Twenty-first century Corpus Workbench: Updating a query architecture for the new millennium

Stefan Evert[*] and Andrew Hardie[†]

[*] University of Osnabrück
[†] Lancaster University

severt@uos.de ; a.hardie@lancaster.ac.uk

Abstract

Corpus Workbench (CWB) is a widely-used architecture for corpus analysis, originally designed at the IMS, University of Stuttgart (Christ 1994). It consists of a set of tools for indexing, managing and querying very large corpora with multiple layers of word-level annotation. CWB's central component is the Corpus Query Processor (CQP), an extremely powerful and efficient concordance system implementing a flexible two-level search language that allows complex query patterns to be specified both at the level of an individual word or annotation, and at the level of a fully- or partially-specified pattern of tokens. CWB and CQP are commonly used as the back-end for web-based corpus interfaces, for example, in the popular BNCweb interface to the British National Corpus (Hoffmann et al. 2008). CWB has influenced other tools, such as the Manatee software used in SketchEngine, which implements the same query language (Kilgarriff et al. 2004).

This paper details recent work to update CWB for the new century. Perhaps the most significant development is that CWB version 3 is now an open source project, licensed under the GNU General Public Licence. This change has substantially enlarged the community of developers and users and has enabled us to leverage existing open-source libraries in extending CWB's capabilities. As a result, several key improvements were made to the CWB core: (i) support for multiple character sets, most especially Unicode (in the form of UTF-8), allowing all the world's writing systems to be utilised within a CWB-indexed corpus; (ii) support for powerful Perl-style regular expressions in CQP queries, based on the open-source PCRE library; (iii) support for a wider range of OS platforms including Mac OS X, Linux, and Windows; and (iv) support for larger corpus sizes of up to 2 billion words on 64-bit platforms.

Outside the CWB core, a key concern is the user-friendliness of the interface. CQP itself can be daunting for beginners. However, it is common for access to CQP queries to be provided via a web-interface, supported in CWB version 3 by several Perl modules that give easy access to different facets of CWB/CQP functionality. The CQPweb front-end (Hardie forthcoming) has now been adopted as an integral component of CWB. CQPweb provides analysis options beyond concordancing (such as collocations, frequency lists, and keywords) by using a MySQL database alongside CQP. Available in both the Perl interface and CQPweb is the Common Elementary Query Language (CEQL), a simple-syntax set of search patterns and wildcards which puts much of the power of CQP in a form accessible to beginning students and non-corpus-linguists.

The paper concludes with a roadmap for future development of the CWB (version 4 and above), with a focus on even larger corpora, full support for XML and dependency annotation, new types of query languages, and improved efficiency of complex CQP queries. All interested users are invited to help us shape the future of CWB by discussing requirements and contributing to the implementation of these features.

1. Introduction

This paper outlines recent development work on the *Corpus Workbench* or CWB software, aimed particularly at bringing a tool whose origins lie in the early 1990s up to date with the requirements that contemporary users have for a twenty-first century corpus analysis system. CWB, of which the present authors are the current lead developers, is a widely-used, and extremely powerful and flexible, architecture for index-based querying of large and/or linguistically annotated corpora.

The paper is organised as follows. In section 2, we provide some background on CWB, exploring in particular three key features: the overall structure of the system; the model of corpus data, organised into a series of corpus *attributes*, on which it is based; and the query syntax associated with the central *Corpus Query Processor* (CQP) component. We will also briefly review a number of web-based corpus analysis systems which use CWB/CQP as their back-end – these being the route through which most corpus analysts have made use of CWB, given that the core system itself is an expert-level tool not targeted at beginners. Subsequently, in section 3, we discuss five different areas in which recent versions of CWB (beginning with version 3.0) have changed to reflect updated design requirements. Each of these five areas – the development process, CWB's support for multiple operating systems including Windows, the shift to a 64-bit environment, CWB's support for multiple character encodings including Unicode, and advances towards more user-friendly interfaces – represents a point where CWB's design was perfectly reasonable by the standards of the early 1990s, when the project to develop CWB began at the IMS Stuttgart, but is no longer reasonable in the changed landscape of the 2000s and early 2010s. Finally, in section 4, we discuss ways in which we anticipate the development of CWB will continue from this point forward, adapting yet further to the demands placed on it by its user community.

2. Some background on CWB

2.1. What is CWB?

CWB, the *IMS Open Corpus Workbench*,[1] is somewhat misleadingly named, as it is not in any sense a comprehensive or general "workbench" for corpus linguistics. Instead, it is a powerful and flexible system for indexing and searching corpus data. It is especially designed for use with annotated data, particularly medium to very large corpora with multiple kinds of linguistic annotation at word level (e.g. part-of-speech tags, lemmatisation, semantic tags or chunk boundaries). To this end it provides a powerful and flexible data model and query language. As such, CWB stands in contrast to concordancers such as WordSmith (Scott 1996)[2] or AntConc (Anthony 2005, 2009).[3] Systems like AntConc and WordSmith are designed to work with plain-text corpora (although WordSmith does also have a mode where it can create and work with indexed data), generally of rather small extent; they lack built-in support for complex annotation; their query language is based on regular expression or wildcard queries on the raw, underlying text; but they have the very great benefit of allowing novice users to perform automated analysis on *ad hoc* datasets. They are thus ideal tools for beginners and especially students. CWB differs from such systems on all these points. Most particularly, it is not really targeted at the beginner, and requires a reasonably detailed understanding of its data model and a familiarity with working with command-line tools. CWB can, then, be fairly characterised as an expert system for advanced users of corpus tools – although, as we will illustrate, systems that use CWB as a back-end can be targeted at beginners in a way that CWB itself is not.

In its current form, CWB actually consists of three different software packages: (i) the CWB core, including the low-level Corpus Library (CL), the CWB utilities, and the Corpus Query Processor (CQP); (ii) the CWB/Perl interface – itself divided into three separate Perl

[1] So called because it was developed at the Institut für Maschinelle Sprachverarbeitung (IMS) at the University of Stuttgart and later released as open-source software.

[2] See http://www.lexically.net/wordsmith/.

[3] See http://www.antlab.sci.waseda.ac.jp/software.html#antconc.

packages, namely CWB,[4] CWB-CL and CWB-Web; and (iii) CQPweb, the most recent addition, which we will discuss in section 3.5. It is the CWB core itself which most users think of as "Corpus Workbench", and it is this core system that we will mostly focus on in this paper.

The CWB core has a number of strengths as a corpus analysis tool. Notable among them is its status as a tool of long standing; its original implementation was created in the early 1990s (see Christ 1994) and it has been in continual widespread use, and under continual development, since then (see Evert 2008). Of course, this is equally true of some other concordancers, for example WordSmith. But among the most powerful expert-level systems for corpus analysis, CWB is the best-established; its long persistence and wide adoption, as other software has come and gone, stands testament to its utility. A side effect of CWB's staying power is that certain aspects of its design – most particularly, the input format for data to be indexed, and the query language it implements – have become *de facto* standards in the field, largely by virtue of being implemented in other analysis tools. For instance, the Manatee software (Rychlý 2007),[5] used as a back-end by the SketchEngine interface (Kilgarriff et al. 2004), adopts CWB's input format and query language, and the Poliqarp system[6] implements a similar (but not identical) query language alongside an alternative input format. Other than string-level regular expression syntax itself, we are not aware of any other corpus query language which has been adopted across multiple concordancers in this way. Another aspect of the *de facto* standardisation of CWB's query language is that it has been adopted by some authors as a means of expressing precisely in reports of their research the nature of the corpus queries on which their analysis is based (e.g. Lee 2000, Mohamed 2011, Evert 2006).

Other than its long and influential history, CWB's primary virtue is that it can function *both* as a server back-end for large-scale, centralised client/server architectures *and* as a tool that an individual expert analyst can download, install and use on their own personal computer. This is possible because CWB's basic hardware requirements are relatively modest – though on a more powerful computer, larger corpora can be handled more quickly. The analyst using CQP on their own computer can employ either a command-line interface or an HTML-based graphical user interface (GUI). When CWB is used as a back-end for a multi-user architecture, of course, the type of interface is at the discretion of the system administrator, but an HTML-based GUI is most typical.

Many linguists have encountered CWB not by using the system directly, but rather via one or more public web-interfaces whose server side is based wholly or partly on CWB. This reflects CWB's nature as a very powerful tool with restricted user-friendliness: a web-interface can provide the accessibility that direct access to CWB does not (but see section 3.5 below). Perhaps the most widely-used interface based on CWB is BNCweb (Hoffmann et al. 2008).[7] This interface to the British National Corpus[8] was originally a front-end for a SARA server (see Aston and Burnard 1998), but was rewritten as a CWB application to benefit from

[4] The basic package CWB contains several different Perl modules (CWB, CWB::CEQL, CWB::CQP and CWB::Encoder), plus three shell scripts that facilitate corpus encoding (cwb-make, cwb-regedit and cwb-align-import)..

[5] See http://nlp.fi.muni.cz/trac/noske.

[6] See http://poliqarp.sourceforge.net.

[7] See also http://bncweb.info. Accounts for Lancaster University's BNCweb server is available via http://bncweb.lancs.ac.uk/bncwebSignup.

[8] See http://www.natcorp.ox.ac.uk/.

the greater speed and expressiveness of CQP queries (see Hoffmann and Evert 2006). In BNCweb, query processing is done by running CQP as a slave process on the web server; additional functionality, such as calculation of collocations and query sorting, is accomplished using a MySQL relational database.

However, BNCweb is far from the only online system to use a CWB back-end. Other examples include the "Leeds CQP" open interface to a collection of large English, Russian and Chinese corpora,[9] its recently developed successor IntelliText,[10] and the online interface of the VISL project.[11] Similarly the OPUS parallel corpus project (Tiedemann 2009) presents a web-based interface that exposes CWB indexes of various OPUS datasets,[12] exploiting CWB's support for sentence- or chunk-level alignment across two or more corpora. Each of these interfaces differs from the others in the structure of the graphical interface and in the details of how the advanced features of CWB/CQP are made available and accessible to the end user.

2.2. How does CWB work?

Three things are key to an understanding of how CWB works. The first is the overall architecture of the system; the second is the data model used to index CWB corpora; and the third is the syntax of the CQP query language and the very wide range of queries it allows.

2.2.1. Architecture of CWB

CWB's architecture is based around a set of encoded and indexed corpora, and two sets of methods by which they can be accessed. This architecture is shown in figure 1. In CWB, the *corpus* is the fundamental unit of data (rather than the file or text). Of course, there need not be a one-to-one match between a "corpus" in CWB terms and what researchers call a "corpus": an indexed dataset in CWB might be generated from a subcorpus or a group of corpora. Within the CWB system, a corpus exists as a set of binary data files that represent the indexed text as well as its linguistic annotations and structural markup and that are stored in a given location on disk. Housekeeping information about the corpus (including the disk directory and a complete data description) is collected in a single *registry file*. Registry files for all the indexed corpora on a given system are normally held in single directory, called the *corpus registry*, so users can access corpora simply by their *CWB name*. Low-level access to the indexed corpus data is provided by a set of functions called the *Corpus Library* or CL, based on the data description in the registry file. The CL is designed in such a way as to be usable independently of the rest of CWB, so that other applications can use its API[13] and do not need to understand the binary format of CWB data files. However, the most typical way of accessing the CL functions is via one of two systems: the CWB utilities and CQP. The *CWB utilities* are a set of command-line programs for manipulating indexed corpora. These include facilities for creating an indexed corpus, compressing it, decoding it to its original textual form, viewing its lexicon (i.e. list of word-types, POS tags, etc.), and compiling and printing basic frequency and co-occurrence statistics. CQP, the *Corpus Query Processor*, is a

[9] See http://corpus.leeds.ac.uk/.
[10] See http://corpus.leeds.ac.uk/itweb/ for the system and http://corpus.leeds.ac.uk/it/ for project information. See also Wilson et al. (2010).
[11] See http://beta.visl.sdu.dk/visl/corpus.html.
[12] See http://opus.lingfil.uu.se/
[13] *Application programming interface*: a formalism or set of formalisms by which a program can access functionality contained within another software component.

concordancer which builds on the CL to perform queries against an indexed corpus, using a query language that we will describe in section 2.2.3 below. While the CWB utilities are invoked and controlled via command-line options, CQP is run interactively in a terminal window (or as a slave process controlled programmatically by some other piece of software).

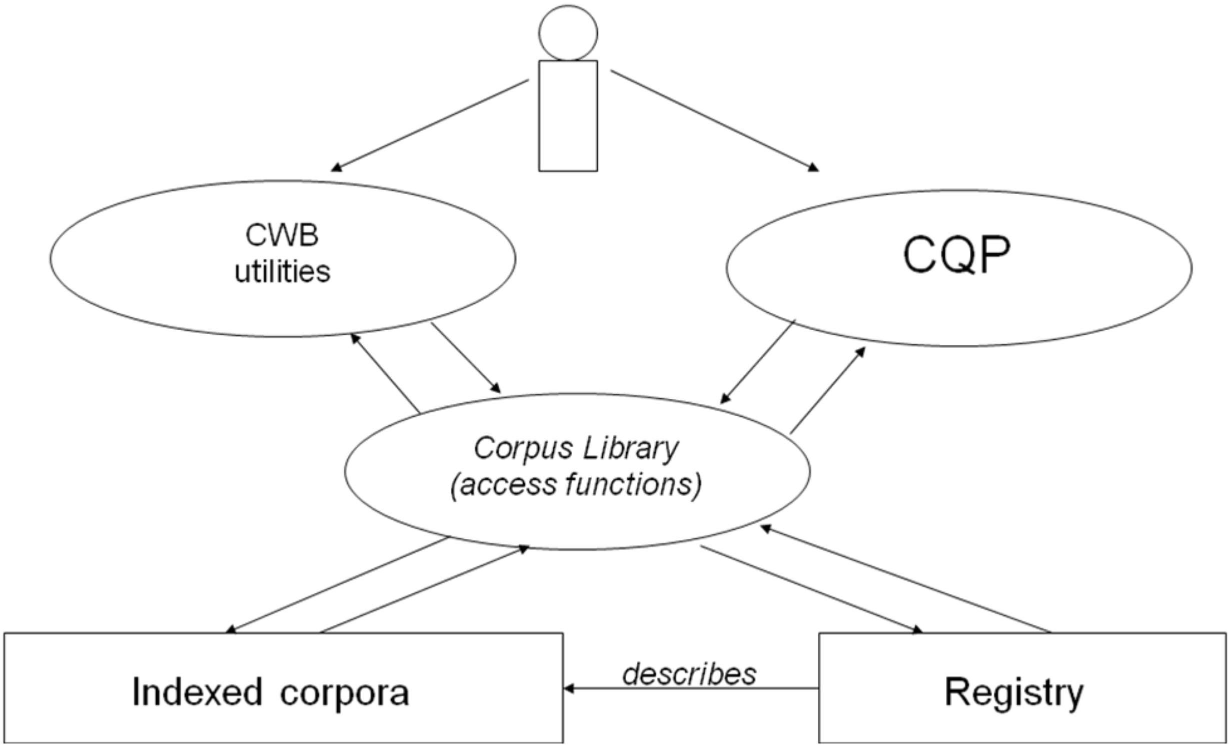


Figure 1. Overall architecture of CWB.

When CWB is used as a back-end system, either to the project's own CWB/Perl or CQPweb packages, or to some external system (such as BNCweb or IntelliText), the architecture is fundamentally similar, except that it is the front-end software which interacts with the CWB utilities and CQP, and the end-user interacts with the front-end. Many front-ends, including CQPweb and BNCweb, use CWB in conjunction with other back-end systems that provide additional functionality, such as a relational database management system (as used by BNCweb/CQPweb to support collocation analysis and other query postprocessing, for instance).

2.2.2. The CWB data model

The CWB data model – that is, the set of concepts and data categories that underlies all CWB indexing and querying – is realised in the format that corpus files must have for the CWB utilities to be able to index them. This is a columnar format similar to a database table, where each (tab-delimited) column represents a particular word-level annotation (including the word form itself), and each row represents a single token (word or punctuation mark) in the corpus. Consider, for instance, the following input data:

```
<sentence>
It         PP         it
was        VBD        be
<np head="elephant">
an         DT         a
elephant   NN         elephant
</np>
.          SENT       .
</sentence>
```

Every CWB corpus is stored internally as a stream of *tokens*. CWB cannot work with untokenised text, so tokenisation can be considered the *minimum* amount of annotation required for a corpus to be indexed into CWB; in practice, most users will find it easiest to rely on the tokenisation provided by a part-of-speech tagger such as CLAWS or the TreeTagger. In the input data format, each token is placed on a separate line; punctuation marks are treated as separate tokens. When the data is indexed, each token is assigned a *corpus position* number,[14] beginning with 0; so in the example above the tokens *It*, *was*, *an*, *elephant* and . will be assigned the corpus positions 0, 1, 2, 3, and 4 respectively. Note that XML tags such as `<sentence>` are not considered as part of the token stream, so that the tokens *was* and *an* are adjacent despite the intervening tag.

The centrality of the *corpus position* numbers to the internal workings of CWB is one of the main features that distinguishes it from corpus query systems based on relational databases (see Davies 2005, 2009, 2010). In a relational database table that stores a corpus, individual records (tokens) are in principle unordered, although it is always possible to explicitly add a numeric field that represents the sequence of the records. But in CWB, the sequencing is automatic and implicit, since the system is designed from the lowest level up to address the needs of describing and querying the stream of sequential token positions in a linguistic dataset.

CWB models an indexed corpus as a group of *attributes* related to the corpus position numbers. There are several different types of attribute. The most straightforward and most central is the *positional attribute* or *p-attribute*. A p-attribute represents an annotation at token level, i.e. any aspect of the corpus where each and every corpus position is linked to a single value. The word tokens themselves form a p-attribute – each corpus position is linked to the word form found at that particular position. Additional p-attributes can be created for whatever other token-level annotation is available in a given corpus, such as phonemic or phonetic transcription, morphological annotation, part-of-speech tagging, lemmatisation, or semantic tagging.

In the input file format, each p-attribute is written in a single tab-delimited column, where every row in the file that represents a token must contain a value in every column. The example data above has three p-attributes: the compulsory word form tokens, a part-of-speech tag, and a lemma. Each p-attribute is assigned a *handle* – a label used to refer to that p-attribute when manipulating or querying the CWB index. These handles are not contained within the input file itself. The handle for the compulsory word form tokens is always *word*. Other handles are specified by the user when indexing the data, and are then listed in the registry file; typical handles for the other attributes in the example above would be *pos* and *lemma*. For example, at corpus position 1, attribute *word* has the value *was*, attribute *pos* has

[14] The term *corpus position* is often abbreviated as *cpos* in the CWB documentation.

the value *VBD*, and attribute *lemma* has the value *be*. CWB stores every annotation value as a plain, unstructured string. However, thanks to CWB's indexing and compression algorithms as well as some special support in the query language (section 2.2.3), it is possible to work efficiently with multi-valued annotation (e.g. ambiguous part-of-speech tags), integer numbers, binary features (*yes*/*no*) and classification attributes (e.g. the 11 "simple" parts of speech annotated in the British National Corpus).

The other types of attributes are *structural attributes* and *alignment attributes*. Structural attributes (*s-attributes*) are associated not with single tokens, but rather with uninterrupted *token ranges* (represented internally as *pairs* of corpus positions: a start point and an end point). They are thus equivalent to XML elements and represent any kind of corpus annotation which applies to a region of text greater than a single token. Typical examples are sentences, paragraphs, and utterances; when text boundaries are included in a corpus index, these are also specified as s-attributes. It is also possible, however, for constituency parsing to be represented as sets of s-attributes, where each s-attribute represents a particular type of phrase and holds the start and end points of all instances of this phrase type (e.g. the noun phrase *an elephant* in the example above). Because of the close correspondence between s-attributes and XML elements, XML-style start and end tags are used to represent s-attributes in the input format, although CWB *is not* and does not aim to be a complete XML parsing/indexing system. Each XML start or end tag is placed on a separate line in the input data. These do not count as tokens, and do not have their own corpus position numbers. In the example above, the `<sentence>` tag is linked to the following token (indicating the start of a sentence at corpus position 0) and the `</sentence>` tag is linked to the preceding token (indicating the end of a sentence at corpus position 4). The handle of an s-attribute is the same as the XML tag that represents it – so the handle corresponding to the `<sentence>` tags is *sentence*. XML-style attribute-value pairs (such as `head="elephant"` in the example above) can also be indexed; in the CWB data model they are represented as subsidiary s-attributes. For instance, from the XML tags in the example above, in addition to the s-attribute *np*, another s-attribute *np_head* is created. Each instance of *np_head* has the same start and end points as the corresponding instance of *np*, but it also has an associated value (in this case *elephant*).

Given a corpus such as the British National Corpus, which is encoded as XML according to the TEI standard,[15] there is clearly a close correspondence between the XML mark-up of the files as distributed and the columnar input format required for CWB. The main difference is that in true XML, whitespace is nearly always irrelevant to the structure of the data, but in columnar format whitespace characters are used to indicate tokenisation and to delimit word-level attributes – which leads to (a) the requirement for each XML element to occur on a line on its own and (b) the requirement for tabs to indicate attributes such as part-of-speech or lemma. So for an XML-encoded token in the BNC such as this:

```
<w c5="VVN" hw="buy" pos="VERB">bought</w>
```

a line in columnar format such as the following must be generated:

```
bought     VVN        buy       VERB
```

[15] See http://www.tei-c.org/index.xml.

A more subtle discrepancy between true XML and CWB's input format is that CWB has no notion of a *corpus file header* providing meta-information about a text document. An XML file header may incorporate elements that themselves have textual content, such as the following standard component of British National Corpus headers:

```
<editionStmt>
     <edition>BNC XML Edition, December 2006</edition>
</editionStmt>
```

Such a header must be recoded into XML elements that enclose the entire text document, making sure that meta-information occurs only as XML attribute-values and not as textual content (since textual content could not be distinguished from the actual words of the corpus). For technically-aware users this is not especially troublesome, as it easily accomplished using XSLT or simple scripting techniques. Specialised import tools are provided for some widely-used corpora such as the British National Corpus. Other corpora are directly available in the CWB input format, e.g. the web corpora compiled by the WaCky initiative (Baroni et al. 2009).[16]

The final type of attribute is the *alignment attribute* or *a-attribute*, mainly intended for sentence-level alignment of translated text. An a-attribute encodes a relationship between text regions in two corpora that represent the same dataset in two different languages (i.e. they make up a *parallel* corpus in the sense that term is used by Baker 1993: 24, McEnery and Wilson 1996: 57, and Hunston 2002: 15). When an a-attribute is defined for a pair of corpora, it is possible to perform a search in one language, and retrieve not only a set of results in that language, but also the section of the corpus in the other language that translates each of the results in that set. We will not discuss a-attributes further here, although they are a key component of CWB-based interfaces to multilingual corpus data such as that of the OPUS project (see section 2.1 above).

2.2.3. The syntax of the CQP Query Language

The third key aspect to how CWB works is the query language used within CQP, which we call the *CQP Query Language* or *CQP-syntax* for short. This language is sometimes referred to in the literature as *Corpus Query Language* or CQL (for example, in papers related to the SketchEngine software). We do not use this term, however, and we strongly recommend against its use by others, because "CQL" has been in use for a long time as the name of a non-corpus query language standard promulgated by the US Library of Congress,[17] as well as a label for the underlying query language of the SARA/Xaira software.[18] This latter "CQL" is XML-based and therefore incompatible with CQP-syntax.

The main distinguishing feature of CQP-syntax is that it employs regular expression patterns at two levels: at the level of character strings for word forms and annotation values, and also at the level of token sequences. This makes CQP-syntax extremely flexible, and, as noted above, it has over time come to approach the status of a *de facto* standard.

In a CQP-syntax query, a token is specified as a set of one or more conditions contained within a single pair of [square brackets]. Multiple conditions may be linked by Boolean

[16] See http://wacky.sslmit.unibo.it/doku.php?id=corpora.
[17] See http://www.loc.gov/standards/sru/specs/cql.html.
[18] See http://www.oucs.ox.ac.uk/rts/xaira/.

operators such as `&` (and) and `|` (or). An individual condition usually consists of a regular expression that will be matched against a specified p-attribute, e.g. `[word = ".*phant"]` for a word ending in *-phant*; regular expressions may be matched normally, in case-insensitive mode (`%c`), in accent-insensitive mode (`%d`), or with both these modes active. In order to include plural forms such as *elephants* or *oliphants* in the search, and simultaneously to avoid matching the adjective *triumphant*, the token specification above might be rephrased as a double condition: `[lemma = ".*phant" & pos = "N.*"]`.

Regular expression syntax can also be applied to a sequence of token specifications in order to search for complex lexico-grammatical patterns. Just as, in a normal regular expression, individual characters may be concatenated in literal sequences, combined with repetition operators, or collected in sets of alternatives, so the same devices can be used at the level of tokens in CQP-syntax. For example, appending + to a token specification will match a sequence of one or more tokens that fit that specification. Token-sequence regular expressions may also make reference to s-attributes in the form of XML tags, which will match the start or end of an appropriate text region. For a comprehensive introduction to CQP-syntax, its use of regular expressions and more sophisticated query options, see Hoffmann et al. (2008: 217-245) and the CQP Query Language Tutorial.[19]

The following query exemplifies some of the features mentioned here – regular expressions matching strings at token level, the use of Boolean operators to combine conditions on a single token, and the construction of a regular expression across a sequence of tokens:

```
[word="queries" %c] [class="PREP"] [pos="AJ.*"]+
     [class="SUBST" & word=".*s"]
```

This query has the following matches in the British National Corpus:[20]

```
queries on old files
queries about albino Oscars
queries to multiple databases
queries on full-text contents
queries to multiple databases
queries from multiple sources
queries on full-text contents
queries on security-related topics
queries on individual services
queries on individual cases
queries about particular legal points
queries to Burning Questions
```

As the results make clear, the import of the original query string is “match all occurrences of the word form *queries*, followed by any preposition, followed by one or more adjectives, followed by any noun that ends in the letter *-s*”. In other words, this is a search for *queries* postmodified by a preposition phrase whose complement noun phrase is determinerless, is

[19] Available from http://cwb.sourceforge.net/documentation.php.

[20] The p-attribute *class* contains the “simple” tags annotated in the XML edition of the British National Corpus, which represent a generalisation across the underlying CLAWS5 part-of-speech tags. We refer to this system of simplified word-classes as the *Oxford Simplified Tagset* in recognition of its origin at Oxford University Computing Services; see http://www.natcorp.ox.ac.uk/docs/URG/codes.html#klettpos.

premodified by adjectives, and has plural number.[21] The query pattern says nothing about postmodification of the complement noun, so this part of the noun phrase is not captured (but is visible, where present, in the right-hand context of the matches when they are printed in CQP). Of course, token-level regular expressions based on the features of individual words can never be fully syntactically accurate. For instance, examination of the context of the final match above shows that in this case, the *to*-phrase following *queries* is not a postmodifier of *queries* but is rather a verbal argument: *Send your queries to Burning Questions, Echoes, The Northern Echo* .... Nevertheless, despite these limitations a carefully designed CQP query can often get an analyst most of the way to a clean and precise set of matches for exactly what they wanted to find.[22]

A full understanding of these technical aspects of CWB – the architecture, the data model, and CQP-syntax – is clearly not necessary for basic-level use of CQP. However, effective use of the indexing tools and the more powerful features of CQP does require an extensive grasp of these issues. It is for this reason that we earlier characterised CWB as expert-level software. These features of how CWB works are responsible for its main strengths as an analysis tool and do not require any serious improvement at present. In the remainder of this paper, we will consider points where, by contrast, the original design of CWB *has* required updating to meet the requirements of a contemporary corpus indexing and search system.

3. Twenty-first century requirements for CWB

As noted above, one of the strengths of CWB is its staying power: it has been in use, and under development, since the early 1990s. Yet this strength is at the same time a weakness, because many features of the system's design reflect decisions that were reasonable in the environment of 1994, but are no longer reasonable in the twenty-first century, given changes in development best practices, hardware and operating system capabilities, and not least user expectations. Most of the work that has been undertaken for versions 3.0 through 3.5 of CWB is directed towards fulfilling these new requirements. This section outlines work in five key areas that have brought CWB up-to-date with the needs of contemporary analysts. Version 3.5 of CWB will be a stable release with long-term support, incorporating all of the improvements described below.

3.1. The development process

For a project of the scope of CWB to be run by, and the resulting software to be available from, a single institution (IMS Stuttgart) was not unusual in the early 1990s. This model for a development process had, in fact, been established since the 1970s as best practice for highly-powerful corpus tools in what McEnery and Hardie (2012: 37) dub the *first generation* of concordancing software. However, in the period since the original versions of CWB, a new set of best practices for non-commercial software has emerged, under the influence of such prominent projects as the Linux kernel and the Mozilla web browser: the model of public development of a code base with an open-source (or *free software*) licence. When CWB moved away from being solely an internal project of the IMS to being a wider community

[21] Note the use of a regular expression to match the CLAWS5 part-of-speech tags *AJ0*, *AJC* and *AJS* for adjectives (including various portmanteau tags), since the simple tag *ADJ* includes predeterminers such as *any* and *all* in addition to regular adjectives.

[22] CQP allows result-sets to be *dumped* (extracted from CWB), modified outside the system, and then *undumped* (reinserted into CWB), so that an imprecise set of hits can be made more (or fully) precise by manual adjustment. This functionality is also exposed by some web interfaces, most notably BNCweb and CQPweb.

project with a much broader user-base, it simultaneously transitioned from its closed-source origins to running on an open-source basis. The CWB code is thus now licensed under the GNU General Public Licence.[23] The formal name of the software – and the community project that supports it – is therefore now *The IMS Open Corpus Workbench*, although the established habit of referring to it as simply *(the) Corpus Workbench* or *CWB* is not expected to change.

Moving to a licence such as the GPL has two distinct advantages for development. First, widely-used open-source libraries can be leveraged to add functionality to the system with minimal effort by the CWB developers. An example of this is the GNU Readline library,[24] on which CQP's interactive command-line editing features are based. Second, it becomes possible for other projects to fork the code base and take their version of CWB in other directions depending on their needs, and for CWB to absorb and reintegrate useful innovations in these forks, without any risk of intellectual property tangles. An example of this happening in practice will be discussed below.

Since moving to an open-source model, the development of CWB is conducted *in public*. This means much more than simply releasing the code or software builds under a particular licence. It means that the project's database of bugs and feature requests is open to all to read and contribute,[25] as are roadmap documents planning out current and future work.[26] Moreover, the CWB team's discussions regarding the development and direction of the software are also public, being conducted via a mailing list which anyone can join and whose archives are visible to all.[27] This mode of operation takes us some way towards Anthony's (2009) vision of an open-sourced, interactive development process for corpus tools.

3.2. Operating system support

The original version of CWB was designed to run only on a commercial version of Unix, and was distributed in binary form. This was a reasonable design decision in the 1990s, when most systems with powerful enough hardware to support analysis of large corpora in CWB were Unix servers rather than personal computers and when the open-source Unixes were only beginning to acquire popularity. Indeed, Linux itself is only slightly older than CWB. However, since around 2000 it has been much less acceptable for a corpus analysis system to be restricted to *any* single operating system. Personal computers, whether Windows or Linux PC or Apple Macintosh, are now more than adequately powerful for CWB installation on such machines to be a commonplace requirement. Moreover, the ubiquity of highly standardised Unix-like operating system environments – notably Linux, Mac OS X, the BSD variants, and Cygwin on Microsoft Windows – means that CWB compatibility with all the widely-used operating systems is much more feasible than it would have been in 1994. It has been possible to build and install CWB on these different Unix-like systems in preview versions since the early 2000s, and in the stable CWB version 3.0 since 2010. Indeed, Linux and Mac OS X are now the project's main development platforms.

[23] See http://www.gnu.org/licenses/gpl.txt, or the file COPYING within the CWB code distribution.
[24] See http://www.gnu.org/s/readline.
[25] At http://sourceforge.net/tracker/?group_id=131809.
[26] See, for instance, http://cwb.sourceforge.net/future.php; http://cwb.svn.sourceforge.net/viewvc/cwb/cwb/trunk/doc/todo-3.5 .
[27] To join the list or access the archives, see http://devel.sslmit.unibo.it/mailman/listinfo/cwb.

Direct compatibility with Windows, *without* the use of the Cygwin environment, was a longstanding (and very reasonable) demand made by the CWB user community. This was implemented in version 3.1 by integrating compatibility changes made in the fork of the CWB code base that forms part of the TXM platform (Heiden 2010).[28] As noted above, it is a benefit of the open model of development that advances made by others can be incorporated into CWB without problems of intellectual property arising; both CWB and TXM are issued under the GNU General Public Licence. The Windows version of CWB is built in a Linux environment using the MinGW cross-compiler version of GCC;[29] the resulting software can then be deployed to Windows machines without the end user having to set up compilation tools on Windows. The use of MinGW means that the changes required for Windows compatibility are much less than they would otherwise be, because Windows as seen by CWB through MinGW is, in effect, another Unix-like environment.

Since the Windows compatibility is a comparatively new development, some bugs do remain; working these out is a primary aim for the forthcoming version 3.4 of CWB. It is also worth noting that the CWB/Perl and CQPweb software packages (see sections 2.1 and 3.5) are not yet completely Windows-compatible; bringing their compatibility up-to-date with that of the CWB core is likewise a key short-term goal of the CWB project.

3.3. 32-bit versus 64-bit environments

When running under a 32-bit operating system, CWB is limited to a maximum corpus size of around 200 to 500 million words, or less if there is a very large amount of annotation. By 1990s standards, this was a very reasonable limitation, given the amounts of disk space and computer memory available on most machines at that time. The British National Corpus, at that time the epitome of a large corpus of contemporary English, is only 100 million words in extent; the larger but less widely-used Bank of English consists of around 500 million words. So a 32-bit version of CWB is more than capable of dealing with the very large corpora that were in use at the time it was designed, and for most of the following decade.

In the period between 2000 and 2010, however, 64-bit operating systems have entered fairly wide use. At the same time, much larger corpora have become available. For example, the English Gigaword corpus[30] (not much used by corpus linguists, but of great interest for computational linguists and NLP specialists) is 1.8 *billion* words in size, an order of magnitude larger than the British National Corpus. Likewise, corpora derived from the web are often of this kind of size. The ukWaC corpus (Baroni et al. 2009), for example, contains 2 billion words, and the WaCky initiative has compiled corpora of comparable size in several other languages as well.[31] Thus, for CWB to continue to work under the restrictions of a 32-bit application would make it incapable of dealing with the kinds of datasets that analysts want to work with today. Therefore, as a part of making CWB fully compatible with additional operating systems (see section 3.2), work has been undertaken to ensure that CWB can be compiled and run under 64-bit operating systems.[32] These 64-bit versions of the

[28] TXM was created by the *Textométrie* project at the Ecole normale supérieure de Lyon; see http://textometrie.ens-lyon.fr/?lang=en.

[29] See http://www.mingw.org.

[30] Distributed by the Linguistic Data Consortium, see http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2003T05

[31] See http://wacky.sslmit.unibo.it/doku.php?id=corpora.

[32] Note that support for 64-bit features in Windows is uncertain in versions prior to the forthcoming CWB version 3.5.

software are capable of dealing with richly annotated corpora of up to 2.1 billion words. This increase, by a factor of just over 4, is much less than what is theoretically permitted by a 64-bit data type; however, the internal format of the indexed and compressed corpora imposes the 2.1 billion token limit.

3.4. Character encoding

For several years now, the Unicode Standard (Unicode Consortium 2006)[33] has been established as the preferred approach to character encoding, supporting as it does text in all the world's languages. In particular, the UTF-8 version of Unicode has long been the recommended encoding for corpus data (McEnery and Xiao 2005) – and since any valid ASCII text is also valid UTF-8, this means that older corpora that use ASCII can be processed as UTF-8 without any adjustment. So UTF-8 support is a high priority for any modern corpus tool that aspires to be truly multilingual in scope.

When CWB was first designed, Unicode did exist; its origins go back to the 1980s. However, hardly any widely-used software implemented Unicode in the early 1990s, and as such most textual data was not encoded as Unicode. Nor, however, was plain ASCII text commonly used. Instead, most textual data was stored using forms of the ISO 8859 standard. In keeping with this environment, CWB was originally designed to work only with ISO-8859-1, the Western European character set, also known as Latin1, which supplements ASCII with the accented characters required by German, French, Spanish, Italian, and a few neighbouring languages. All CWB's internal string handling functions – such as the code tables for case-insensitivity and accent-insensitivity – were designed around the ISO-8859-1 standard.

This design decision has persisted up to the first stable open-source version 3.0 and into the earliest Windows builds in version 3.1. However, while it was an entirely reasonable strategy in 1994, in the twenty-first century full support for Unicode is a critical requirement in any corpus analysis tool. Indeed, many CWB-based applications (such as early versions of CQPweb) achieved multilingual support by indexing UTF-8 data *as if it were* ISO-8859-1 data. This solution was, however, wholly unsatisfactory, since (a) ISO-8859-1 case and accent folding will not work with UTF-8 data and is in fact liable to scramble such data badly, and (b) the POSIX regular expressions used by the query engine cannot recognise UTF-8 multi-byte sequences as single characters (so a single character in the Greek or Arabic sections of Unicode would match, for instance, *two* dot characters in a regular expression, instead of one). All in all, proper UTF-8 support was a clear priority.

The development of CWB's Unicode support was contributed by the Textométrie project, which as noted above also contributed Windows compatibility patches. As a result, versions 3.2 and above have full compatibility with UTF-8. CWB takes account of Unicode in two main ways. Firstly, all the case-folding, accent-folding and other low-level string manipulation functions are UTF-8-aware. This has been accomplished by using Unicode support functions from the open-source GLib library[34] within CWB's low-level CL. Secondly, the string-level regular expression engine has been changed from POSIX regular expressions to the PCRE (*Perl-Compatible Regular Expressions*) library,[35] which like GLib is a widely-used open-source project. Among other advantages, such as the access it provides to a very popular and powerful variant of the regular expression syntax (that found in the Perl

[33] See http://unicode.org/.

[34] See http://developer.gnome.org/glib/.

[35] See http://www.pcre.org/.

scripting language), PCRE offers support for UTF-8 regular expressions. Thirdly, the indexing utilities have been amended so that each indexed corpus is associated with a particular character set (declared at indexing time), and – critically – so that all data imported into CWB is checked for validity in the character set it has been encoded as. This means that all data *inside* a CWB index is known-valid as ASCII or ISO-8859-1 or UTF-8, as the case may be, since indexing fails if bytes or byte sequences are detected in the input data that are not valid for the declared character encoding.

Unicode brings with it significant advantages – not least among them the availability of open-source libraries that can be leveraged, as noted above. However, it does have some countervailing disadvantages. It is, in general, a bulkier format than ISO-8859; for example, UTF-8 text in Greek or Arabic requires twice as much disk space as the same text in ISO-8859-7 or ISO-8859-6 respectively (pre-Unicode Greek and Arabic character sets respectively). Given the historically low and falling price per megabyte of hard disk space, and even of computer memory, this is unlikely to be a major problem for most corpora – but for a very large corpus the size differential might be an important factor.[36] This constitutes, then, a reason why even in the age of widespread Unicode support, a user might prefer to work with ISO-8859 data. This is not the only reason – quite simply, some users may be reluctant to undertake the potentially rather large task of updating the character encoding of all their existing indexed data. Since CWB corpora by and large cannot be modified (only deleted and then re-indexed from input text), this is not a trivial proposition. To address these concerns, we implemented support for the ISO-8859 legacy encodings alongside UTF-8. The Corpus Library in the CWB core now contains data tables for validity-checking, case-folding and accent-folding each of these character sets. Furthermore, since the PCRE library's Unicode mode can be turned on and off, it is simple to use this regular expression engine with both UTF-8 and ISO-8859 strings.

3.5. The user interface

It has been noted above that CWB is *not* intended as a tool for beginners in corpus analysis. Corpora are managed via the command-line utilities, requiring a knowledge of the Unix option/argument conventions, and interactive-mode CQP also requires knowledge not only of the (complex) query language but also of the control syntax for its own configuration. CQP is fairly unforgiving of mistakes, and there is in addition little in the way of online help. The lack of a user-friendly interface was a reasonable design decision in the early 1990s. It is not reasonable any longer, and represents CWB/CQP's greatest limitation.

Several efforts have been made to address this limitation. One early effort, the Xkwic GUI, proved inflexible and has been abandoned. In CWB version 3.0, a more adaptable approach has been taken by the provision of the CWB/Perl component, which offers a programming interface by which bespoke web-based or command-line interfaces can be created with relative ease for different purposes and at different levels of complexity/user-friendliness. CWB/Perl is used by most CWB-based web front-ends. An important feature of CWB/Perl is the *Common Elementary Query Language* or CEQL. CEQL is a query formalism which gives access to the most commonly-used features of CQP-syntax in a simpler form. For example, regular expression syntax is replaced by a simpler set of wildcards, and rather than requiring an explicit specification of what p-attribute a regular expression is to be matched against,

[36] It should be noted that we are here discussing the size of the data in its original corpus format and in the CWB input format. The CWB index itself is condensed and compressed.

shortcut characters and an assumed default attribute are employed. So the following CQP-syntax query:

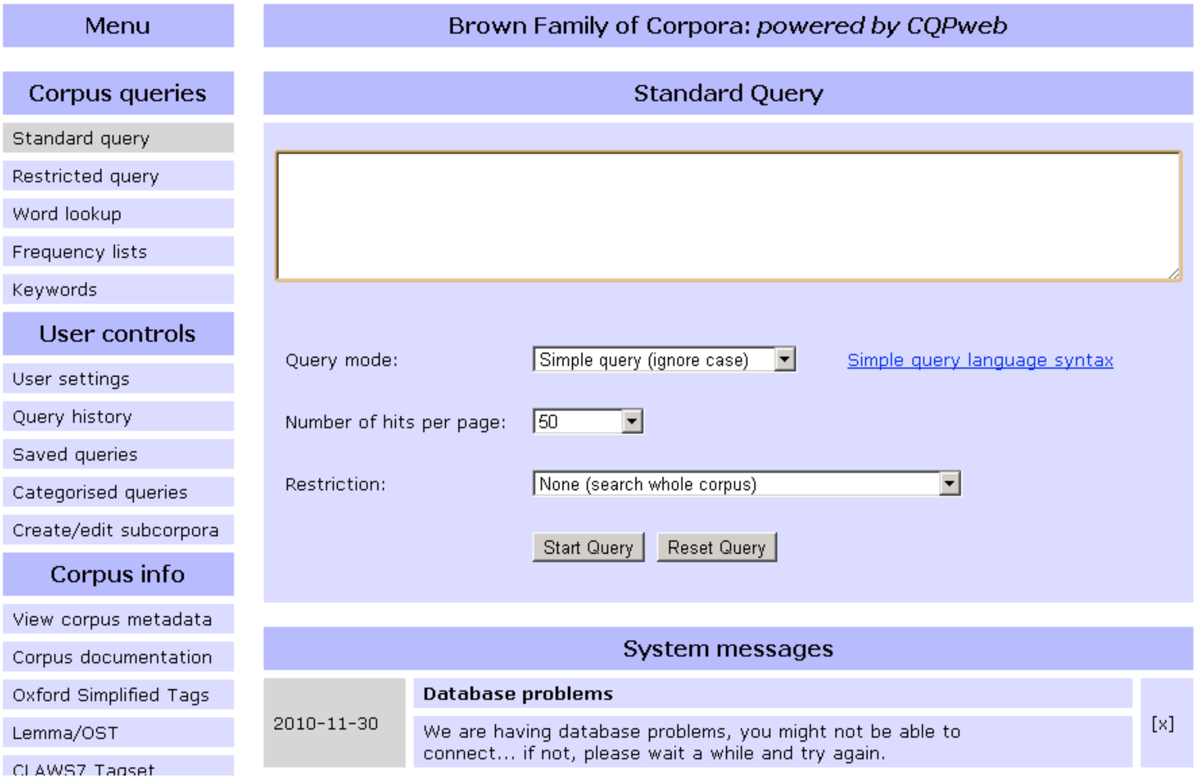
```
[lemma = "present" & pos = "V.*"] [word = "an"%c]
```

(i.e. any form of *present* tagged as a verb, followed by *an*) would in CEQL be expressed as

```
{present}_V* an
```

Note the lack of square brackets, the use of {curly braces} to match against lemma instead of word form, and the underscore separating lemma and POS tag. Offering a choice between CEQL and CQP-syntax in front-end software thus partially addresses CWB's limited user-friendliness. CEQL was designed in particular for use with corpora annotated after the style of the British National Corpus (and thus provides privileged access to part-of-speech tags, lemmata, and simple tags), and sees most extensive use within BNCweb. See Hoffmann et al. (2008, 93-117) for a complete description of CEQL syntax.

More recently CWB has acquired a new standard GUI in the form of CQPweb, which clones the BNCweb interface format and additional functions (collocations, frequency lists, keywords, subcorpus management, and so on; all implemented via the MySQL relational database) in such a way as to make it usable with any corpus, not just the British National Corpus (see figure 2). Like the most recent versions of the CWB core, CQPweb supports UTF-8 corpora throughout and is thus largely language-independent. It also allows CEQL to be configured to key into any form of annotation – that is, the "simple query" syntax is not restricted to corpora with part-of-speech, lemma and simple tag annotation.



Brown Family of Corpora: *powered by CQPweb*

Query mode: Simple query (ignore case)

Number of hits per page: 50

Restriction: None (search whole corpus)

2010-11-30

Database problems

We are having database problems, you might not be able to connect... if not, please wait a while and try again.

Figure 2. The welcome-page and main query interface of a corpus indexed in CQPweb

CQPweb is discussed extensively by Hardie (forthcoming) and we will not go into further detail here. The key point for current purposes is to note what CQPweb adds to CWB: a standard, corpus-independent graphical user interface suitable for use either locally on a single computer or over the web. The user-friendliness provided by CQPweb addresses one of the key requirements of a twenty-first century corpus analysis system.

4. The future of CWB

Some avenues of development for the CWB project in versions 4.0 and higher have already been mentioned in foregoing sections. Other obvious improvements constitute direct extensions of the developments made to date and discussed above. An example of this is the maximum corpus size. As outlined in section 3.3, we have raised this to approximately 2.1 billion tokens by allowing CWB to run under 64-bit operating systems. But this new limit merely “buys time”. The size of datasets that analysts wish to work with will only continue to grow. The ukWaC corpus, at just under 2 billion words, is already pushing the boundaries of the higher limit and cannot be indexed in its entirety as a single CWB corpus. So one clear need is to raise the maximum corpus size by further orders of magnitude, allowing datasets of not only arithmetically but geometrically greater extent to be analysed.

Yet other planned improvements reflect further shifts in requirements from the 1990s to the twenty-first century that have not yet been addressed. For example, in the 1990s there was a strong requirement for CQP concordances to be displayed and exported in a variety of different formats, including LaTeX, SGML, HTML and plain text. CQP does indeed support each of these, in a partly configurable manner. However, given the very wide availability nowadays of powerful tools for working with XML, a much more practical system will be for CQP to provide two formats only – plain text for interactive usage, and XML for other uses including, most particularly, data interchange – and to allow users to generate other concordance formats from the XML output using the tools of their preference. Rather than force users who want HTML concordances to work within the limits of CQP’s HTML output format, for instance, it would make more sense for users to generate HTML according to their requirements using XSLT and the XML output format. The CWB package could, in this case, provide a “default” example HTML-generating stylesheet suitable for novice users.

The same factors arising from the rapid growth in popularity and ubiquity of XML suggest that CWB clearly needs to support more extensive indexing and querying of XML elements beyond what the current implementation of s-attributes permits. For example, a more rigorous and efficient representation of XML attribute-value pairs than the current subsidiary s-attributes would be a clear improvement, as would support for recursive nesting of a given XML element within itself (currently only possible via further automatically generated subsidiary s-attributes, each representing a different level of embedding).

Beyond such issues, it is the CQP query engine itself that is the most obvious target for improvement, since any gain in power here has a direct pay-off for all users of CWB. A first step will necessarily be to rework some of the internals of CQP to make it more modular than is currently the case. However, once this is accomplished, adding new features to the engine will be vastly easier. For instance, one much-requested improvement is the capability for multiple target positions to be generated within query results. A *target position* is a specially marked token-slot within a CQP query for which statistics can be calculated, such as a frequency list of word types or annotations appearing in the marked slot. For instance, if a query is written to capture a particular type of noun phrase, and the token-slot representing the head noun is designated as the target position, then a frequency list of the nouns that occur as the head of that particular sort of noun phrase can be extracted. The capacity to specify multiple target positions will make this functionality much more powerful – making it possible, for instance, to extract the joint frequencies of all combinations of types found in two or more different target positions.

Similarly, there is scope for improvement in the speed of execution of complex queries. A known weakness of CQP is that multi-token query strings which begin with a token specification with very many matches are slow to execute. For instance, consider the following query (a fairly primitive approach to "find all instances of *happy* premodifying a definite noun"):[37]

```
    [word="the"%c] [pos="R.*"]{0,2} [word="happy"%c] [pos="N.*"]
```

CQP's internal query evaluation procedure is to convert the token-sequence regular expression into a finite state automaton. In consequence, the very first step in evaluating the query is to retrieve all corpus positions where the p-attribute *word* has the value *the*. This is a very large number of results (slightly above 6 million hits in the 100 million word BNC), most of which will been thrown out at later stages as the finite state automaton processes the rest of the query expression – because the overwhelming majority of those millions of instance will prove, on further examination, *not* to be followed by an instance of the word *happy* (after optional adverbs). The most efficient way of carrying out this query would be to start by retrieving all instances of the *third* token specification (i.e. `[word="happy"%c]`), which has the fewest hits. But the current implementation of CQP's query engine does not permit this.

This weakness in CQP is quite widely known; for instance, Davies (n.d.; see also Davies 2005) argues for the inferiority of CWB/CQP to a corpus search system that uses a relational database engine on the basis of evaluation times for three different multi-token queries, all three of which begin with a token specification that has a very high match rate in English: the article *the* (~6% of tokens), any pronoun (~8% of tokens), and any conjunction (~5% of tokens).[38] What is perhaps less widely appreciated is that this weakness in CQP is fundamentally a trade-off for the power of token-sequence regular expressions, which is rather harder to implement in a relational database query language such as SQL[39] – all Davies' example queries are fixed-length token sequences, without any optional tokens, optional repetitions, alternative sequences of different lengths, or any of the other regular expression features available in CQP-syntax. This trade-off produces, in our view, a more powerful and flexible system than one which performs faster but offers less scope for complex queries – especially since speed in query evaluation is at least as dependant on the available hardware as on the software. All that said, however, these kinds of query are clearly a point where significant scope exists for improvement in CQP, by amending the query evaluation procedure to start from a token specification with a small number of hits (which

[37] This query pattern assumes an English corpus with a CLAWS7-style part-of-speech tagset, where noun tags all begin with N, adjective tags with J, and adverb tags with R.

[38] Percentage figures estimated from the British National Corpus; the precise percentage depends, naturally, on the precise definition of the linguistically fuzzy categories of *pronoun* and *conjunction* embodied in the particular tagset in use.

[39] Davies (2005: 309) appears to assert that CQP searches are performed by generating SQL queries behind the scenes. This is not the case. CWB attribute indexes *can* be regarded as a form of relational database, but access to the indexes is not SQL-based; see also section 2.2.2 above.

can easily be retrieved from the CWB index files) and then to match the full query expression to the left and right of each hit (see Meurer 2010). In the example above, this strategy would first find all occurrences of the word *happy* and then check for each instance whether (i) it is preceded by optional adverbs and a definite article and (ii) it is followed by a noun.

Another promising avenue for future development is to offer different types of query within CQP-syntax. For instance, support for *fuzzy* queries of different sorts would be a useful extension to what CQP is currently capable of, especially when searching noisy texts such as web-corpora. It would also be desirable to support Boolean queries, as typically encountered in information retrieval and web search engines, not least because they are easier to optimise than standard CQP queries based on regular-expressions. An initial version of this capability already exists in the form of the undocumented *MU query* (meet-union), but this functionality needs to be extended and improved. With the rapid adoption of dependency parsers for large-scale corpus annotation,[40] there is increasing need to support queries based on relationships between words – as expressed in the graph of a dependency parsing analysis. It is already possible to index a constituency parse as a set of s-attributes, so a dependency parse represents the major form of corpus annotation not currently supported by CWB. Addressing this is likely to require the establishment of a new kind of attribute, the *dependency attribute* or *d-attribute* where each corpus position is associated with another corpus position (its parent in the dependency parse tree) and where each such relationship between corpus positions is classified into a type such as *subject-of*, *object-of*, *complement-of* and so on. It would of course also be possible for dependencies other than syntactic relations to be encoded as a d-attribute, such as the anaphoric or cataphoric links between a pronoun and a full noun phrase. A special type of query would have to be implemented in order to make efficient use of such annotation, by following dependency links from child node to parent node and *vice versa*.

For all these query features, both new search types and optimisation, however, the obvious benefit of enhancing the capabilities of CQP must be balanced against a certain risk inherent in making any changes to the CQP-syntax. As noted above, the CQP-syntax has acquired a *de facto* standard status, and is implemented by other software such as Manatee. Drastic changes to the query language within CQP could jeopardise this status. It would clearly be detrimental to the community of corpus linguists as a whole if CQP-syntax queries recorded in the published literature cannot be counted on to continue to be compatible with CWB/CQP – this is an issue which speaks to the *replicability* of results in the field, replicability being one key element of the scientific method. Equally undesirable would be a situation where several different tools all implement CQP-syntax, but where each implements a slightly different and marginally incompatible form of CQP-syntax. Therefore, substantial extensions should always be implemented as separate, clearly marked query types. The currently implemented (but undocumented) *meet-union* query system exemplifies how this works. One variant of meet-union query has the following form:

```
MU(meet word1 word2 s-attribute)
```

This query finds all instances of the specified s-attribute that contain both *word1* and *word2*. The special syntax, starting with a reserved keyword `MU`, ensures that such meet-union queries do not conflict with the more usual token-sequence type of query.

[40] For instance, the WaCky initiative offers dependency-parsed versions of their web-corpus ukWaC (“pukWaC”) and of the English Wikipedia (“Wackypedia”).

The roadmap for future development that has been sketched out in this section is based on the major needs of CWB as we currently perceive them to be. Ultimately, however, CWB is too extensive and widely-used a system for its direction to be determined solely by the ideas of two or three developers. Already-planned features that are particularly desired by the user community will be prioritised, and other requests for enhancements and new features will be adopted wherever possible. Current and future users are urged to contribute their feature requests to the project, as all input of this kind is extremely helpful in planning and prioritising our work on the system.[41]

5. Summary

In this paper, we have given an introduction to the main features of the IMS Open Corpus Workbench, explaining how it works and the main advantages it possesses (a) as an architecture particularly aimed at indexing and querying large corpora with relatively extensive token-level annotation and (b) as a back-end for user-friendly, especially web-based, concordancers. Subsequently, in sections 3 and 4, we have outlined recent developments to bring CWB versions 3.0 and above up-to-date with the requirements of a twenty-first century corpus analysis tool, and the further enhancements that we anticipate will be required in the medium-to-long term.

Readers who are interested in CWB – either the software itself or the community project supporting it – are referred to its website, hosted at SourceForge.net,[42] and to the email list hosted at the Università di Bologna.[43] We welcome queries, bug reports, feature requests and code contributions from both novice and experienced users.

References

Anthony, L. 2005. “AntConc: a learner and classroom friendly, multi-platform corpus analysis toolkit”, in *Proceedings of IWLeL 2004*: *An Interactive Workshop on Language e-Learning*, pp. 7–13. Tokyo: Waseda University.

Anthony, L. 2009. “Issues in the design and development of software tools for corpus studies: the case for collaboration”, in P. Baker (ed.) *Contemporary Corpus Linguistics*, pp. 87–104. London: Continuum.

Aston, G. and Burnard, L. 1998. *The BNC Handbook*: *Exploring the British National Corpus with SARA*. Edinburgh: Edinburgh University Press.

Baker, M. 1993. “Corpus linguistics and translation studies: implications and applications”, in M. Baker, G. Francis and E. Tognini-Bonelli (eds) *Text and Technology: in Honour of John Sinclair*, pp. 233–352. Amsterdam: John Benjamins.

[41] This applies to the CWB/Perl and CQPweb systems as well as the CWB core. CQPweb, in particular, does not yet have a finalised feature set and there is an extensive “wish-list” for future enhancements of its capabilities, quite separate to the developments of the core suggested here. See http://cwb.sourceforge.net/future.php#cqpweb.

[42] http://cwb.sourceforge.net or the shorter alternative http://cwb.sf.net .

[43] http://devel.sslmit.unibo.it/mailman/listinfo/cwb.

Baroni, M., Bernardini, S., Ferraresi, A. and Zanchetta, E. 2009. “The WaCky Wide Web: A collection of very large linguistically processed Web-crawled corpora”, *Language Resources and Evaluation* 43 (3): 209–226.

Christ, O. 1994. “A modular and flexible architecture for an integrated corpus query system”, in *Proceedings of COMPLEX '94*, pp. 23–32. Budapest. Available online at http://cwb.sourceforge.net/files/Christ1994.pdf

Davies, M. 2005. “The advantage of using relational databases for large corpora: speed, advanced queries and unlimited annotation”, *International Journal of Corpus Linguistics* 10 (3): 307–34.

Davies, M. 2009. “Word frequency in context: alternative architectures for examining related words, register variation and historical change”, in D. Archer (ed.) *What’s in a Word-list? Investigating Word Frequency and Keyword Extraction*, pp. 53–68. Farnham: Ashgate.

Davies, M. 2010. “More than a peephole: Using large and diverse online corpora”, *International Journal of Corpus Linguistics* 15 (3): 412–8.

Davies, M. (n.d.) Corpus.byu.edu site, “Architecture” page. Available on the internet at http://corpus.byu.edu/architecture.asp, accessed 6th August 2011.

Evert, S. 2006. “How random is a corpus? The library metaphor”, *Zeitschrift für Anglistik und Amerikanistik* 54(2): 177-190.

Evert, S. 2008. “Inside the IMS Corpus Workbench”. Presentation at the IULA, Universitat Pompeu Fabra, Barcelona, Spain. Available online at http://cwb.sourceforge.net/files/Evert2008_InsideCWB.pdf

Hardie, A. Forthcoming. “CQPweb – combining power, flexibility and usability in a corpus analysis tool”. Draft available online at http://www.lancs.ac.uk/staff/hardiea/cqpweb-paper.pdf

Heiden, S. 2010. “The TXM Platform: Building Open-Source Textual Analysis Software Compatible with the TEI Encoding Scheme”, in K. I. Ryo Otoguro (ed.), *24th Pacific Asia Conference on Language, Information and Computation*, pp. 389–398. Institute for Digital Enhancement of Cognitive Development, Waseda University, Sendai, Japan. Available online at http://halshs.archives-ouvertes.fr/halshs-00549764/en .

Hoffmann, S. and Evert, S. 2006. “BNCweb (CQP-Edition) – The Marriage of Two Corpus Tools”, in S. Braun, K. Kohn and J. Mukherjee (eds.) *Corpus Technology and Language Pedagogy: New Resources, New Tools, New Methods*, pp. 177-195. Frankfurt am Main: Peter Lang.

Hoffmann, S., Evert, S., Smith, N., Lee, D. and Berglund Prytz, Y. 2008. *Corpus Linguistics with BNCweb – a Practical Guide*. Frankfurt am Main: Peter Lang.

Hunston, S. 2002. *Corpora in Applied Linguistics*. Cambridge: Cambridge University Press.

Kilgarriff, A., Rychly, P., Smrz, P. and Tugwell, D. 2004. “The Sketch Engine”, in G. Williams and S. Vessier (eds) *Proceedings of EURALEX 2004*, pp. 105–116. Bretagne, France: Université de Bretagne-Sud.

Lee, D. Y. W. 2000. *Modelling variation in spoken and written language: the multi-dimensional approach revisited*. Unpublished PhD thesis, Lancaster University.

Meurer, P. 2010. *Corpuscle – a new search engine for large annotated corpora*. Technical report, Uni Digital, Uni Research AS, Bergen, Norway.

McEnery, T. and Hardie, A. 2012, in press. *Corpus Linguistics: Method, Theory and Practice*. Cambridge: Cambridge University Press.

McEnery, T. and Wilson, A. 1996 (second edition 2001). *Corpus Linguistics*. Edinburgh: Edinburgh University Press.

McEnery, T. and Xiao, R. 2005. “Character encoding in corpus construction” , in M. Wynne (ed.) *Developing Linguistic Corpora: A Guide to Good Practice*, pp. 47–58. Oxford: Oxbow Books.

Mohamed, G. 2011. *Text classification in the BNC using corpus and statistical methods*. Unpublished PhD thesis, Lancaster University.

Rychlý, P. 2007. “Manatee/Bonito – A Modular Corpus Manager”, in P. Sojka and A. Horák (eds.) *First Workshop on Recent Advances in Slavonic Natural Language Processing, RASLAN 2007*, pp. 65-70. Brno: Masaryk University Available online at http://nlp.fi.muni.cz/raslan/2007/papers/12.pdf.

Scott, M. 1996. *WordSmith Tools*. Oxford: Oxford University Press.

Tiedemann, J. 2009. “News from OPUS – A Collection of Multilingual Parallel Corpora with Tools and Interfaces”, in N. Nicolov, K. Bontcheva, G. Angelova and R. Mitkov (eds.) *Recent Advances in Natural Language Processing* (vol V), pp. 237-248. Amsterdam: John Benjamins.

Unicode Consortium. 2006. *The Unicode Standard, Version 5.0*. London: Addison-Wesley.

Wilson, J., Hartley, A., Sharoff, S. and Stephenson, P. 2010. “Advanced corpus solutions for humanities researchers”, in *Proceedings of PACLIC 24*, Sendai, Japan. Available online at http://corpus.leeds.ac.uk/serge/publications/2010-PACLIC.pdf.