Chapter 15
Information Extraction and Semantic Annotation for Multi-Paradigm Information Management

Hamish Cunningham, Valentin Tablan, Ian Roberts, Mark A. Greenwood, and Niraj Aswani

Why 'multiparadigm'? Why 'information management', instead of the more familiar 'information retrieval' or 'search'? Is our terminology, as suggested by one of our reviewers, 'PR drivel'?! At one level, perhaps, our title does indeed reflect the increasing penetration of short-termist market-oriented motivations into science and engineering (itself a part of the wider subjection of all areas of public life to corporate profit—see for example [3]). The work reported here was funded in part by a company with a close eye on its commercial potential, and we were concerned to describe that potential in our publications. There are also, however, two substantive points which we wanted to make, which are worth explaining in a little more detail. First, we believe that Mímir is distinctive in combining three types of indexing under one roof—hence *multiparadigm*—full text/boolean; conceptual/semantic; annotation (graph) structure. It is not the case that this combination is either commonplace or straightforward, and we are hopeful that the work will be influential as a result. Second, the technology suite into which Mímir fits is not just about indexing (or information retrieval as commonly defined)—hence *information management*—including as it does both GATE Teamware, a workflow-based distributed collaborative manual annotation and annotation management platform, and OWLIM, a semantic repository which is increasingly used for curated data sets as well as indices (examples in the UK include the BBC, the Press Association and the National Archive). We feel, therefore, that Mímir is an appropriate name for this enterprise.

H. Cunningham (✉) · V. Tablan · I. Roberts · M.A. Greenwood · N. Aswani
Department of Computer Science, University of Sheffield, Sheffield, UK
e-mail: H.Cunningham@dcs.shef.ac.uk

V. Tablan
e-mail: V.Tablan@dcs.shef.ac.uk

I. Roberts
e-mail: I.Roberts@dcs.shef.ac.uk

M.A. Greenwood
e-mail: M.Greenwood@dcs.shef.ac.uk

N. Aswani
e-mail: N.Aswani@dcs.shef.ac.uk

Abstract This chapter describes the development of GATE Mímir, a new tool for indexing documents according to multiple paradigms: full text, conceptual model, and annotation structures. We also present a usage example for patent searchers covering measurements and high-level structural information which was automatically extracted from a large patent corpus.

15.1 Introduction

Dear Reader,

The gentle tinkling noise that you can hear in the background is the sound of genre expectations shattering. This is not an *'intellectual property'* paper (indeed I[1] am uncertain that such a thing really exists—how could *intellect* be *property*?). As Glyn Moody[2] points out, Turing's results imply the atomicity of the digital revolution, and consequently it seems likely that the electronic genie is now so far out of the 'rights' bottle that all bit stream representations of our human achievements will follow into the realm of openness and cooperative enterprise sooner or later. Nor is this an *information retrieval* paper, at least not in the well-behaved sense of positing a hypothesis about model performance within a particular set of parameters and then testing and drawing some familiar variant of an ROC curve to show how well our hypothesis applies in the context of some particular data set.

We will break the expectations of patent searchers by paying little attention to the particular needs of that community (although the work we report was initially applied to patents and is likely to have benefits there), and perform similar violence to the expectations of IR researchers by making a fairly rudimentary evaluation. Now that only the curious are still reading, I can appeal to you as a kindred spirit.[3] Join me in a short history of some technological developments that my colleagues and I have had the pleasure of making over the last few years. I promise not to tell anyone about your paradigm-shifting deviance if you'll extend the same courtesy to me.

The paper covers work on two areas. First, the integration of standard information retrieval techniques with semantic annotation and information extraction work in order to deliver search capabilities that may be more flexible and interactive than previously. Second, on scalability via distributed processing and efficient indexing. It is structured as follows:

- we begin in Sect. 15.2 with some context and terminology relating to both the characteristics of patent searching and the text mining technology from which Mímir has developed
- in Sect. 15.3 we present the design and implementation of Mímir

[1] In the interests of protecting the innocent the first author lays claim to the introduction.

[2] http://opendotdotdot.blogspot.com/.

[3] I used to hope that as time passed I would become older and wiser, but it seems that in fact I just become odder and wider.

- Sections 15.4 and 15.5 go on to describe the semantic annotation of patent documents with general categories such as bibliographic references or sectioning and specific data on measurements that appear in the texts
- Section 15.6 gives an extended example of the type of multi-paradigm search process that is possible as a result of pushing the annotations described in 15.4 and 15.5 into a Mímir index server
- Section 15.7 wraps up with discussion of the main achievements described within this chapter

15.2 Background

15.2.1 Semantic Annotation

Semantic annotation is the process of attaching metadata tags and/or ontology classes to text segments, as an enabler for knowledge access and retrieval tools. Automatic annotation is carried out by employing Information Extraction (IE) [4] techniques, which recognise automatically instances of a given set of events, entities or relationships. From an algorithmic perspective, IE approaches fall in to two broad categories: manually engineered ones (frequently based on pattern-matching like rules, see e.g. [13]) and machine learning ones (see e.g. [2, 12]). Rule-based approaches are more suitable where a carefully engineered, high precision system is needed and there are not sufficient training data for a machine learning approach to be successful. From an operational perspective, IE tools can be deployed in both fully and semi-automatic applications (where users can inspect and, if needed, correct the automatically created metadata). In general, fully automatic methods are preferred when the volume of data is too large to make human post-annotation practicable, as is the case with patents.

15.2.2 Patents

Patents are an important vehicle for the expression of corporate strife, and this importance is increasing in the current intensification of international competition. When researching new product ideas or filing new patents, inventors and patent attorneys need to retrieve all relevant pre-existing know-how and/or exploit and enforce patents in their technological domain. This process may be hindered, however, by a lack of rich metadata, which if present, would allow powerful concept-based searches to complement the traditional keyword-based approaches.

Patent searchers require high recall methods, capable of operating robustly on large volumes of data. Much early IE research was carried out on smaller datasets from narrower domains, often news articles [2, 9, 14]. A challenge addressed more recently is in scaling up these methods to deal with the diversity and volume of patent data.

Applications of IE to patent annotations are quite scarce, mostly focusing on optical character recognition (OCR) and text classification, whilst only briefly discussing the importance and challenges of identifying references to figures and claims in patents. In this area, [11] carried out a small feasibility study using the Xerox language processing tools. The PatExpert project [15] has developed some content extraction components based upon deeper linguistic analysis than the approach proposed here.

15.2.3 ANNIE and ANNIC

In 2007 we began work on adapting an IE and semantic annotation system to patent data. This system (ANNIE, A Nearly-New IE pipeline) is part of GATE (http://gate.ac.uk/ [6, 7]), which also includes a diverse set of development tools for language processing R&D. One such tool is ANNIC (ANNotations In Context), which predates the work described here [1]. ANNIC was designed to support the development of finite state transduction patterns in GATE's JAPE language.[4] ANNIC is used to search corpora that have been both annotated with GATE and indexed using Lucene.[5] Users make searches based in a query language very similar to JAPE and are presented with a results summary similar in form to KWIC (Key-Words In Context) tools: the portions of text that match the query form a column down the centre of the screen and are preceded and followed by the proximate text on either side.

ANNIC was designed as a development tool, not as an end-user tool, and is tightly integrated within GATE Developer (a specialist tool for R&D workers), and is inefficient beyond the range of a few hundred documents. We had no intention of proposing the tool as appropriate for patent searchers, but by chance we used it to demonstrate some of the IE work to a patent search expert group. The feedback from this group was very positive, and we were commissioned to produce a version of ANNIC that would scale to a one terabyte plain text database of patent documents—hence Mímir, to whose design we now turn.

15.3 GATE Mímir—A Multiparadigm Index

Mímir[6] is a multi-paradigm information management index and repository which can be used to index and search over text, annotations, semantic schemas (ontologies), and semantic metadata (instance data). It allows queries that arbitrarily mix full-text, structural, linguistic and semantic queries, and that can scale to gigabytes of text.

[4]JAPE is a regular expression based language for matching annotations—see http://gate.ac.uk/userguide/chap:jape.

[5]http://lucene.apache.org/java/.

[6]Old Norse "*The rememberer, the wise one*".

15.3.1 What Is in a Mímir Index?

A typical semantic annotation project deals with large quantities of data of differing kinds. Mímir provides a framework for implementing indexing and search functionality across all these data types. The data types currently supported within a Mímir index are listed below in the order of increasing information density.

Text All documents have a textual content.[7] Support for full text search represents the most basic indexing functionality and it is required in most (if not all) cases. Even when semantic annotation is used to abstract away from the actual textual data, the original content still needs to be accessible so that it can be used to provide textual query fragments in the case of more complex conceptual queries.

Mímir uses inverted indexes[8] for indexing the document content (including additional linguistic information, such as part-of-speech or morphological roots), and for associating instances of annotations with the positions in the input text where they occur. The inverted index implementation used by Mímir is based on MG4J.[9]

Annotations The first step in abstracting away from plain text document content is the production of *annotations*. Annotations are metadata associated with text snippets in the documents. Typically an annotation is described by:

- the document it belongs to;
- the start and end offsets of the referred text snippet;
- the annotation type (a textual label or an URI);
- an arbitrary set of <feature, value> pairs.

An annotation index supports a more generic search paradigm. Depending upon the type of annotations available, the user can search across different dimensions. For example, if we suppose that all words in the indexed documents are annotated according to their part of speech, then one could search for sequences of type `{Determiner}{Adjective}{Noun}`, which would match phrases like *The red car* or *The new method*, etc. When the annotations are semantically richer, this new search paradigm gains more representational power. If, for example, the documents are annotated with occurrences of `Person`, `Location`, `Organization` entities, then searches like `{Person}, CEO of {Organization}, based in {Location}` become possible.

[7]Although the focus is currently on indexing text documents, specifically patents, it would be perfectly feasible to associate annotations and KB data with multimedia documents, where offsets may refer to time spans in videos or areas of an image etc.

[8]*Inverted Indexes* are data structures traditionally used in Information Retrieval to support indexing of text.

[9]http://mg4j.dsi.unimi.it/.

Knowledge Base Data Knowledge Base (KB) data consist of an ontology populated with instances. The ontology represents the data schema and comprises a hierarchy of class types, and a hierarchy of properties that are applicable between instances of classes. The instance data represent facts that are known to the system and are typically, or at least partially, derived from the semantic annotation of documents. KB data are used to reach a higher level of abstraction over the information in the documents and enables conceptual queries such as measurement ranges. A KB is required for answering such queries as they may often involve converting from one measurement unit into another, and reasoning about scalar values.

A KB that is pre-populated with appropriate world knowledge can perform other generalisations that are natural to humans users, such as being able to identify Vienna as a valid answer to queries relating to Austria, Europe or the Northern Hemisphere.

Mímir uses a KB to store some of the information relating to annotations. The links between annotations, the textual data, and the KB information are created by the inclusion into the text indexes of a set of specially-created URIs that are associated with the annotation data. Furthermore, URIs of entities from the KB can be stored as annotation features.

KBs are typically represented as a collection of triples that are kept in highly-specialised and optimised triple stores; using standards such as RDF or one of the versions of OWL.[10] The implementation used by Mímir is based on ORDI and OWLIM.[11]

15.3.2 Searching Mímir Indexes

From a user's point of view, Mímir is a tool for searching a collection of semantically annotated documents. It provides facilities for searching over different views of the document text, for example one can search the document's words, the part-of-speech of those words, or their morphological roots. As well as searching the document text, Mímir also supports searches over the documents' semantic annotations; where queries are based on annotation types and restrictions over the values of annotation features. These different search paradigms can be combined freely into complex queries, with support for sequences, repetitions, and Boolean operators.

A search session entails the formulation of a query, running the query with the Mímir query engine, and then consuming the results.

There are two different methods for constructing Mímir queries:

Query Language: A simple language has been defined that allows the formulation of Mímir queries using plain text.

[10] See http://www.w3.org/RDF/ and http://www.w3.org/TR/owl-features/.

[11] See http://www.ontotext.com/ordi/ and http://www.ontotext.com/owlim/.

Java API: The Mímir Java API defines a set of classes that represent query nodes. Each class corresponds to a type of query that Mímir supports. Individual nodes, representing sub-queries, are combined to form a query tree which embodies a more complex query. The node for the root of the query tree can then be used to execute the query through the Mímir query engine. This format is always used internally by Mimir to represent queries; queries sent in textual form (using the query language) are first converted to a tree of query nodes, and then executed.

There are three different methods for searching with Mímir:

Web Interface: When run as a web application, Mímir exposes a GWT- (Google Web Toolkit) based web interface that can be used from any browser. This is the simplest (and most user-friendly) way to access the search functionality of Mímir.

Java API: When Mímir is embedded into another Java application the Mímir search API can be used to construct queries, execute them, and process the results.

Web Service: When Mímir is run as a web application, a RESTful web service is published that allows the formulation of queries (using the query language), the execution of queries, and the retrieval of results.

Whilst this plethora of query building and search facilities makes Mímir extremely flexible it is unlikely that most patent searchers will need to venture further than entering queries into the web interface (or some other user interface built on top of one of the other search APIs). Given this reasoning, the rest of this section will focus on constructing queries using the plain text query language. For the adventurous, full details of the Java API and Web Service interface can be found in [10].

15.3.2.1 Constructing a Query

Mímir queries consist of one or more sub-queries linked by operators. The rest of this sections details the different query types and the operators that can be used to combine them to form more complex queries.

String Queries: The simplest form of query is a query term. This will match all occurrences of the query term in the indexed documents.
If the Mímir index being interrogated includes multiple string indexes, then the particular index to be searched can be specified by prefixing the query term with the index name and a colon, for example the query '*root:be*'[12] will match all morphological forms of the verb *to be*. If the name of the string index is omitted, then the first configured index is used. By convention (reflected in the default Mímir configuration) the first string index is used to store the terms text, so the default behaviour is to search over the document text, as expected.

[12]This assumes that an index named `root` exists, and was used to store the morphological root of the words.

Table 15.1 Escaping reserved constructs in the Mímir query language

Reserved input	Escaped form
{, }	\\{, \\}
(,)	\\(, \\)
[,]	\\[, \\]
:	\\:
+	\\+
|	\\\|
&	\\&
?	\\?
\\	\\\\
.	\\.
″	\\″
=	\\=
IN	“IN”
OVER	“OVER”
OR	“OR”
AND	“AND”

Some words are part of the query language definition so they cannot be used directly as query terms. If that is desired, then these constructs must be escaped as shown in Table 15.1.

Annotations Queries: If annotations were indexed then Mímir allows searching for annotation-based patterns. An annotation is a piece of metadata associated with a text segment. When indexed in Mímir, annotations are defined by:

- *type*: a string value
- *start and end offsets*: two numeric values that link the annotation with the text segment they refer to
- *features*: a set of named values. Each indexed feature must have one of the following types:
  - *nominal*: when the permitted values are strings from a limited set
  - *numeric*: floating-point numbers representable in double precision
  - *text*: arbitrary string values
  - *URI*: URIs are used to create links to resources (such as classes or entities) in semantic knowledge bases

When searching for annotations, the user needs to describe their request by providing an annotation *type* and, optionally, one or more *feature constraints*. An annotation query takes the following form: `{Type feature1=value1 feature2=value2 ...}`.

While the example above uses equality for the feature constraints, other operators are also available. Here is the full list:

Equality: Represented by the sign = matches annotations which have the given value for the specified feature. The equality operator is applicable to features of any type.

Comparison Operators: Represented by one of the following symbols: <, <=, >, >=, with the usual meaning. These operators can apply to features of type `nominal`, `numeric`, or `text`.

Regular Expressions: Can be specified using the syntax `REGEX(pattern, flags)`, where the `pattern` represents the regular expression sought, and the `flags` are optional, and can be used to change the way matching is performed. See http://www.w3.org/TR/xpath-functions/#regex-syntax for a full specification of the regular expression support. The `REGEX` operator can only be used for `nominal`, and `text` features.

Some example annotation queries are:
`{PatentDocument date > 20070000}`this searches for all patent documents published from 2007 onwards.[13]
`{Reference type = figure}`—retrieves all references to figures within the index.

Sequence Queries and Gaps: As sequence is the default operator in Mímir, there is no graphical sign for it: simply writing a set of queries one after another will cause a search for sequences of hits, one from each sub-query. For example, the query "`the energy level`" is actually a sequence query where the first sub-query searches for the word "*the*", the second for "*energy*", and the last for "*level*". This would match occurrences of the exact phrase '*the energy level*' in the indexed documents. Note that this is different from the standard behaviour of search engines, the majority of which would simply match documents in which all three query terms occur, in whichever order. This type of searching is also supported in Mímir, through the `AND` operator which is discussed later in this section.

It is sometimes useful to include gaps in a sequence query, that is, to allow arbitrary text fragments (of specified length) to occur in-between the hits from some of the sub-queries. This can be done by using the gap markers "`[n]`", or "`[m..n]`". These will match a sequence of length n, or with a length of between m and n of arbitrary tokens.

For example the query "`the [2] root:time`" will match phrases like "*the best of times*" or "*the worst of times*", whereas the query "`the [2..10] root:time`" would also match "*the best use of one's time*" (where the gap consists of six tokens—five words and an apostrophe).

`AND` Operator: The '`AND`' (also '`&`') operator can be used to specify queries that should match document segments that include at least one hit from each of the sub-queries. The results returned will always be the shortest document segments that satisfy the query.

`OR` Operator: `OR` queries are used to search hits that match one of a set of alternative query expressions. This is indicated by using the `'OR'` (also'`|`') operator between

[13]In general dates are encoded as yyyymmdd. This encoding allows dates to be treated as numbers, enabling a wide variety of search restrictions.

the sub-queries. A query of the form `Query1 | Query2` will return hits that match either sub-query `Query1` or sub-query `Query2`.

`IN` and `OVER` Operators: The operators `IN` and `OVER` are used to search for hits of a query that contain, or are contained in the hits of another query. For example:

Query1 IN Query2 will match all the hits of `Query1` that are contained in a hit of `Query2`.

Query1 OVER Query2 will match all hits of `Query1` that contain (are overlapping) a hit of `Query2`.

Repetition Operator: The + operator can be used to match text segments that comprise a sequence of hits from the same sub-query. The length of the sequence is specified through a number (representing the *maximum* number of repetitions) or through two numeric values (representing the *minimum* and *maximum* number of repetitions). For example:

"`to+3`" will match one, two, or three repeated occurrences of the word *to*. The returned hits will be of the form "*to*", "*to to*", or "*to to to*").

"`{Measurement}+2..5`" will match sequences of two, three, four, or five adjacent `Measurement` annotations.

Grouping: In the case of complex queries that include multiple sub-queries, parentheses '`(`', '`)`' can be used to group a set of sub-clauses together.

15.4 The Patent Annotation Task

The experiments in this paper are based upon three different kinds of patents taken from the MAREC collection[14]: American (USPTO), Japanese (JP) and European (EPO). The reason for choosing multiple data sources is because the three patent types differ in terms of the metadata, formatting, quality, and legal language used. These differences ensure that the approaches we develop can be applied to a wide range of documents, and hopefully to unseen document types with little loss in performance.

The semantic annotation process adds new metadata to the patents (in the form of XML tags). These new metadata fall into two broad categories; wide and deep annotation types. Wide annotations are intended to cover metadata types that apply to patents in general, and do not depend on the specific subject area of the patent (as identified, for example, by its IPC code). Examples of such metadata include document sections and references to cited literature, examples, figures, claims, and other patents. Deep annotations are specific to one or more subject areas and are of interest to specialised patent searchers. The experiments reported here focus upon automatic annotation of measurements (as they are very important for patent professionals) whilst also being very hard to find using keyword search. This is due to the diverse ways in which they can be expressed via natural language.

[14]http://ir-facility.net/prototypes/marec/.

The benefits from the automatic metadata enrichment process are three-fold. Firstly, information extraction (IE) is capable of dealing with variable language patterns and format irregularities much better than text-based regular expressions. For example, references to other patents can be very diverse: U.S. Patent 4,524,128, Korean laid open utility model application No. 1999-007692. Secondly, once the additional metadata have been added to the patent, IE tools can also carry out data normalisation. Again, taking an example from references to figures or similarly claims, expressions such as "Figures 1–3" or "Claims 5–10" imply references not just to the explicitly mentioned figure/claim numbers but also to all those in between. Lastly, by using text mining techniques we are able to extract a significantly wider range of useful information, than could be obtained via keyword search, and provide it as additional XML tags in the patent documents.

The rest of this section details the metadata we currently extract from patents and highlights some of the problems and how these have been overcome.

15.4.1 Section Annotations

Patent documents are typically quite long, contain multiple required sections, and use highly formalised legal and technical terminology (with the notable exception of literature references and measurements). Different aspects of the patent application are typically presented in a pre-defined set of sections and subsections (e.g. prior art, patent claims, technical problem addressed and effect). Both USPTO and EPO documents have at least three main parts, *the first page* containing bibliographical data and abstract, *the descriptions part*, and *the claims part*.

Automatic section recognition is based upon identifying typical section titles and using them to automatically partition the text. Pre-existing section markup is used, if available. For instance, Bibliographic Data, Abstract and Claims sections tend to be already annotated in patent documents so we use them directly. There are, however, around 20 different sections within most patents[15] and so most sections still need to be detected automatically.

15.4.2 Reference Annotations

Reference annotations are used for parts of text that refer to either objects in the current document (e.g. figures, tables, etc.) or to other documents (e.g. scientific papers).

A reference annotation consists of two parts; a header indicating the type of reference, and one or more identifiers which typically consist of a mixture of numbers

[15]The number of sections within a patent can vary widely from one patent office to another and even, over time, within the same office. Most of the patents we examined during the reported work do, however, contain around twenty sections.

and letters. For example, in *Figure 1 and 2* the header is *Figure* and the identifiers are *1* and *2*. In *U.S. Pat. No. 3,765,999* the header is *U.S. Pat.* and the identifier is *No. 3,765,999*.

Conjunctive phrases mentioning references to two or more objects of the same reference type are tagged initially as one reference annotation, including the conjunction and all punctuation. For example, *Figures 1 and 2*; *Claims 1–3*; *Tables 1 to 10* are first annotated as one Reference each, of type Figure, Claim and Table respectively. The normalisation step then separates these into their constituent references; including all implied references (e.g., to Claim 2).

From an IE perspective, some types of references are much simpler to identify than others. For instance, there is little variability in the way patents refer to figures, tables, claims, equations, and examples. References to other patents tend to be slightly more challenging, as they often include the inventors' names, patent date, or even title—in addition to a simple header and identifier. The hardest of all are the references to external sources, such as published papers (see e.g., Hudson & Hay, Practical Immunology (Blackwell Scientific Publications, Oxford, UK, 1980), Chap. 8), which tend to be quite long and typically contain many abbreviations and idiosyncratic formatting. We have also observed significant differences between American and European patents in this respect and had to adapt our IE tools to deal with this accordingly.

15.4.3 Measurement Annotations

Most measurements comprise a scalar value followed by a unit, e.g. 2×10^{-7} Torr. Furthermore, two scalar values with or without a unit can be contained in an interval. Sometimes there are also accompanying words, such as "less than" or "between" which are important for professional searchers and, therefore, need also to be marked by the IE tools, e.g., "less than about 0.0015 mm", "2×10^5 to 2×10^7 cpm/ml". Lastly, we also deal with relative measurements, such as percentages and ratios.

The main challenge involved in recognising measurements in patents comes from the large number of measurement units in existence (e.g., units used in physics patents are very different to those used in engineering ones). Another challenge is that some units have single letter abbreviations. These can introduce ambiguities and therefore require a wider context to be considered in order to determine whether a specific sequence of numbers followed by a letter is indeed a measurement. One frequently encountered example of such ambiguities are temperatures, e.g., "1C" where we need to distinguish correct temperature mentions from other cases, such as references to figures, examples, tables, etc. (as in "see Figure 1C").

Table 15.2 The SAMIE components listed in runtime order (items in **bold** were developed specifically for SAMIE, other components were customised as needed)

Processing resource	Description
Cleanup	Remove annotations from previous application runs
Import Relevant Markup	Makes relevant markup from the original document available to the rest of the pipeline
Roman Numerals	Annotates Roman numerals which are used for detecting references
Numbers in Words	Recognises numbers written as words and converts them to actual values
Tokeniser	Pattern matcher for detection of words and other lexical items
Sentence splitter	Regular expression-based detection of sentence boundaries
POS tagger	Addition of part of speech (grammatical categories) to tokens
Gazetteer (case sensitive)	Lookup of known domain terms
Gazetteer (case insensitive)	Lookup of known domain terms, with case insensitive matching
Numbers	Find and annotate all remaining numbers
References Transducer	Find and annotate all the references within the documents
Measurement Tagger	Find and annotate all the measurements within the documents

15.5 Automatic Patent Annotation

Our approach to the large scale semantic annotation of patent documents is embodied in an information extraction system called SAMIE. This section discusses both SAMIE and the processing infrastructure we have developed to support large scale IE tasks.

15.5.1 SAMIE Architecture

SAMIE is provided as a GATE Application Pipeline consisting of a number of independent modules. The modules which make up the application are shown in runtime order in Table 15.2. The pipeline works as follows:

NLP Infrastructure: A basic set of NLP components were used to perform a shallow analysis of the input documents; adding simple linguistic features, such as part-of-speech, to the document. These features are added as annotations on the document.

JAPE Grammars: Numbers in the documents were mostly identified using the JAPE pattern matching language [5]. Every number which was recognised was augmented by the addition of a 'value' feature holding a double representation of the number. JAPE grammars were also employed to detect and annotate sections and references as described in Sects. 15.4.1 and 15.4.2.

Measurement Tagger: The measurement tagger is a complicated mix of JAPE rules and Java code that can recognise valid combinations of known units and reduce the units to a form in which they consist only of SI units. This reduction to SI units then allows measurements of the same dimension (i.e. length) expressed in different ways (e.g. metres, inches, feet ...) to be indexed, compared against each other and retrieved no matter the unit expressed by the user. The measurement tagger relies on the previous number annotations to remove spurious matches (i.e. a measurement *nearly* always starts with a number).

To enable complex measurement-based queries we have extended the Mímir query language so that `Measurement` annotations support a special synthetic feature, named `spec` which can be used to specify in natural language a measurement value, or a range of values to search for

The values used by the `spec` feature can take one of two forms:

number unit: This will match scalar measurements that have the exact specified value,[16] and interval measurements that contain the specified value. For example, '23 cm' or '3 inches'.

number to number unit: This will match scalar measurements that fall within the specified interval, and interval measurement that overlap with the specified range. For example, '2.5 to 15 amperes'. would match all of the following values: '3000 mA', '0 to 5 A', '7 to 100 Amperes', etc.

In either case unit normalisation is performed, so a query expressed in metres can match annotations expressed in inches, or millimetres, etc. For example, all the following represent the same query:

```
{Measurement spec = "3 to 5 metres"}
{Measurement spec = "300 to 500 cm"}
{Measurement spec = "3000 to 5000 mm"}
{Measurement spec = "118 to 197 inches"}
```
[17]

An evaluation of SAMIE [10] has found that the accuracy of the annotations detailed in this sections is comparable to that of human annotators tasked with producing the same metadata. This evaluation gives us the confidence to apply SAMIE to the task of large scale automatic annotation of patents.

15.5.2 Large Scale Annotation with GATE Cloud

One of the main challenges faced in this project is the sheer scale of the task. Patent databases typically contain tens of millions of patents, and hundreds of thousands of new ones are produced each year. Worldwide, millions of new patent applications are submitted yearly.[18] Any application aimed at the IP domain requires a good scalability profile if it is to maintain any credibility.

[16]Within the precision allowed by floating-point arithmetic of double precision.

[17]This query is approximately equal to the others as the two values have been rounded to the nearest whole numbers.

[18]Detailed statistics are available from the World Intellectual Property Organization at http://www.wipo.int/ipstats/.

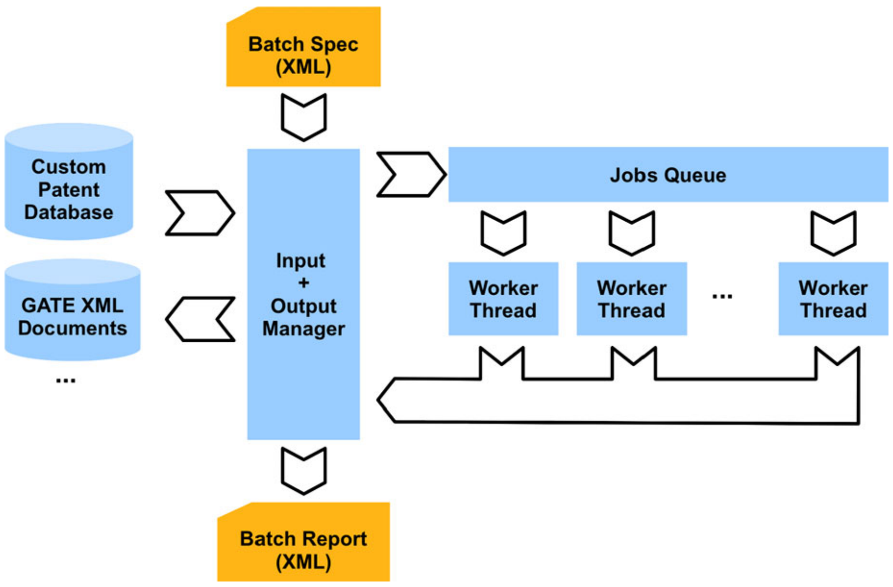

Fig. 15.1 Overall architecture of GATE Cloud platform

To answer our need for scalability, we have developed GATE Cloud[19]—a platform for parallel semantic annotation of text documents. GATE Cloud is designed as a parallel version of the execution engine found in GATE [7]. It takes a language processing pipeline created using the GATE Developer environment (in this case, the SAMIE application detailed in the previous section), and executes it using a set of parallel threads. The job control is effected through document batches, which are XML files describing outstanding tasks.

A high-level view of the architecture of GATE Cloud is presented in Fig. 15.1. The main elements in the diagram are detailed below:

Batch Spec: A batch is a unit of work that needs to be performed. It comprises a list of IDs for the documents that need to be processed, a pointer to the prototype of the processing pipeline that should be used, and configuration data specifying input/output options.

Input Output Manager: The I/O manager reads the batch files, parses them, and extracts the IDs for the documents that need to be processed. Its main role is to handle the import/export operations for the patent documents. Internally, GATE Cloud uses GATE Document objects as defined by the GATE Java API; the I/O Manager's job is to create the initial GATE document object for each new document, and to handle the saving of the results at the end of the process. This is also where the integration with various document stores (such as on-disk GATE datastores, or custom patent databases) is handled.

[19]http://gatecloud.net.

Jobs Queue: Each document to be processed represents a job. These are represented as document IDs and are stored in the jobs queue. The queue is accessed in parallel by all the execution threads whenever they become available for work.

Worker Threads: A worker thread is a copy of the processing pipeline that manages its own execution thread. Its execution comprises a loop in which it gets the ID for the next available document, it reads the document through the I/O Manager, it executes the processing pipeline over the document, and, finally, it exports the results, again using the facilities provided by he I/O Manager. The number of parallel worker threads is a configuration option for each instance of GATE Cloud, and it depends on the hardware characteristics of the host.

Batch Report: The execution of each batch is reflected in a batch report file in XML format. This includes, for each document, whether the execution was successful or some error occurred, and some simple statistics regarding the number of annotations of each type that were produced. Furthermore, once the batch execution completes, details are included regarding the total number of document processed, how many encountered errors, and the total execution time for the whole batch.

In order to determine the most suitable hardware configuration for running GATE Cloud, we have performed a series of experiments. The main parameters we were trying to estimate were memory requirements, and CPU load, i.e. how many worker threads should be allocated given the number of available CPU cores. Finding the optimal memory allocation is important because low values lead to excessive amounts of CPU time being used for garbage collection, while large values are wasteful. The number of worker threads for a given CPU configuration also needs to be optimised to increase CPU utilisation, while avoiding excessive context switching and locking due to access to shared resources (such as the disk, or network interfaces).

The optimal values will vary depending on the type of documents being processed, and the requirements of the actual processing pipeline used. For each new deployment of GATE Cloud, these parameters should be estimated experimentally. In our particular case, the highest throughput was obtained when each worker thread had 2 GB of RAM allocated, and the number of threads was 1.5 times the number of CPU cores. In this configuration, the *execution speed* was over 1000 documents per hour and per CPU core.

GATE Cloud is designed for parallel execution and it aims at 100% utilisation of a multi-core and/or multi-CPU computer. When combined with an engine for distributed execution of jobs,[20] GATE Cloud can be deployed on large computer farms, or commodity compute clouds. This results in a highly scalable solution for semantic annotation of documents.

GATE Cloud is also intended to run for extended periods of time; conceivably it could even be deployed as a continuously running process. This places some stringent requirements with regard to the robustness of the process, which have influenced the design and implementation. Any errors and exceptions that may occur

[20]Such as the Sun Grid Engine (http://gridengine.sunsource.net/.) or Hadoop (http://hadoop.apache.org/).

during processing are trapped and reported, without crashing the entire process. If the GATE Cloud process does crash, for whatever reason (e.g. hardware failure, or power cut), the process can be restarted using exactly the same mechanism as was used to launch it originally. GATE Cloud will automatically identify the previous incomplete run, will parse the partial execution report file to find which documents were already processed successfully, and will resume execution from the point where the previous run stopped.

15.6 Multi-Paradigm Patent Search

Whilst some may find the technical details of Mímir interesting, most patent searchers simply wish to know if it will help them to do their job. This section aims to answer that question by showing an example search session over a corpus of 100,000 patents.

The scenario for this example search session involves finding inventions, and their inventors, that make use of transistors.

As with any search engine a good place to start is a keyword-based search.

```
transistor
```

This returns 75,208 hits in the example index—that is the word ‘transistor’ appears 75,208 times within the 100,000 patents. The main problem with this query is that because it matches words, rather than sequences of characters, it does not include any mention of the word ‘transistors’. We could rectify this in one of two ways. In this case, where there are only two variations of the word, we could issue the query `transistor OR transistors`. The problem with this query is that when you have words with more variations, or multiple words where you need to match different tenses, the queries can quickly become unwieldy. A better approach is to use one of the other Mímir string indexes to search on the root form of the word.

```
root:transistor
```

This query now returns 99,742 results. This is a lot of results to search through, and it is likely that most of the results refer to inventions in which transistors only play a minor role. One way to refine the search would be to concentrate on those results which occur within an abstract as this is suggestive of transistors playing an important role in the patent.

```
root:transistor IN {Abstract}
```

Our refined query now returns just 3,053 instances of ‘transistor’ or ‘transistors’ from within the index, but this does not equate to 3,053 different patents.

We can invert the previous query so, that instead of returning all mentions of transitors within abstracts, it instead returns the abstracts which mention transistors.

```
{Abstract} OVER root:transistor
```

The results show that there 1,088 such abstracts within the index. Given the way patents are written they often include multiple abstracts in different languages.[21] We can restrict our query to only focus on abstracts in a given language using the `lang` feature. For example we could focus on just the 369 abstracts written in French.

```
{Abstract lang=FR} OVER root:transistor
```

Given that we are using the English spelling of transistor (and the index was created using an English part-of-speech tagger) it would, however, make more sense to focus upon just those abstracts written in English.

```
{Abstract lang=EN} OVER root:transistor
```

This query returns 713 abstracts from the 100,000 patent index. Whilst this maybe a small enough number for a team of patent searchers to handle it is likely that refining the search further could be beneficial. One option would be to restrict the search based upon the date of a patent. For example, we could limit the search to only those patents published from 2007 onwards.[22]

```
  ({Abstract lang=EN} OVER root:transistor)
IN {PatentDocument date > 20070000}
```

This reduces the number of retrieved abstracts to just 321. We can also place an end date on the search in a similar fashion. Restricting the search to just those patents published during 2007 gives us the following query.

```
  ({Abstract lang=EN} OVER root:transistor)
IN {PatentDocument date > 20070000 date < 20080000}
```

This query retrieves 251 English language abstracts. Whilst this is a useful query (and a reasonable number of results to manually read through), it might be more helpful to start from the title of the inventions rather than the abstracts.[23]

```
  {InventionTitle lang=EN}
IN ({PatentDocument date > 20070000 date < 20080000}
OVER ({Abstract lang=EN} OVER root:transistor))
```

As a final example, maybe the aim of this whole search session was to find inventors that you could invite to join an expert pool focusing on transistor-based inventions. We can easily modify the query to retrieve the inventor's instead of the inventions.

```
  {Inventor}
IN ({PatentDocument date > 20070000 date < 20080000}
OVER ({Abstract lang=EN} OVER root:transistor))
```

The results from this query shows that there are 2,066 inventors related to the 251 inventions.

[21] Whilst this is true for the patents in the MAREC collection, which we used when building this example index, it may not be true for all patents. In fact the structure of patents varies widely which is one reason why effectively searching large patent corpora by hand is difficult.

[22] As previously mentioned, dates are usually encoded as numbers in the form yyyymmdd. As such 20070000 is not actually a valid day but does fall between the last day of 2006 and the first day of 2007.

[23] As with abstracts the titles of the inventions are also listed in multiple languages and so a restriction to English is included in the query.

The proceeding examples are of course just a small glimpse into the types of knowledge that can be easily discovered using Mímir. The index used for this example is publicly accessible[24] and we encourage interested readers to try Mímir for themselves.

15.7 Discussion and Conclusions

Quoting [8]:

> Information retrieval (IR) technology has proliferated in rough proportion to the expansion of knowledge and information as a central factor in economic success. How should end-users choose between them? Three main dimensions condition the choice:
>
> - Volume. The GYM big three web search engines (Google, Yahoo!, Microsoft) deliver sub-second responses to hundreds of millions of queries daily over hundreds of terabytes of data. At the other end of the scale desktop search systems can rely on substantial compute resources relative to a small data set.
> - Value. The retrieval of high-value content (typically within corporate intranets or behind pay-for-use turnstiles) is often mission-critical for the business that owns the content. For example the BBC allocates a skilled staff member for eight hours per broadcast hour to index their most important content.
> - Cost. Semantic indexing, conceptual search, linked data and so on share with controlled-vocabulary and metadata systems a higher cost of implementation and maintenance than systems based on counting keywords and hyperlinks.
>
> To process web-scale volumes GYM use a combination of one of the oldest and simplest retrieval data structures (an inverted file that relates search terms to documents) and a ranking algorithm whose most important component is derived from the link structure of the web. These techniques work much better than was initially expected, profiting from the vast number of human relevance decisions that are encapsulated in hyperlinks. Problems remain of course: first, there are still many data in which links are not present, and second the familiar problems of ambiguity (*index term synonymy* and *query term polysemy*) can lead to retrieval of irrelevant information and/or failure to retrieve relevant information.
> High-value (or low-volume) content retrieval systems address these problems with a variety of semantics-based approaches that attempt to perform conceptual indexing and logical querying. For example, the BBC system cited above indexes using a thesaurus of 100,000 terms that generalise over anticipated search terms. Similarly in the Life Sciences publication databases increasingly use rich terminological resources to support conceptual navigation (MeSH, the Gene Ontology, Snomed, the unified UMLS system, etc.).
> An important research theme in recent years has been to ask to what degree can we have our cake and eat it? In other words, how far can the low-volume/high-value methods be extended?

We believe that Mímir makes a contribution to this theme by demonstrating the possibility of scaling up annotation structure indices and combining annotation structure search with full text methods and with conceptual search based on RDF or OWL.[25]

[24]http://demos.gate.ac.uk/mimir/patents/gus/search.

[25]http://www.ontotext.com/owlim/.

When we began developing Mímir we assumed that we would be able to find an appropriate technology base in a related field such as XML indexing, or database management systems. To this end we convened a workshop in May 2008 on Persisting, Indexing and Querying Multi-Paradigm Text Models (at the IRF in Vienna).[26] A number of researchers working on IR, XML and DBMS were kind enough to participate.[27] It became clear at that point that there was no off-the-shelf solution to our problem (to cut a long story short XML-based techniques were too tree-oriented, and difficult to adapt to the graph structures in which we store annotation data, whereas DBMS techniques are similarly oriented on relational models). Luckily we identified a viable indexing mechanism in the form of MG4J from Sebastiano Vigna,[28] and this forms the core of annotation index management in Mímir.

In this paper we presented the results applied to a use case in patent processing. We also briefly introduced GATE Cloud, our approach to scalability for large annotation tasks. GATE Cloud allows us to take a GATE application and deploy it across machines in a cloud environment allowing the number of documents we can process to be limited only by the machine power available to us.

Together Mímir and GATE Cloud allow us to deliver applications that appear useful in a wide variety of multi-paradigm search contexts.

Acknowledgements This work was funded by the Information Retrieval Facility (ir-facility.org). Erik Graf helped us get off the blocks with MG4J and Sebastiano Vigna helped us run the extra mile. Thanks also to all the workshop participants listed above. We are grateful to our reviewers who made salient and constructive contributions.

References

1. Aswani N, Tablan V, Bontcheva K, Cunningham H (2005) Indexing and querying linguistic metadata and document content. In: Proceedings of fifth international conference on recent advances in natural language processing (RANLP2005), Borovets, Bulgaria
2. Bikel D, Schwartz R, Weischedel R (1999) An algorithm that learns what's in a name. Mach Learn, Special Issue on Natural Language Learning 34(1–3)
3. Chomsky N (1999) Profit over people: neoliberalism and global order, 1st edn. Seven Stories Press, New York
4. Cunningham H (2005) Information extraction, automatic. In: Encyclopedia of language and linguistics, 2nd edn, pp 665–677
5. Cunningham H, Maynard D, Tablan V (2000) JAPE: a Java annotation patterns engine, 2nd edn. Research Memorandum CS-00-10, Department of Computer Science, University of Sheffield, Nov 2000

[26] http://gate.ac.uk/sale/talks/sam/repositories-workshop-agenda.html.

[27] Gianni Amati (Fondazione Ugo Bordoni/University of Glasgow); Mike Baycroft (Fairview Research); Norbert Fuhr (University of Essen-Duisburg); Eric Graf (University of Glasgow); Atanas Kiryakov (Ontotext); Borislav Popov (Ontotext); Ralf Schenkel (MPG); John Tait (IRF); Arjen de Vries (ACM/CWI); Francisco Webber (Matrixware/IRF); Valentin Tablan (University of Sheffield); Kalina Bontcheva (University of Sheffield); Hamish Cunningham (University of Sheffield).

[28] http://mg4j.dsi.unimi.it/.

6. Cunningham H, Maynard D, Bontcheva K, Tablan V, Dimitrov M, Dowman M, Aswani N, Roberts I, Li Y, Funk A (2000) Developing language processing components with GATE Version 6.0 (a user guide). http://gate.ac.uk/
7. Cunningham H, Maynard D, Bontcheva K, Tablan V (2002) GATE: a framework and graphical development environment for robust NLP tools and applications. In: Proceedings of the 40th anniversary meeting of the association for computational linguistics (ACL'02)
8. Cunningham H, Hanbury A, Rüger S (2010) Scaling up high-value retrieval to medium-volume data. In: Cunningham H, Hanbury A, Rüger S (eds) Advances in multidisciplinary retrieval (the 1st information retrieval facility conference). LNCS, vol 6107. Vienna, Austria, May 2010. Springer, Berlin
9. Day D, Robinson P, Vilain M, Yeh A (1998) MITRE: description of the *Alembic* system used for MUC-7. In: Proceedings of the seventh message understanding conference (MUC-7)
10. Greenwood MA, Cunningham H, Aswani N, Roberts I, Tablan V (2010) GATE Mímir: philosophy, development, deployment and evaluation. Research Memorandum CS-10-05, Department of Computer Science, University of Sheffield
11. Hull D, Ait-Mokhatar S, Chuat M, Eisele A, Gaussier E, Grefenstette G, Isabelle P, Samuelsson C, Segond F (2001) Language technologies and patent search and classification. World Pat Inf 23:265–268
12. Li Y, Bontcheva K, Cunningham H (2005) SVM based learning system for information extraction. In: Winkler MNJ, Lawerence N (eds) Deterministic and statistical methods in machine learning. LNAI, vol. 3635. Springer, Berlin, pp 319–339
13. Maynard D, Tablan V, Ursu C, Cunningham H, Wilks Y (2001) Named entity recognition from diverse text types. In: Recent advances in natural language processing 2001 conference, Tzigov Chark, Bulgaria, pp 257–274
14. Maynard D, Bontcheva K, Cunningham H (2003) Towards a semantic extraction of named entities. In: Recent advances in natural language processing, Bulgaria
15. Wanner L, Baeza-Yates R, Brugmann S, Codina J, Diallo B, Escorsa E, Giereth M, Kompatsiaris Y, Papadopoulos S, Pianta E, Piella G, Puhlmann I, Rao G, Rotard M, Schoester P, Serafini L, Zervaki V (2008) Towards content-oriented patent document processing. World Pat Inf 30(1):21–33