=car};
start=NodeImpl: id=10; offset=34; end=NodeImpl: id=14; offset=44

    AnnotationImpl: id=25; type=Split; features={kind=internal};
start=NodeImpl: id=19; offset=53; end=NodeImpl: id=20; offset=54


    AnnotationImpl: id=56; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[54, 59, 53,
60, 58, 55, 57, 56]}; start=NodeImpl: id=11; offset=40;
end=NodeImpl: id=19; offset=53

    AnnotationImpl: id=10; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=6, string=CITRON};
start=NodeImpl:id=10; offset=34; end=NodeImpl: id=11; offset=40

    AnnotationImpl: id=5; type=SpaceToken; features={kind=space,
length=1, string= }; start=NodeImpl: id=5; offset=20; end=NodeIm
pl: id=6; offset=21

    AnnotationImpl: id=69; type=car; features={kind=car,
rule=new_car0, idAnnotationType=car_reg, matches=[71, 70, 68,
67, 69]}; start=NodeImpl: id=14; offset=44; end=NodeImpl: id=19;
offset=53

    AnnotationImpl: id=0; type=Token; features={category=NNP,
kind=word, orth=allCaps, length=8, string=NICHOLAS};
start=NodeImpl: id=0; offset=0; end=NodeImpl: id=1; offset=8


    The total concept counts in the DB:
      Concept: car_reg, count: 143
      Concept: op_intel, count: 3
      Concept: manufacturer, count: 3
      Concept: colour_colour, count: 22
      Concept: colour, count: 22
      Concept: model_model, count: 11
      Concept: Resource, count: 0
      Concept: pub, count: 0
      Concept: op_intel_pc, count: 1
      Concept: op_intel_report_id, count: 3
      Concept: car, count: 143
      Concept: opIntel, count: 0
      Concept: op_intel_intel, count: 3
      Concept: model, count: 11
      Concept: manufacturer_manufacturer, count: 3


    Checking for annotations that are subsumed by another of the
same type.
      Replacing subsumed annotation 48 with 43
      Replacing subsumed annotation 59 with 43
      Replacing subsumed annotation 73 with 43
      Replacing subsumed annotation 70 with 43
      Replacing subsumed annotation 61 with 43
      Replacing subsumed annotation 63 with 43
      Replacing subsumed annotation 43 with 38
      Replacing subsumed annotation 51 with 43
      Replacing subsumed annotation 40 with 43
      Replacing subsumed annotation 46 with 43
      Replacing subsumed annotation 59 with 48
      Replacing subsumed annotation 73 with 48
      Replacing subsumed annotation 70 with 48
      Replacing subsumed annotation 61 with 48
      Replacing subsumed annotation 63 with 48
      Replacing subsumed annotation 48 with 38
      Replacing subsumed annotation 51 with 48
      Replacing subsumed annotation 46 with 48
      Replacing subsumed annotation 73 with 59
      Replacing subsumed annotation 70 with 59
      Replacing subsumed annotation 61 with 59
      Replacing subsumed annotation 63 with 59
      Replacing subsumed annotation 59 with 38
      Replacing subsumed annotation 51 with 59
      Replacing subsumed annotation 73 with 70
      Replacing subsumed annotation 73 with 63
      Replacing subsumed annotation 73 with 38
      Replacing subsumed annotation 70 with 63
      Replacing subsumed annotation 70 with 38
      Replacing subsumed annotation 61 with 63
      Replacing subsumed annotation 61 with 38
      Replacing subsumed annotation 61 with 51
      Replacing subsumed annotation 61 with 40
      Replacing subsumed annotation 61 with 35
```

```
      Replacing subsumed annotation 61 with 46
      Replacing subsumed annotation 63 with 38
      Replacing subsumed annotation 51 with 38
      Replacing subsumed annotation 40 with 38
      Replacing subsumed annotation 35 with 38
      Replacing subsumed annotation 46 with 38
      Replacing subsumed annotation 51 with 40
      Replacing subsumed annotation 51 with 35
      Replacing subsumed annotation 51 with 46
      Replacing subsumed annotation 40 with 35
      Replacing subsumed annotation 46 with 40
      Replacing subsumed annotation 46 with 35


        Annotations of interest
        Annotation Details:
        Schema element = '<<model>>', value = '2CV', Gate ID is
62 and the ID is the value of the related model
        Schema element = '<<colour>>', value = 'BLUE', Gate ID
is 34 and the ID is the value of the related colour
        Schema element = '<<car>>', value = 'BLUE CITRON 2CV
CE21 FGH', Gate ID is 38 and the ID is the value of the related
car_reg
        Schema element = '<<car_reg>>', value = 'CE21 FGH', Gate
ID is 72 and the ID is the value of the related car_reg
        Schema element = '<<manufacturer>>', value = 'CITRON',
Gate ID is 45 and the ID is the value of the related
manufacturer
      Adding annotation <<model>> [2CV]
      Adding annotation <<colour>> [BLUE]
      Adding annotation <<car>> [CE21 FGH]
      Adding annotation <<car_reg>> [CE21 FGH]
      Adding annotation <<manufacturer>> [CITRON]
      Removing templates with only one attribute...
      Template Instances Are:
        <<car>>,estestInstance15
      In buildTemplateInstances <<car>>,estestInstance15, model
      1 templates found
      Templates are:
        Template: 6 -- <<car>>, Instance ID: estestInstance15

        Attribute: <<car_reg>>, Instance ID: CE21 FGH
        Attribute: <<colour>>, Instance ID: BLUE
        Attribute: <<manufacturer>>, Instance ID: CITRON
        Attribute: <<model>>, Instance ID: 2CV
```

```
    merging template6 into7
    Set idAnnotationType to <<car_reg>>for template <<car>>
    Replacing templateInstanceId null with estestInstance15
    Comparing <<car_reg>> to <<car_reg>>
    Merging templates has found a new annotation of the template
id type <<car_reg>>, using this as the id of the template, id is
CE21 FGH
    Comparing <<colour>> to <<car_reg>>
    Comparing <<manufacturer>> to <<car_reg>>
    Comparing <<model>> to <<car_reg>>
    Changing template instance id from estestInstance15, to CE21
FGH
    mergeTemplates() is returning 1 merged templates
    1 templates after merging
    Templates are:
      Template: 7 -- <<car>>, Instance ID: CE21 FGH
        Attribute: <<car_reg>>, Instance ID: CE21 FGH
        Attribute: <<colour>>, Instance ID: BLUE
        Attribute: <<manufacturer>>, Instance ID: CITRON
        Attribute: <<model>>, Instance ID: 2CV

    Checking templates for matches
      Checking template to see if it should be added or
merged....
        Template: 7 -- <<car>>, Instance ID: CE21 FGH
        Attribute: <<car_reg>>, Instance ID: CE21 FGH
        Attribute: <<colour>>, Instance ID: BLUE
        Attribute: <<manufacturer>>, Instance ID: CITRON
        Attribute: <<model>>, Instance ID: 2CV


      Evidence of a match with<<car>>,<<car_reg>> --> CE21
FGH,CE21 FGH, weight is 79.89%
      Contradiction with existing edge: <<car>>,<<colour>> -->
CE21 FGH, the annotation attribute is BLUE while the db
attribute is RED evidence is-12.29%
      Evidence of a match with<<car>>,<<colour>> --> BD51
ABC,BLUE, weight is 12.29%
      Evidence of a match with<<car>>,<<manufacturer>> --> CE21
FGH, CITRON, weight is 1.68%
      Evidence of a match with<<car>>,<<model>> --> CE21
FGH,2CV, weight is 6.15%
        Match with BD51 ABC, evidence is 12.29%
```

```
        Match with CE21 FGH, evidence is 75.42%
      Best match was CE21 FGH at 75.42%
      Found match with more than 50% likelyhood CE21 FGH,
evidence: 75.42%


    Storing Templates.
      Storing template: <<car>>/CE21 FGH
        Storing template attribute edge
<<attribute,car,car_reg>>[3,CE21 FGH]
        Storing template attribute edge
<<attribute,car,colour>>[3,BLUE]
        Storing template attribute edge
<<attribute,car,manufacturer>>[3,CITRON]
        Storing template attribute edge
<<attribute,car,model>>[3,2CV]
      .... and there were none.
    Closing debug log file.
```

List of Acronyms

Abstract Syntax Graph (ASG), 56

Both-As-View (BAV), 26

Central Intelligence Agency (CIA), 40

Database Management Systems (DBMS), 20

Data Source Repository (DSR), 59

ESTEST Data Model (EDM), 70

ESTEST Metadata Repository (EMR), 74

Experimental System To Extract Structure from Text (ESTEST), 18

General Architecture for Text Engineering (GATE), 36

Global-As-View (GAV), 26

Global-Local-As-View (GLAV), 26

Graphical User Interface (GUI), 47

Hypergraph Data Model (HDM), 25

Information Extraction (IE), 16

Information Retrieval (IR), 15

Intermediate Query Language (IQL), 45

Java Annotation Patterns Engine (JAPE), 153

Java WordNet Library (JWNL), 74

Local-As-View (LAV), 26

Model Definition Repository (MDR), 59

Message Understanding Conference series (MUC), 33

National Institute of Standards and Technology (NIST), 40

Natural Language Processing (NLP), 15

Resource Description Framework (RDF), 25

Resource Description Framework Schema (RDFS), 47

Road Traffic Accidents (RTA), 87

Schemas & Transformations Repository (STR), 45

Uniform Resource Identifier (URI), 48

Unstructured Information Management Architecture (UIMA), 36

Glossary

anaphora resolution

Linguistics task consisting of identifying references in text to some previously mentioned item.

annotation

In IE, the result of running a processing resource is a set of annotations over the text. Each annotation has a start and an end position in the text, it has a type, and may have features.

annotation chains

Where coreference detection has found annotations referring to the same entity, these are linked into annotation chains.

binary-relational data model

In this data model, every real-world concept of interest is an entity type and associations between entity types are modelled by binary relationships

both-as-view (BAV)

Data integration approach based on the use of reversible sequences of primitive schema transformations, termed pathways.

coreference detection (or coreference annotation)

NLP task which determines when some text contains multiple references to the same entity.

data integration

A unified view of data is provided over a number of data sources each of which may be structured according to different data models.

ESTEST data model (EDM)

Data model used by ESTEST which defines other data models in terms of concepts, attributes and an isA hierarchy.

ESTEST metadata repository (EMR)

AutoMed repository for the schema element metadata extracted from data sources by ESTEST, such as word forms and type information.

global-as-view (GAV)

Data integration approach where the global schema is defined as a set of views over the data sources.

global-local-as-view (GLAV)

Variation of LAV which combines the expressive power of GAV and LAV.

hypergraph data model (HDM)

Graph-based data model used in the AutoMed system.

information extraction (IE)

A branch of NLP where pre-defined entitles are extracted from text.

information retrieval (IR)

A collection of documents is examined and, with respect to a user query, an ordered list of potentially relevant documents is found.

java annotation patterns engine (JAPE)

GATE component which provides finite state transduction over annotations based on regular expressions.

language engineering

Application of a software engineering approach to the development of NLP applications e.g. to promote reuse of components.

local-as-view (LAV)

Data integration approach where the global schema is independently created and each source is defined as a set of views over the global schema.

model definition repository (MDR)

AutoMed repository for holding the HDM specifications of the constructs of each high-level modelling language.

named entity recognition

IE task to identify proper names in text and to annotate the matching text.

natural language processing (NLP)

Field of computer science where knowledge of language is used to automatically process text.

partially structured data

Information consisting partly of some structured data conforming to a schema and partly of information left as free text.

pronominal coreference

Coreference detection for pronouns such as “I”, “me”, “my” and “yourself”.

proper names coreference

Coreference detection for proper names such as “IBM” or “Big Blue” (also known as orthographic coreference).

resource description framework (RDF)

World Wide Web Consortium specification for a metadata model, and component of the Semantic Web.

resource description framework schema (RDFS)

Type system for RDF which can be used to define a domain in terms of a set of specific classes and properties.

sentence splitter

IE component which divides text into sentences.

schema matching

Data integration task — given a set of data sources, identify correspondences between pairs of elements in their schemas, as a prerequisite to defining mappings.

schemas & transformations repository (STR)

Repository in AutoMed storing the definitions of source, intermediate and integrated schemas, and the pathways between them.

semantic web

Tim Berners-Lee's vision of extending the world wide web content to be able to be processed by computers as well as humans.

template

Concepts in ESTEST's global schema which have attributes are used to create templates to be filled in by the IE process.

text mining

Uses information retrieval or summarisation to transform a text corpus into a structured dataset for further data mining.

tokeniser

IE component which splits text into tokens, such as strings or punctuation. A specific tokeniser is required for a language such as English, for example.