Chapter 5

Web-Based Corpus Software

Saturnino Luz

5.1 Introduction

What is a web-based corpus and what is web-based corpus software? The answer is, strictly speaking, that there is no such thing as *web-based corpus software*. However, one should not be discouraged by this rather negative assessment. In fact, if one examines the title closely, different bracketings of the phrase might suggest interesting possibilities. For example, if one chooses to write it as '(web-based corpus) software', the emphasis falls on the idea of the World Wide Web as a large corpus. It is, however, a very chaotic one. It is chaotic in the sense that it is difficult for its users to account for and control the sort of phenomena such a large and dynamic repository might reflect when explored, say, through an ordinary search engine. This makes the task of formulating and testing hypotheses extremely difficult. All sorts of 'noise' might creep in: there are texts written by native and non-native writers, computer-generated text (e.g. text resulting from the ubiquitous web-page translation services currently on offer), duplication, and other forms of text which do not conform to standard norms. Little, if anything, can be done to guarantee the quality or integrity of the data being used. Still, this chaotic, noisy environment can be of some use to the statistically minded (computational) linguist. To borrow an example from Manning and Schütze (1999), one could use the web to decide which of the following word sequences to treat as a language unit: 'strong coffee' or 'powerful coffee'. A quick search reveals over 30,000 occurrences of 'strong coffee' against just over 400 occurrences of 'powerful coffee', thus indicating that the former forms a collocation pattern while the latter apparently does not.

In contrast, should one wish to write 'web-based corpus software' as 'web-based (corpus software)', the emphasis clearly falls on 'corpus software', of which web-based corpus software would simply be one type. In other words, one could simply regard the Web as the medium through which better constructed, human-designed corpora can be searched and studied by a large community of geographically dispersed researchers. Many tools have undoubtedly been designed which do just that, including the Translational English Corpus (TEC) system (Luz and Baker 2000). The main shortcoming of this approach stems from the very fact that it is better defined. One of the attractive aspects of the World Wide Web is that all stages of information exchange are distributed. That is, in principle, anyone can provide and access any data. Information consumers benefit from a simple and intuitive access model (hypertext) and the widespread availability of web-browsing software on virtually all platforms. More importantly, all an information consumer who wishes to become an information provider needs to do is learn a few idioms of a simple mark-up language (HTML). By using more specialized tools, such as corpus servers and clients, this flexibility is lost. And with flexibility goes the dream of a massive (and therefore statistically attractive), self-maintained corpus.

This chapter presents and discusses recent advances in tools, technologies and standards which might enable the corpus research community to bring about the basics of an infrastructure for creating and sharing distributed, dynamic and widely accessible corpora. The end result may still be a far cry from the dream of using the entire web as an evolving corpus, but it will certainly advance the idea that implementing a world-wide web of corpora is feasible, thus rendering moot the ambiguity in the title of this chapter. Since the above-mentioned infrastructure does not exist at present, we will necessarily have to focus on corpus software and tools that use the Web (or, more appropriately, the internet) as a communication medium. We start by presenting an overview of the technology involved, describe the model adopted by the TEC system to support remote access to a corpus of translated English, and finally discuss perspectives for future research in this area.

This overview of web technologies covers the main tools for data storage and mark-up, text indexing and retrieval, and the issue of distributed storage of corpora. It also addresses, though superficially, the issue of creating and maintaining metadata, including storage and database management. The aspects of text indexing and retrieval covered include tokenization, data structures for indices and search. Ways of moving from storage of an individual corpus to distributed storage of collections of corpora, and the non-technical issue this entails, namely, copyright, are also discussed. These issues are illustrated through a case study: the Translational English Corpus (TEC) system. We end the chapter by presenting a vision for web-based corpus software: its overall architecture and development philosophy.

5.2 The Internet and Corpus Software

The internet technologies presented and discussed below have not been developed specifically to deal with corpora, or even text, though text comprises the vast majority of data available on the internet. In what follows, we describe a selection of those technologies which have been found to be particularly useful for text storage and retrieval in the Translational English Corpus (TEC) project, namely: mark-up languages, indexing techniques and the client-server architecture. This represents a large array of technologies which are covered only partially and superficially, as the contribution aims to give the reader a perspective of the bigger picture and the possibilities it suggests for corpus research, rather than its details. References to the relevant literature are included in each section.

The TEC corpus is a collection of English texts translated from a wide variety of languages, both European and non-European, and is held at the University of Manchester (Baker 1999). It consists of four sub-corpora: fiction, biography, news and in-flight magazines. The corpus is an expanding collection which contains more than ten million indexed words, and 'headers' which store a variety of extralinguistic data about each text in the collection (known as *metadata*).

Although the individual texts that make up TEC are not directly available in full, researchers can access them over the internet, and run different types of analyses on the corpus or selected subsets of it through a set of software tools available from the TEC website. These tools support standard corpus analysis functionality, such as concordancing, as well as functionality specifically designed to allow translation researchers to explore different hypotheses by constraining search parameters derived from metadata. Possible search modes include selection by source language, by translator, by author, gender, nationality, and so on. For examples of how this functionality can be used and a discussion of its significance for translation studies see Baker (1999).

As a stochastic entity, the usefulness of a corpus is invariably dependent on its size and composition. Large amounts of text are needed if a corpus is to significantly reflect general language phenomena. For the sort of investigation carried out in corpus-based translation studies, where the corpus user is interested in discovering patterns inherent to a particular type of language, the constraints and mechanisms these patterns might reflect and so on, careful attention must be paid to text selection and documentation of extralinguistic features that might have a bearing on the sort of phenomena of which the corpus is supposed to provide samples. Together with legal constraints such as the need to protect copyright, corpus size and composition concerns dictate the main requirements for the software infrastructure: it must provide means for efficient physical storage and retrieval of data, and a loosely coupled, but secure and fine-grained database layout. The first step towards meeting these requirements is to supply the corpus designer with a simple and flexible way to annotate the data and encode metadata for further processing by other software modules.

5.3 A Brief Introduction to XML

A mark-up language is a collection of mechanisms that allow a corpus designer to annotate data with various forms of meta-information in a standardized and portable way. Probably the best known example of this kind of device is the hypertext mark-up language (HTML), which allows its users to specify text formatting and layout information. However, formatting mark-up languages such as HTML do not suffice for corpus annotation. In fact, different corpora designed by different people are likely to require different sets of tags, depending on the sort of metadata their user communities require. What is needed in addition to a standard syntax for annotation is a way of specifying the vocabulary (i.e. tags), syntax and constraints of the annotation scheme itself. The standard generalized mark-up language (SGML) was the product of the first comprehensive attempt to meet this need. SGML defines a set of simple, basic conventions (syntax) that should apply universally to all SGML-annotated documents,[1] and mechanisms for constraining this basic syntax so that it can be used in different contexts. Particular ways of constraining SGML syntax are known as SGML *applications*. HTML itself is an SGML application. Although the primary goal of SGML was standardization of document storage, its development was also greatly influenced by the goals of SGML application writers, namely: flexibility and user-friendliness. The result was a complex formalism described in over 150 pages of an ISO standard (ISO8879, 1986). That complexity ended up working against the primary goal of user-friendliness. The success of a data storage standard depends on it being well supported by software libraries and tools. In the case of SGML, library and application developers often chose to support only those parts of the standard they found essential or useful for certain applications, thus effectively undermining document portability.

The extensible mark-up language, XML, originated from an attempt to remedy this situation. Its designers started from SGML and simplified it by removing a number of exotic, rarely used features. XML is simple and flexible in that it has no pre-defined set of tags. It provides a uniform means of encoding metadata which is easily interpretable by humans as well as machines. It has been widely adopted by industry and academia, and its development has been coordinated by the World-wide Web Consortium (Bray et al. 2006). A large number of software libraries and tools exist which support all aspects of XML parsing and validation in most programming languages.

XML documents can be encoded in plain text, or in a variety of (language) encoding standards. The default encoding is UTF-8, a Unicode standard that supports virtually every character and ideograph from the world's languages. As with SGML, there is a basic (built-in) syntax which defines which documents (or text fragments) are *well formed*. This basic syntax can be constrained in a variety of ways, defining which documents are *valid*. Analogously to SGML, a specific way of constraining basic XML syntax yields what is called an *XML application*. From a parsing perspective, *well-formedness* and *validity* are the most important concepts in XML. XML also provides mechanisms for incorporating a form of semantics for XML documents with respect to XML applications. In the following sections, we examine each of these aspects in some detail.

5.3.1 Well-Formed XML

The basic syntax to which all XML documents must conform consists of a small set of simple rules governing essentially: the placement of tags, which strings are allowed as element names, and how attributes should be attached to elements. XML tags are text strings enclosed by angle brackets. Figure 5.1 shows two XML tags for author data: a begin tag and an end tag. This is a typical pattern in XML documents: annotated text appears between a begin tag and an end tag. Begin tags start with <. End tags start with </. In general, XML syntax is similar to HTML syntax. However, the following exceptions should be noted: in XML one is allowed to "invent" one's own tag names, start tags must always be matched by end tags unless they are *empty elements* (as in *<br/>*), tags are case-sensitive (i.e. *<author>* is not the same as *<Author>*), and tag overlapping is not allowed.

XML documents can be represented as tree structures. Each XML document must have (and each XML application must define) a unique root element of which all other elements are descendants. Figure 5.1 shows a well-formed XML document and its corresponding syntax tree.

The kind of XML encoding shown in Figure 5.1 is called data oriented. It resembles, in a way, the sort of structure one would find in a database management system. XML is obviously also appropriate for annotation of ordinary text, in which the structure is implicit, or more loosely encoded. This encoding style is often referred to as narrative-organized. The distinction, however,

```
<author>
<name>
<first>Thomas</first>
<last>Bernhard</last>
</name>
<gender>male</gender>
<Nationality>German</Nationality>
</author>
```

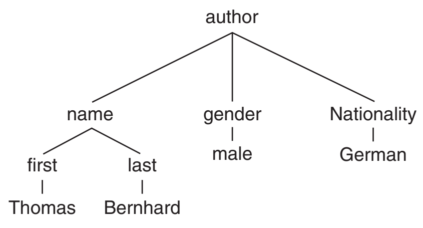

FIGURE 5.1 A simple XML document represented as a tree

```
<author>
The <Nationality>Irish</Nationality> writer <name> <first>Samuel</first>
<last>Beckett</last> </name> was born in <city>Dublin</city> in
<date><month>April</month> <year>1906</year></date>.
</author>
```

FIGURE 5.2 Narrative-organized XML fragment

is purely pragmatic. Note the presence, in Figure 5.2, of data-oriented elements (data and name) within the narrative-organized author element. In TEC, for instance, translated texts are encoded in a narrative-organized form, while the metadata encoding of its header files is data oriented.

5.3.2 Basic Syntax Rules

Angle bracket delimited tags such as the ones shown above form the basic units of XML markup. Pairs of tags and their contents are known as XML elements. XML elements can also have attributes. Attributes are name-value pairs attached to a begin tag, as shown in Example (1). An attribute name is separated from its values by an equals sign. Values must always be enclosed in single or double quotes.

(1) <author nationality=‘Irish’ sex=‘male’> Samuel Beckett </author>

The lexicon of an annotation scheme is the set of tokens which name its elements and attributes. In XML, the form these names can take is precisely defined by the following set of rules:

- Names may contain any alphanumeric characters (and also non-Latin characters and ideograms) as well as the following punctuation characters: ‘_’, ‘-‘ and ‘.’.
- White spaces and other punctuation characters are not allowed
- Names may start only with letters (or ideograms), or the underscore character.

Because certain characters play special roles in XML documents, their print format needs to be escaped in certain contexts to avoid ambiguity. The ‘less-than’ symbol (<), for instance, is always interpreted as a marker that an XML tag is about to appear. It must always be matched by a ‘greater-than’ character (>). Therefore, a fragment such as *<maths> 2 < x </maths>* is not well-formed XML. In such contexts, one needs to replace the < character by a special symbol sequence. The symbol sequence &lt; serves this purpose. This syntax (& . . . ;) allows XML users and document designers to specify any symbol. Symbols so defined are called entity references. The logical consequence of using the ampersand character to signal the beginning of an entity reference is that the ampersand symbol itself will need to be escaped. Therefore we have *&amp;* as the entity reference corresponding to the ampersand character.

In addition to tags, angle brackets are used in XML to encode comments and processing instructions. Comments can be added to improve the legibility of the document, and are simply ignored by applications. Comments start with a *<!–* and end with a *–>* sequence. Example (2) shows an XML comment. Comments are not allowed inside other comments or inside tags.

(2) <!– this is a comment –>
(3) <?robots index='yes' follow='no'?>

XML documents can pass processing instructions to other applications by enclosing them in tags of the following format: *<? . . . ?>*. An example of processing instruction is shown in Example (3), which is actually used in websites to suggest appropriate behaviour to indexing 'software robots' like the ones used by most search engines.

Although comments and processing instructions are markup, they are not XML elements. Therefore they can appear anywhere in the document. An exception to this rule is the XML declaration, a processing instruction which declares a number of facts about the document for use by parsers and other applications. An XML declaration is not a compulsory part of an XML document. However, if it is added, it must be the first string to appear in the document. Common attributes of XML declarations include: version, for backward and forward compatibility purposes, encoding (if none specified, UTF-8 is assumed), and standalone, which tells the parser whether to look for an external document definition. A typical XML declaration is shown below:

(4) <?xml version="1.0" encoding="ISO-8859–1" standalone="yes"?>

5.3.3 An Online XML Parser

Readers willing to follow a tutorial introduction to XML will find a set of resources at http://ronaldo.cs.tcd.ie/tec/CTS_SouthAfrica03/. One of the resources available at the site is a simple, online XML parser. In order to use it, one needs a web browser and Java Web Start,[2] which is available with all recent versions of the Java Run-time Environment. The data needed for all XML-related exercises can be accessed through the website or downloaded directly in an archive file.[3] Exercises covering well-formedness of XML documents can be found in data/01/ (a directory created by decompressing the archive file). Descriptions of the exercises can be found in data/01/README.txt.

5.3.4 Document Type Definitions and XML Validation

In addition to the basic conventions described above, XML provides mechanisms to enable the syntax of an annotation scheme to be fully specified. In TEC, for instance, two schemes have been defined: *techeader*, for encoding data about the corpus, including information about translations, translators, authors and so on, and *tectext*, which supports light annotation of the corpus itself in narrative-oriented style. Figure 5.3 shows a fragment of an actual TEC header file.

An XML application such as *tectext* or *techeader* can be defined through document type definitions (DTD). DTDs provide a formalism that allows document designers to specify precisely which elements and entities may to appear in a document, their hierarchical relationships, and their admissible contents and attributes. The following are examples of constraints one can specify through DTDs: 'a book must have an author', 'an author must have a first and a last name, and optionally a middle name', and so on.

Consider the DTD shown in Figure 5.4, for instance. Each line defines a valid XML element. The string *!ELEMENT* is a reserved word. It is followed by an element name (e.g. *book*) and a specification of the element's admissible children in the XML tree. Children elements can simply be parsed non-annotated data (including entity references) or other XML elements. The former is specified through the reserved word *#PCDATA*. In Figure 5.4, title has book as the parent element, and parsed data as children. Children can be specified as sequences or choices, or combinations of these. Sequences are comma-separated lists of element names (and possibly *#PCDATA*), as in (*title,author+,publisher?,*

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE techeader SYSTEM "techeader.dtd">
<techeader>
<title subcorpusid="fiction" filename="fn000001.txt">
<subcorpus>fiction</subcorpus>
<collection>Restless Nights.</collection>
</title>
<section id="fn000001.000">
<translator sexualOrientation="heterosexual" gender="male">
<name>Lawrence Venuti</name>
<nationality description="American"></nationality>
<employment>Lecturer</employment>
</translator>
. . .
</section>
</techeader>
```

FIGURE 5.3 TEC header file fragment using techeader.dtd

```
<!ELEMENT book (title,author+,publisher?,note*)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (fname,mname?,lname)>
<!ATTLIST author sex (male|female) #REQUIRED>
<!ELEMENT publisher ANY>
<!ELEMENT fname (#PCDATA)>
<!ELEMENT mname (#PCDATA)>
<!ELEMENT lname (#PCDATA)>
<!ELEMENT note (#PCDATA|p)>
     <!ELEMENT p EMPTY>
```

FIGURE 5.4 A simple DTD

note), where title, author, publisher and note must appear in the order specified by the declaration. Choices are represented by lists of elements separated by vertical bars, such as *(#PCDATA|p)*, which specifies that parsed data or <p/> tags (and possibly both, in any order) can appear as children. The number of children allowed is determined by special suffixes appended to element names in the DTD: '?' for zero or one occurrence, '*' for zero or more occurrences, and '+' for one or more occurrences. In the example below title must appear first, followed by one or more *author* elements, possibly followed by a *publisher* and so on. One can also combine constraints by using parentheses. An element name followed by XML reserved word ANY indicates that any (possibly mixed) content may appear as text tagged by this element.

Among the declarations in Figure 5.4 is one which specifies the allowable values for the *sex* attribute of element *author.* This declaration is identified by an *ATTLIST* token. The element author requires an attribute *sex*, which may take value '*male*' or '*female*', but no other value. Supplying another value would cause a validating parser to signal an error. Attributes appear inside element tags. The following is a sample *tectext* tag which tells the user why a certain text fragment should be ignored by the indexer: *<omit desc='picture'/>*. Figure 5.5 shows how this tag can be declared in a DTD.

An advantage of using attributes is that they allow contents to be more tightly constrained. The type of data acceptable as attribute values can be specified through certain reserved words. DTDs also allow document designers to specify default values for attribute slots or restrictions on how such slots should be handled (e.g. whether values are required or optional, fixed, defined as literals, etc.).

The user tells an XML parser how to validate a document by declaring the document to be of a certain type, as defined by a DTD. This is done through the *DOCTYPE* declaration, which provides a link between a document and its syntax specification. TEC headers, for instance, declare their DTD as follows: *<! DOCTYPE techeader SYSTEM 'techeader.dtd'>*. The word *techeader* indicates the document's root element, and the string following reserved word *SYSTEM* indicates the location of the DTD. This could also be a fully specified URL or,

<!ATTLIST omit	desc	CDATA	#REQUIRED>
↓	↓	↓	↓
declaration tag and element name	attribute name	data type	default

FIGURE 5.5 DTD declaration of a desc attribute for a tectext omit element

in fact, any URI. An alternative to *SYSTEM* is *PUBLIC*, which is used for publicly recognized DTDs, such as the DTD for the XML version of HTML.[4]

Despite the fact that they provide a simple and flexible mechanism for defining XML document syntax, DTDs have a number of limitations. First of all, there is a question of consistency. Although similar, DTD syntax isn't really XML. Some consider this to be a serious drawback. In addition, many applications require more expressive ways of constraining document syntax than DTDs can provide. This prompted the development of competing XML document specification schemes. The most widely used alternative to DTDs is XML Schema, a formalism developed by the World Wide Web Consortium whose syntax is itself XML-compliant.[5]

5.4 Adding Some *Semantics* to an Annotation Scheme

An attractive aspect of using XML for corpus annotation is that, once annotated, documents can be viewed from a variety of perspectives ranging from application needs to user tasks to corpus maintainer goals. Checking a text for well-formedness helps prevent typos and other simple annotation errors. Validating helps ensure consistency across the collection. Once these basic requirements are met, the user or maintainer can interpret the annotated text and benefit from the corpus designer's work in many ways. Although simple visual inspection of raw XML might provide the user with greater understanding of the data, the main benefit of a consistently annotated XML document derives from its amenability to processing by computers. Post-processing of annotated text may facilitate different levels of analysis. Annotation could be used, for instance, to control what should be indexed[6] or displayed, how the text should be laid out for presentation on a web browser or formatted for printing and so on. Markup adds structure to text and ultimately enables applications to assign meaning to that structure. A general processing architecture for annotated data is shown in Figure 5.6. First the original is annotated, then the resulting document is checked for well-formedness and validity, and finally the valid document is transformed for presentation, storage in a database,

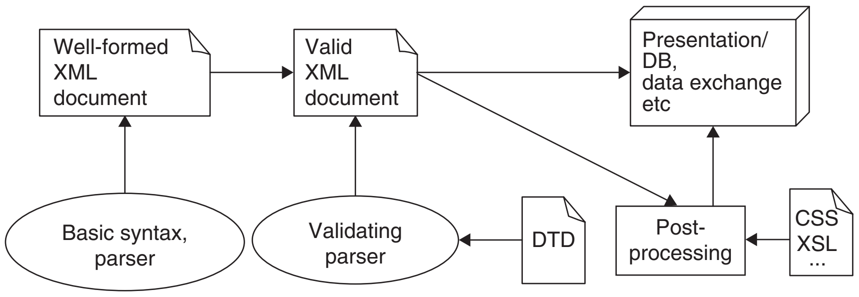

Figure 5.6 Overall XML processing architecture

data visualization, or whatever other application need the annotation scheme happens to support.

The semantics of a document structure can be defined in numerous ways. Some of them are quite ad hoc, such as the way TEC handles storage of metadata into an efficient database system in order to speed up retrieval within subcorpora. Others, however, have been standardized, as they range over a large set of applications. These include formatting instructions, which prompted the development of CSS, a language for defining Cascading Style Sheets (Bos et al. 2009), as well as more powerful formalisms for specifying general document transformations such as the Extensible Stylesheet Language (XSL) and its formatting counterparts (Clark 1999).

CSS is a simple and widely used language for formatting of XML-annotated text. CSS can also be used in conjunction with HTML documents, being supported by nearly all modern web browsers. Figure 5.7 illustrates how a CSS can be used in conjunction with an XML document to produce a formatting effect.

A CSS file consists of a list of elements followed by a list of style specifications to be applied to these elements. The most commonly defined properties include the display attribute, which specifies the way an element should be laid out on the main document (as block, inline, list-item, table, or none), how elements are formatted inside a table (inline-table, table-row, table-column, table-cell etc.), and general lengths, such as font-size, border-width, width and height. There are many other properties (e.g. font style and colours) whose enumeration is outside the scope of this chapter.

In order to link an XML file with a given CSS one uses a specific processing instruction of the form *<?xml-stylesheet type='text/css' href='mystyle.css'?>*, where the value of the *href* could have been any valid URI. If viewed on a CSS-compliant web browser, an XML text containing the above processing instruction will be formatted according to the rules specified in *mystyle.css*.

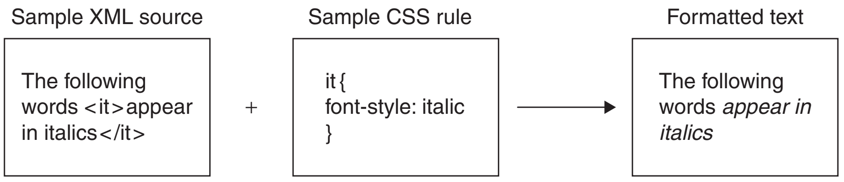
Sample XML source
The following words <it>appear in italics</it>
+
Sample CSS rule
it{
font-style: italic
}
Formatted text
The following words *appear in italics*

FIGURE 5.7 XML fragment formatted as specified by CSS rules

In corpus applications, CSS can be used for formatting concordance lines (i.e. with keywords aligned at the centre of the screen and contexts flushed left and right), for example. The tutorial website contains an exercise on creating a CSS file for formatting XML-annotated concordance lines.

XML semantics does not necessarily need to be implemented through style sheets or any standard language, for that matter. A common form of post-processing of XML documents is storage and retrieval of elements in a database system. Since XML itself is not meant to be a database system, application-specific code is often used to load selected elements into a database management system (DBMS) for efficient access. The TEC system uses a native XML DBMS (Meier 2002) to store its metadata.

Should one wish to learn more about XML, there are many books on XML, ranging from short introductions to XML itself to comprehensive guides to XML-related technologies. *XML in a Nutshell* (Harold and Means 2004) is a good introductory book which manages to contain a reference guide to the main XML-related technologies. Less readable but more detailed are the official W3C technical reports (Clark 1999; Clark and DeRose 1999; Bray et al. 2006; Bos et al. 2009) which can be used as concise reference texts once one has mastered the basics of XML.

5.5 Text Indexing and Retrieval

In addition to markup standards, corpus software in general needs to be able to perform basic text indexing and retrieval functions. This section provides an introduction to the main issues in indexing and retrieval. As before, the contribution is set against a TEC backdrop, so the focus will be on the choices made in the implementation of the TEC system.

Text retrieval involves four of basic operations: tokenization, indexing, compression and search. Tokenization, or lexical analysis consists of deciding what counts as a token (i.e. selecting strings for indexing). Indexing consists of storing information about where those tokens can be found. Compression aims at keeping indices within acceptable space bounds. Search is the basic operation on which corpus analysis tools such as concordancing and collocation are built. The first three operations are often performed offline, in a single step, while search is usually performed online.

5.5.1 Tokenization

As mentioned above, tokenization is the process of converting a stream of characters into a stream of tokens (or words). At a higher level of abstraction one finds what is known as lexical analysis. The key question in tokenization is: what counts as a word delimiter? In English, for instance, the best candidates are blanks (including tabulation spaces, line breaks, etc.), hyphens and punctuation characters. One might, in principle, define a token as a string of characters delimited by those characters. In certain cases, however, this definition is inadequate. The hyphens in *state-of-the-art* separate out legitimate tokens, whereas the hyphen in *B-52* does not. Punctuation characters can be equally deceptive, as in *360 B.C.* A related issue is how to deal with capital letters. Should *White* as in *The White House* be indexed separately from *white* as in *white cells*? Sometimes capitalization provides valuable hints about collocation. The TEC system adopts the approach of preserving as much information as possible: hyphenated words are indexed both as individual words and as compounds, superficial lexical analysis rules out tokens such as *B* in the examples above, and case is preserved through the use of two separate indices (case-sensitive search is assumed by default). Figure 5.8 shows the overall layout of the inverted index used in TEC.

Tokenization is an essential phase of corpus processing and involves reading through the entire string of text and testing for pattern matches. Fortunately, the computational cost of performing tokenization is relatively low. Finite-state automata (FSA) can tokenize in linear time. Regular expressions provide convenient syntax for token matching which can be converted into suitable FSA. Regular expressions are well supported in modern programming languages. Interpreted languages such as Perl and Python provide built-in support for regular expression matching. Other languages such as Java, C, and C++ provide a variety of libraries for dealing with pattern matching.

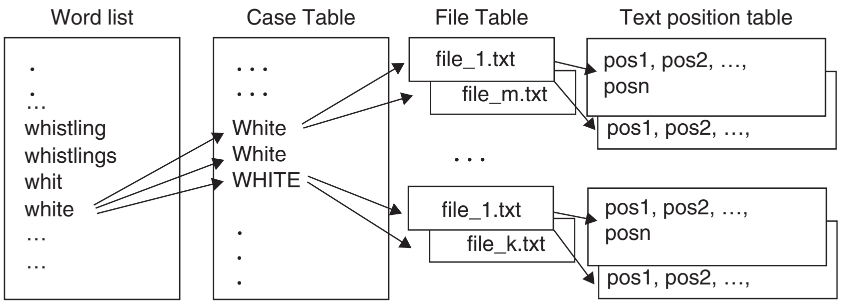

FIGURE 5.8 Structure of case-preserving inverted indices in TEC

A regular expression is a pattern that denotes a (possibly infinite) class of strings to match. Basic regular expression syntax is very simple. Its atomic symbols are: *e*, which matches any character, and ε, which matches the empty string. In addition to ordinary atomic symbols, regular expressions can contain suffix operators, binary operators and parentheses. Suffix operators quantify over atomic symbols much in the same way as suffix operators quantify over elements in DTDs (see Section 5.3.4). The suffix operator * in *e** denotes a sequence of zero or more characters *e*. All the other suffix operators can be defined in terms of * (plus atomic symbols and binary operators). Binary operators | and, are also analogous to their counterparts in DTDs. Regular expression *e*1|*e*2 matches character *e*1 or character *e*2, while the expression *e*1,*e*2 matches character *e*1 immediately followed by character *e*2 (and is usually abbreviated as *e*1 *e*2). Parentheses are used for grouping sub-expressions.

The following simplification might help illustrate the use of regular expressions. Imagine a language whose alphabet consists entirely of two letters: *a* and *b*, and in which white spaces are word separators. A regular expression for tokenizing texts written in this language could look like this: *(a|b)(a|b)*_*.

5.5.2 Indexing

In small corpora, it might be practical to use simple string or even regular expression matching to determine the exact occurrences of a query, typically a keyword. Although matching is reasonably fast, it is still not fast enough for use in real-time with larger corpora. Even for medium-sized corpora such as TEC one needs to use pre-compiled indices in order to attain an acceptable search speed. Pre-compiling an index simply means matching all possible keywords in advance (offline) and storing the results into a data structure that can be searched more efficiently online.

Text indexing techniques have been extensively used in the area of Information Retrieval (IR). IR techniques can be straightforwardly adapted for use in corpus processing, and there are many indexing techniques on offer. Each has advantages and disadvantages. As with most applications, one's choice of indexing technique for corpus processing will depend on the size of the database, search speed constraints, and availability of storage space. In what follows, we focus on our choice for the TEC corpus, inverted indices, and describe it in detail. However, we also say a few words about other common indexing strategies, with a view to contextualize this choice, and refer the interested reader to Baeza-Yates and Ribeiro-Neto (1999) for further details.

Generally speaking, indexing strategies differ according to the data structures they use. Inverted indices (also known as inverted files) are tables whose keys are words and whose values are lists of pointers to all positions in which the key occurs in the corpus. In suffix tree and suffix array indexing, each position in the text is considered a suffix. This facilitates search for phrases and longer text fragments as well as word prefixes. Suffix arrays and trees allow fast search but have heavy storage space requirements. Signature files, on the other hand, have much lower space requirements at the price of search speed. Signature files divide the text into blocks each of which is assigned a bit mask. Words are mapped to these bit masks by means of a hashing function. The bit map of a block is obtained by combining the signatures of all individual words in the block. Therefore, searching for a word consists in finding all blocks whose bit maps match the word's signature and performing sequential search on them. Search time in signature files is high compared to inverted indices and suffix trees. A comparison of the main indexing strategies is shown in Figure 5.9. The graph indicates that although inverted files do not exhibit the fastest performance, they represent a good compromise between space requirements and search speed.

In TEC, index construction is very simple: as the text is tokenized, the exact position of each token is recorded into a file (called the inverted index, or inverted file). Figure 5.10 shows a possible index file consisting of a table mapping words to lists of word positions.

Inverted indices are generally space-efficient. Space required to store the vocabulary is rather small in comparison to corpus size. Heaps' Law states that the size of the vocabulary grows sub-linearly on the size of the text.[7] Storing the position of each word in the text, however, takes by far the most space and might cause space requirements to increase greatly if one is not careful. The

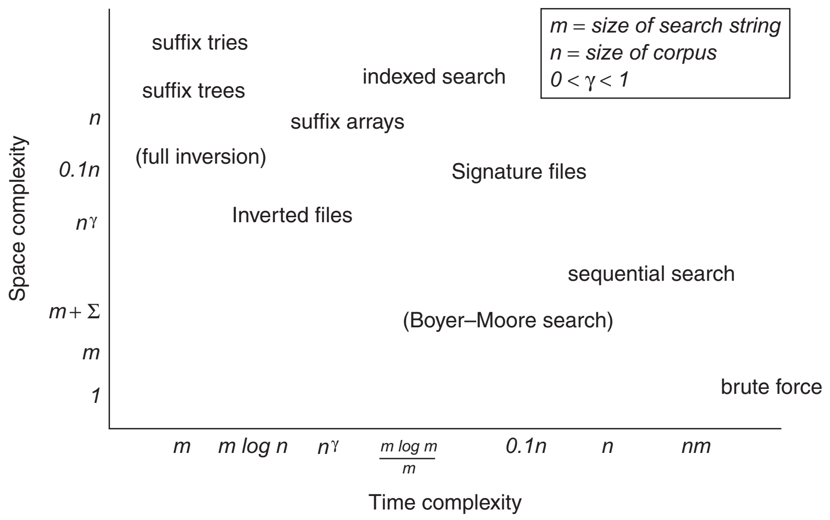

Figure 5.9 Space and time complexity of most common indexing techniques, adapted from Baeza-Yates and Ribeiro-Neto (1999)

Each	Word	is	analysed.	The	position	of	each	word . . .
1	6	11	14	25	29	38	41	46

vocabulary		**positions**
each	→	1, 44
word	→	6, 46
is	→	11
. . .	→	. . .

FIGURE 5.10 Sample text fragment (top) and corresponding inverted file (bottom). Numbers indicate word positions

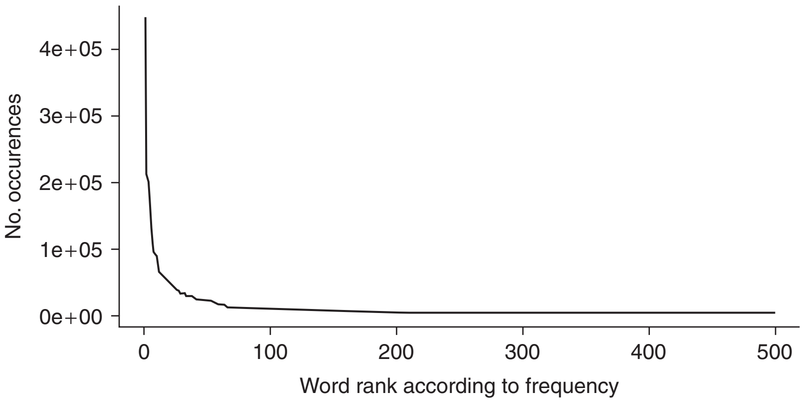

FIGURE 5.11 Word distribution in TEC

number of positions to be stored is distributed according to Zipf's Law. In its simplest formulation, Zipf's Law states that the most frequent word occurs *x* times as frequently as the most frequent word. This pattern has been observed in TEC, whose word distribution curve is shown in Figure 5.11. As inverted indices consist of lists of integers each of which points to a position in the text, keeping space requirements within acceptable bounds implies finding an economical way of storing such lists.

A number of index compression techniques exist. In TEC, we have opted for a simple but effective approach. Instead of storing absolute word positions, we simply store the position of the first occurrence followed by offsets pointing to subsequent occurrences, as shown in Figure 5.12. This technique of storing relative rather than absolute values achieves a 57 per cent index size reduction for a 33,398-word file.

$$\boxed{\underbrace{word}_{\text{key}}\quad \underbrace{pos_1, pos_2, \ldots, pos_n}_{\text{start positions}}\qquad \underbrace{word}_{\text{key}}\quad \underbrace{pos_1, pos_2 - pos_1 \ldots, pos_n - pos_{n-1}}_{\text{relative positions}}}$$

FIGURE 5.12 Schematic representation of two indexing strategies: uncompressed index versus index compression used in TEC

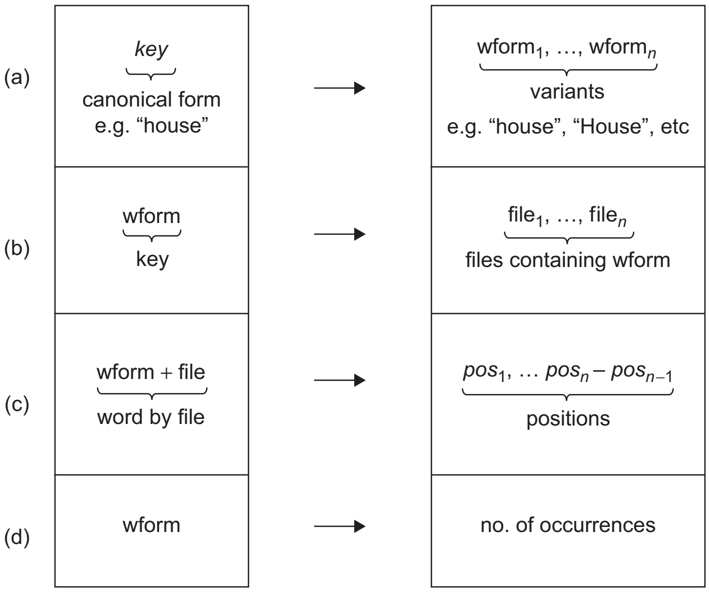

FIGURE 5.13 TEC inverted index structure

Because most of the search task (tokenization and indexing) is really performed offline, searching for a keyword via inverted indices is usually fast. Actual performance varies depending on how the index is stored. If the index is simply stored as a list sorted in lexicographical order, it can be searched in binary mode which yields search time proportional to the size of the vocabulary (i.e. $O(\log n)$), where n is the number of words in the vocabulary. However, if indices are stored as *hash tables* or as *tries* very fast retrieval can be achieved. Search times drop to linear in the size of the word searched for (i.e. $O(m)$, where m is the size of the word).

The TEC system stores its inverted index in hash tables. In fact, in order to optimize various kinds of queries the system builds a set of interrelated tables. Its index structure is shown in Figure 5.13. The system allows case-sensitive as well as case-insensitive search, so it creates separate entries for different forms of a word with respect to the capital letters that occur in it (Figure 5.13 (a)) but also stores a canonical word form for case-insensitive searches (Figure 5.13 (b)). Tokens are primarily associated to the files in which they occur, so start positions are recorded for each file (Figure 5.13 (c)). Thus files can be easily removed or re-indexed if needed. Finally, the system stores a pre-compiled frequency list (Figure 5.13 (d)). This last table is redundant, as its content can be retrieved from the other three tables. However, being able to access frequency information instantly enables the system to provide immediate feedback to the user on the progress of the search. This is an essential feature in web-based interfaces.

5.6 Data Storage and the Internet

The overall architecture of the internet is based on a client-server model. In this kind of model, data are stored in a central location (the server) and accessed through specialized software running on remote machines (the clients). This model exhibits two interesting features: it allows for resources to be distributed while preserving data integrity (as data are centrally maintained), and it allows for heterogeneity of access, as long as different clients agree to abide by the protocol implemented by the server.

Why are these features interesting for corpus users? First of all, the client-server model helps overcome copyright issues. A great deal of material of interest to translation studies scholars is subjected to copyright restrictions.[8] Distributing large volumes of copyrighted material as an integral corpus would be very costly, if at all feasible. However, not all data need to be available in order for corpus research to be carried out. Corpus researchers are mostly interested in uncovering patterns, often aided by metadata, rather than sequential access to an entire text. The client-server model enables corpus maintainers to restrict access to indirect retrieval of text fragments, thus protecting copyright. By removing this barrier, the client-server model facilitates sharing of resources among researchers, and might help provide the research community with larger volumes and variety of data than has been available so far. In addition, using the Web as a distributed storage medium has other advantages, such as the existence and ubiquity of standards and ease of access.

On the other hand, the model also has potential disadvantages. The most conspicuous disadvantage is the fact that the performance of a web-based corpus tool tends to be poor if compared to single-user tools. The speed of execution of a web-based client is directly affected by the bandwidth and latency of its communication channel to the server (the network bottleneck problem). Although the general trend is for network connections to become faster and more reliable, there is little corpus providers and developers of corpus clients can do to alleviate the network bottleneck problem. Another problem is the instability introduced by relying on a centralized server. If a server crashes, or has its network connection interrupted, all clients are affected.

5.7 The TEC Browser: A Tutorial Introduction

The TEC browser can be started directly via the Web.[9] If accessed through a web browser, the corpus browser will automatically detect and use the network settings of the web browser. Because the TEC system uses the same communication protocol as a standard web server, the browser can operate seamlessly across network security devices such as firewalls.

The main functionality of the TEC browser is the retrieval of concordances. Concordances are retrieved by entering a query on a search box. The general format of a query expression accepted by the browser is the following: *word_1(+([no_of_intervening_words])word_2 . . .)*, where brackets denote optional items. The expression *the+end*, for example, will match all occurrences of the immediately followed by end. Note that if *[no_of_intervening_words]* is omitted, the system assumes it to be 0. So *the+end* is equivalent to *the+[0]end*. One can retrieve all concordances for the phrases *the very end, the end* and *the wrong end* simply by specifying *the+[1]end* as query expression.

In addition to word sequences and intervening words, the server also accepts 'wildcards'. Wildcard syntax allows the user to select word prefixes by appending an asterisk to the query. For example, typing in *test** will retrieve all words which start with test, including test, testament testimony etc. Any *word_n* token in the query syntax expression described above can be replaced by a word or a wildcard. An expression such as *a*+test** is a perfectly valid query which will return *a test, acid test, about testing, a testament* among other phrases. The same result could obviously be achieved by searching for, say, *test** and sorting by the left context. This last strategy, however, would be less efficient since all concordances would need to be transmitted to the client, and transmission delay is the main factor affecting the performance of the browser, as discussed above. Another factor that affects performance in searching for combined keywords is the choice of the primary keyword. The following example illustrates this point. Suppose one is interested in retrieving the expression *the lonely heart*. Although TEC contains a single instance of this phrase, the corpus contains over half a million occurrences of the article *the*, over 3,500 occurrences of the word *heart*, and about 30 occurrences of the word *lonely*. If one chooses *the* as the primary keyword, the system will have to read though over half a million word sequences in order to find that single instance. Searching for *heart* would certainly improve things, but the best choice would be to take *lonely* as the primary keyword, as this would reduce the computation to at most 30 comparisons. If a word sequence query such as *the+lonely+heart* is submitted, the system will automatically choose lonely as the primary key, thus minimizing search time. It is nearly always a good idea to submit a sequence query if a sequence of words is what one is after.

There are situations, however, in which one needs to be able to retrieve all concordances for a given word and then explore possible collocations. A sorting function can be used to support this kind of exploration. TEC allows sorting by left and right contexts of various sizes. 'Sort context' pull-down menus located next to the sort buttons allow the user to specify how many words to the left or right of the keyword the list will be sorted by. Sort keywords appear highlighted by colour on the concordance list. One could, for instance, search for test, and then use sorting to group together occurrences of a test, acid test, the test etc. Once a concordance list has been downloaded, sort can be activated by the sort buttons. If one attempts to start sorting before downloading is complete, the system will ask whether the user wants to interrupt the download operation and truncate the concordance list. Although sorting is quite efficient, approaching linear performance in certain cases, it illustrates one of the advantages of a distributed architecture: since sorting is done directly on concordances, it can be entirely performed by the client (i.e. the corpus browser), thus freeing the server to perform searches.

Search can also be constrained by selection of sub-corpora. We have seen above that the information in TEC header classifies the various text files according to number of attributes of its authors, translators and so on. These attributes can also be used to define sub-corpora. Since the original XML-encoded metadata contained in the header files has been parsed and stored in a native XML database (Meier 2002), sub-corpus selection can be done through standard XPATH syntax. For example, the following query will select a sub-corpus containing texts written by Belgian or Brazilian men, and translated by either British or Canadian women[10]:

```
(5) ($s/author/@gender='male') and ($s/translator/@gender='female') and
    ($s/author/nationality/@description='Belgian' or
    $s/author/nationality/@description='Brazilian') and
    ($s/translator/nationality/@description='British' or
    $s/translator/nationality/@description='Canadian')
```

The sub-corpora selection tool is activated via the 'Options' menu on the TEC browser menu bar. Choosing 'Select sub-corpus . . . ' brings up the selection tool. The sub-corpus selection tool is a window which contains a number of selection boxes representing metadata attributes and their possible range of values. This allows a form of sub-corpus selection by direct manipulation. For both authors and translators, the user can specify the author's (or translator's) name directly, or a combination of the following attributes: gender, nationality and sexual orientation. Selections are activated by highlighting the items on the selection boxes. Alternatively, the user can enter XPATH queries such as the one in Example (5) directly.

In addition to its core functionality of concordancing and sub-corpus selection, the TEC browser also supports *Plug-ins.* Plug-ins are tools that perform specific functions, building on the main functionality provided by the core browser (GUI, network connection, concordancing, etc.). TEC has a few external plug-ins which serves to illustrate the concept: a frequency list browser, a corpus descriptor and dynamic concordance visualization tool. Other TEC plug-ins currently under development include: selection by part-of-speech tags and collocation analysis.

5.8 A Vision of Web-Based Corpus Software

Before discussing the future of the TEC system and perspectives on the future of web-based corpus software in general we must make a few remarks on the current status of the software. The current TEC architecture employs the standard client-server model depicted in Figure 5.14. Whereas the basic functionality for corpus indexing and access is handled at the server side, the client handles not integral texts directly but concordances generated by the server. The client itself is implemented in a modular architecture that enables new functionality to be easily incorporated. The TEC client may also communicate with a standard web (HTTP) server for requests that involve direct retrieval of metadata. The reasons for splitting the server functionality between a specialized concordance retrieval module and a generic content server are related to performance and security issues. The concordance server can perform more efficiently if it is dedicated to retrieving concordance data. Furthermore, having a dedicated concordancer makes it easier to

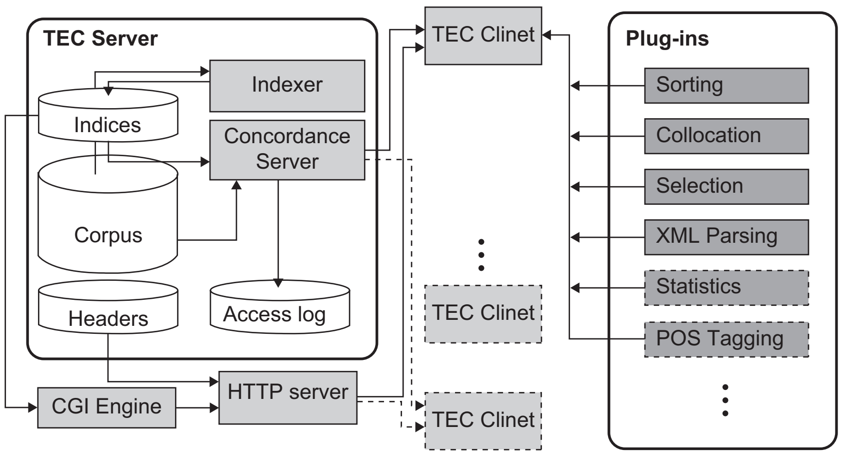

Figure 5.14 The current architecture of the TEC system

guarantee that copyrighted material will not be accidentally made available on the internet.

The TEC tools have been implemented in Java, as part of a suite of language processing tools called *modnlp*. The client can run as a stand-alone program, or over the Web using Java Web Start[11] technology. The entire system has been developed as free software and is distributed under the GNU Public License (GPL[12]). Code, licence and documentation are available at the modnlp website.[13]

5.8.1 Processing Models and Web-Based Corpora

In computer science, the areas of parallel and distributed computing study ways in which processors can be combined in order to improve the efficiency and flexibility of a system. In computational terms, distribution can be achieved in two forms: distribution of data and distribution of processing power. One uses the expression *data stream* to refer to the flow of data from a processor to another, and analogously, *processing stream* when referring to the flow of instructions from a processor to another. There are four classes of (logically possible) architectures with respect to the distribution of computational resources: SISD (single instruction, single data streams), MIMD (multiple instruction, multiple data streams), SIMD (single instruction, multiple data streams) and MISD (multiple instruction, single data streams). Although these classes are primarily used as a taxonomy of computer hardware, at least the first two of them are relevant to the way corpus software works.

While the model adopted by most single-user tools for corpus indexing is analogous to SISD, the TEC architecture depicted in Figure 5.14 can be considered as a MIMD architecture. A few differences must be noted, however. In MIMD architectures, subtasks are usually allocated different processors on the same (multiprocessor) machine. Coordination, data exchange and control therefore often relies on shared memory-based inter-process communication mechanisms. TEC processing tasks, on the other hand, will typically be allocated to remote processors and communication will therefore use network-based communication mechanisms and protocols (e.g. TCP/IP). For example, in the current system, while the server processor is occupied in retrieving concordances and sending them across the internet to the various clients, each client might be displaying the concordances it has received, and placing other requests (e.g. metadata) to the server. This type of loosely coupled interaction is characteristic of distributed architectures.

In the context of our corpus-software application, we can perhaps establish a further distinction within the MIMD class: single-server versus multiple-server architectures. The current TEC system operates as a single-server architecture. The entire corpus is stored at a central location and manipulated by a single processor. Although this kind of architecture helps improve access by remote users, it does little, if anything, to improve information providing. Different user groups that share an interest in corpus research have similar needs, some of which might be met by in-house resources (e.g. a small-scale corpus of translated text). Corpus research often involves analysing data from different source corpora. TEC users, for instance, use corpora like the BNC in order to compare language usage in translated and non-translated English. Ideally, it should be possible for diverse corpus sources to be pooled into a common framework for corpus processing. One might argue that such framework could be implemented as a single-server model simply by using a large, centralized corpus. However, copyright issues and other practical constraints present problems for centralized models. An alternative approach would be to develop a uniform interface that would mask the complexities and physical locations of various, heterogeneous services, as suggested by Sharoff (2006). However, such an approach would still require centralized management and manual updating.

Arguably, the essence of a multiple-server approach to corpus processing is to enable geographically distributed research groups to build smaller-scale corpora and share them through their own servers, on their own terms. Clients would then be able to discover and query selected servers, and autonomously combine the resulting responses into a coherent presentation. Some progress has been made towards automating this process of service description and discovery in the area of service-oriented computing (Papazoglou et al. 2008) but tools to enable easy sharing of language resources in this manner are still scarce.

A typical usage scenario for this improved architecture would involve the client selecting a set of corpora to be searched and broadcasting queries to selected corpus servers, each server evaluating the query (independently and in parallel), and the client receiving and combining the results from each server into a final result to be displayed to the user. An extension of the current architecture of the TEC system to implement this scenario is shown in Figure 5.15.

5.8.2 Challenges

The key challenges to truly distributed web-based corpus software are corpus selection and server capability description.

Corpus discovery and selection can be framed in the context of over a decade of efforts aimed at encoding metadata. These efforts range from language-specific standards such as the ones promoted by the Text Encoding Initiative (Ide and Veronis 1995) to more ambitious proposals (e.g. Zanettin, this volume, Chapter 4). The issues involved are essentially how to encode metadata and what information to encode. Although some progress has been made regarding the former, the latter is still very much an open problem. The development of credible standards such as XML helps solve the problem of how to

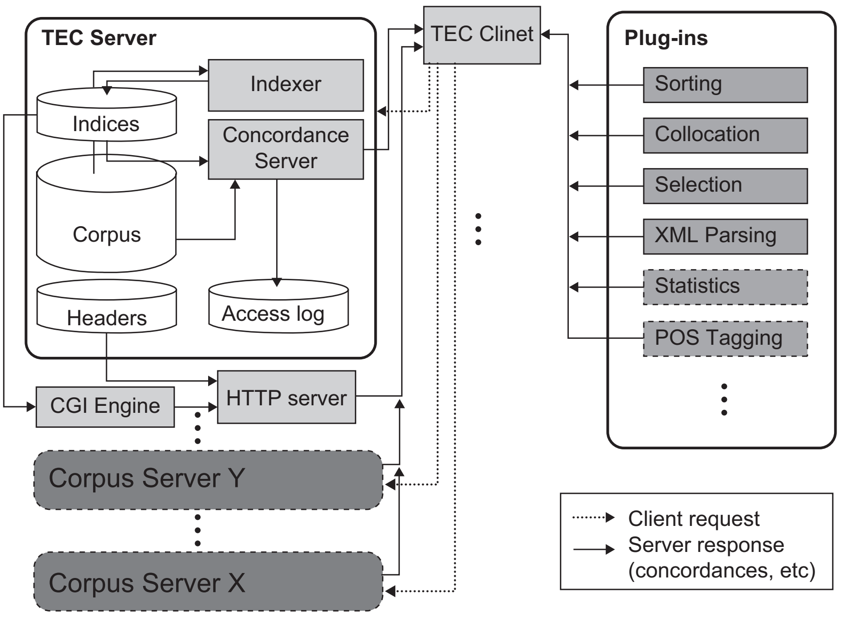

FIGURE 5.15 Multiple-server TEC system architecture

encode metadata so as to allow interoperability of different applications. New XML-based standards are being developed which aim to do the same at the semantic level. What is less clear is how this semantically structured information can be used. A typical dilemma of metadata encoding concerns determining how strict its semantics should be. If the semantics is too restrictive, it will yield over-specific metadata. If it's too lax, it will make it difficult for software clients to handle the potentially large variety of metadata created by the various servers.

Solutions to these problems might well have to be domain-specific. In the domain of translation studies, for instance, one could aim to define an extensible but minimal set of metadata items relevant to the community of corpus users and build basic software functionality around it.

Describing operational capabilities of servers is an issue that appears to have received considerably less attention from web researchers and language resources organizations. This is perhaps due to the fact that it is generally assumed that the sole function of a server of language resources is to make such resources available to its users, via the web or by other means of distribution. However, a quick look at the TEC server's processing capabilities described above suffices to reveal that this assumption is inadequate. In order to function properly, clients might need access to details such as whether the server supports case-sensitive search, what exactly it considers as a token, whether it supports search by part-of-speech tags, and numerous other features. Flexible, loosely coupled distribution of corpora and corpus software cannot be achieved unless it is supported by reliable capability description mechanisms. Emerging technologies such as multi-agent systems and peer-to-peer computing might play an important role in bringing about these mechanisms.

5.9 Conclusion

Although there are still many obstacles to the implementation of a model for the processing of distributed corpora as a complement to existing systems for distributed processing of corpora, recent developments in the areas of annotation standards and internet technologies suggest that this goal is achievable. This chapter has described several technologies currently used in web-based applications which might help bring this to fruition.

Notes

1 The term *document* has in this chapter, as in the SGML/XML literature, the connotation of a unit of data which is often, though not necessarily, of a textual nature.
2 http://java.sun.com/products/javawebstart/
3 http://ronaldo.cs.tcd.ie/tec/CTS_SouthAfrica03/data.tgz
4 <!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Strict//EN' 'http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd'>
5 See http://www.w3.org/XML/Schema for details.
6 As in TEC, where omit tags are used to inhibit indexing of non-translated material which would otherwise contaminate the corpus.
7 In other words, vocabulary size $v = O(n\beta)$, where $0<\beta<1$ and n = text size.
8 TEC, for instance, consists largely of translated fiction and biographies, all of which is copyrighted material.
9 http://ronaldo.cs.tcd.ie/tec2/jnlp/
10 The XQUERY selection statement is automatically added by the browser.
11 http://java.sun.com/products/javawebstart
12 http://www.gnu.org/
13 http://modnlp.berlios.de/

References

Baeza-Yates, Ricardo and Berthier Ribeiro-Neto (1999) *Modern Information Retrieval*, London: Addison-Wesley-Longman.

Baker, Mona (1999) 'The Role of Corpora in Investigating the Linguistic Behaviour of Professional Translators', *International Journal of Corpus Linguistics* 4(2): 281–98.

Bray, Tim, Jean Paoli and C. M. Sperberg-McQueen (eds) (2008) *Extensible Markup Language (XML),* Version 1.0. W3C recommendation REC-xml-19980210. Available online at: http://www.w3.org/TR/REC-xml

Bos, Bert, Tantek Çelik, Ian Hickson and Håkon Wium Lie (eds) (2009) *Cascading Style Sheets Level 2 Specification.* Available online at: http://www.w3.org/TR/CSS2/ (accessed 31 May 2010).

Clark, Jim (ed.) (1999) *XSL Transformations (XSLT) Version 1.0.* W3C recommendation REC-xslt-19991116. Available online at: http://www.w3.org/TR/xslt/

Clark, Jim and Steve DeRose (eds) (1999) *XML Path Language (XPath) V. 1.0.* W3C recommendation REC-xpath-19991116. Available online at: http://www.w3.org/TR/1999/xpath/

Harold, Elliotte Rusty and W. Scott Means (2004) *XML in a Nutshell,* third edition. Cambridge: O'Reilly & Associates, Inc.

Ide, Nancy M. and Jean Veronis (1995) *Text Encoding Initiative: Background and Contexts*, Dordrecht, The Netherlands: Kluwer Academic Publishers.

ISO8879 (1986) *Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML)*, International Organization for Standardization, Geneva, Switzerland, first edition.

Luz, Saturnino and Mona Baker (2000) 'TEC: A Toolkit and API for Distributed Corpus Processing', in Steven Bird and Gary Simmons (eds) *Proceedings of Exploration-2000: Workshop on Web-Based Language Documentation and Description,* Philadelphia: University of Pennsylvania, 108–12.

Manning, Christopher D. and Heinrich Schütze (1999) *Foundations of Statistical Natural Language Processing*, Cambridge and Massachusetts: The MIT Press.

Meier, Wolfgang (2002), 'eXist: An Open Source Native XML Database', *Web-Services, and Database Systems*, Berlin: Springer-Verlag, 169–83.

Papazoglou, M. P., Paulo Traverso, Schachram Dustdar, Frank Leymann and B. J. Krämer (2008) 'Service-Oriented Computing: a Research Roadmap', *International Journal of Cooperative Information Systems* 17(2): 223–55.

Sharoff, Serge (2006) 'A Uniform Interface to Large-Scale Linguistic Resources', in *Proceedings of the Fifth Language Resources and Evaluation Conference, LREC 2006*, Genoa, Italy, 538–42.

Zanettin, Federico (this volume, Chapter 4) 'Hardwiring Corpus-Based Translation Studies: Corpus Encoding'.