An Open Architecture for the Construction and Administration of Corpora

Constantin Orăsan and Ramesh Krishnamurthy

Computational Linguistics Group
School of Humanities, Languages and Social Sciences
{C.Orasan, R.Krishnamurthy}@wlv.ac.uk
University of Wolverhampton
Stafford Street
Wolverhampton, WV1 1SB
United Kingdom

Abstract

The use of language corpora for a variety of purposes has increased significantly in recent years. General corpora are now available for many languages, but research often requires more specialized corpora. The rapid development of the World Wide Web has greatly improved access to data in electronic form, but research has tended to focus on corpus annotation, rather than on corpus building tools. Therefore many researchers are building their own corpora, solving problems independently, and producing project-specific systems which cannot easily be re-used. This paper proposes an open client-server architecture which can service the basic operations needed in the construction and administration of corpora, but allows customisation by users in order to carry out project-specific tasks. The paper is based partly on recent practical experience of building a corpus of 10 million words of Written Business English from webpages, in a project which was co-funded by ELRA and the University of Wolverhampton.

1. Introduction

1.1. What is a corpus?

The term ***corpus*** has been used to designate a body of naturally-occurring (authentic) language data which can be used as a basis for linguistic research (Leech, 1997). A corpus can consist of written texts and/or spoken texts of general language, or it may represent only a particular genre or language variety. Currently, the term ***corpus*** has come to be applied specifically to a body of language texts that exist in electronic format. The explosion of information available online has made it easier to build a corpus for a particular purpose by downloading relevant texts from the World Wide Web.

1.2. What are corpora used for?

The use of corpora for a variety of linguistic and non-linguistic purposes has increased rapidly in the past few years. Teachers, students, and researchers in a variety of fields (e.g. languages, business studies, law, medicine, and engineering) use corpora for teaching materials, classroom exercises, and dissertations. Software developers, engineers, and programmers use corpora to develop reference tools, CALL and NLP applications. However, large general-purpose corpora are not suitable for many tasks. And the rapid development of the web has improved access to data significantly. As Jean Veronis has suggested "Today, one can easily surf the web and download millions of words in no time at all" (email announcement on Euralex-list of Armstrong, 1999).

1.3. The need for more domain-specific corpora, and software to aid in corpus building

In order to meet the demand for corpora, various data initiatives and corpus projects have collected large amounts of electronic texts, but there are still domains for which corpora are not available. Therefore, when researchers decide to work in such a domain, they have to design their own corpora. This has given rise to a new need: for generally available, flexible software.

1.4. The need for reusable corpus building software

Another problem is the enormous duplication of effort: it is not at all uncommon for researchers to develop tailor-made systems that replicate much of the functionality of other systems, and subsequently to create programs that are highly purpose-specific and cannot be reused by others. The reusability of programs and data is a much-discussed topic in recent years, e.g. 109 references to reusability in the documents at CORDIS, the European Community Research and Development Information Service website. Indeed, the existence and success of linguistic resources distribution agencies such as ELRA and LDC indicate the level of demand for reusable resources. In particular, reusable software has become a priority, to avoid constantly reinventing the wheel (Veronis, 1996).

1.5. Research focus on corpus annotation tools, rather than on corpus building tools

Much of the research in corpus linguistics has been directed towards designing annotation schemes for marking different linguistic features in the corpus texts (Garside, 1997a; Mitkov, 1999), and designing tools for doing this (Cunningham, 1996; Day, 1998; DeCristofaro, 1999; Garside, 1998b), but little attention has been paid to the acquisition of the data which constitutes the corpus (Davies, forthcoming). At the present moment we are not aware of any tool that helps corpus builders in their work. Given the large number of ad-hoc decisions that are made during the building of a corpus, some people have suggested that such a tool is impossible. However, we consider that it is possible to design a very general-purpose tool, which can be customized according to users' needs. In this paper we propose a client-server architecture that will implement the basic operations involved in building and administering a corpus, while allowing users to customize the specifications with minimum effort, and to adjust the software to the needs of a specific corpus-building project.

In Section 2 of this paper, we will summarize the main steps involved in building a corpus, address some of the problem areas, and outline some of the solutions adopted. In Section 3, the proposed client-server architecture is described and discussed. Section 4 indicates the conclusions drawn from this proposal, based on recent practical experiences in corpus building at Wolverhampton University.

2. Main steps in building a corpus

2.1. General language corpora and more specific corpora

In corpus linguistics, we are usually more interested in a whole range of language, rather than in an individual text or author. Therefore in building a corpus, we are interested in collecting data which comes from more than one source, and usually more than one genre. When building a general corpus of this kind, any available data can be included in the corpus. It is also possible to build specific corpora which reflect the language used in a certain domain (e.g. business language), and then only data from that domain is included in the corpus.

2.2. Corpus composition

A corpus is not just a collection of texts. Rather it tries to represent a language or a part of language (Biber, 1998). If there are many texts from the same source, they can lead a researcher to false conclusions about the language and any specific aspects of the language studied. Therefore the first question which corpus builders have to ask themselves is what types of data are going to be included in the corpus and in what proportions. At this point, the corpus builder usually has a clear view about the categories and subcategories of the texts which are to be collected.

2.3. Corpus size

When we are talking about building a corpus, we also have to decide what size it is going to be. In a few cases, the size of a corpus is not an important issue because the collection of texts is dynamic and open-ended (e.g. the Collins COBUILD Bank of English corpus at Birmingham University). In most cases however, the size of the corpus is known from the outset, and therefore there is a target which has to be reached, which marks the end of the data collection phase.

2.4. Data collection and copyright permission

After the decisions about corpus structure have been made, the next step in building a corpus is the actual collection of the data. Strictly speaking, for corpora based on printed materials, once texts have been selected for inclusion in the corpus, the obtaining of copyright permissions should be the next step. However, we are focusing on corpora based on domain-specific documents obtained from the web and, as we explain later (see 2.10), in this case the collecting of data usually precedes seeking copyright permission, as it is not easy to identify the candidate documents beforehand. The moment of identification of a suitable document is also the most obvious moment for downloading it.

2.5. Data collection methods

The data may consist of written or spoken language texts. Recent experience (Krishnamurthy, 1992; Clear et al. 1996) shows that the cheapest way to build a corpus is to use data which already exists in electronic format. Data can also be acquired by optically scanning printed material (suitable only for good quality print and paper) and thereby converting it into electronic format. This method is slower and more expensive. Even slower and more expensive is to keyboard printed texts, which is necessary for poorer quality print and paper, and for highly formatted texts, with words superimposed on background images, or words running at various angles across the page, and for non-sequential texts (e.g. magazine articles, which are frequently interrupted by highlighted extracted quotes or tables, illustrations, advertisements, etc). Until speech-to-text technology becomes more readily available, spoken material requires a keyboarder to listen to an audio source tape of some kind, and special equipment (with headphones, and foot-controls for pause, rewind, fast-forward, and playback) often has to be used to allow this transcription process to take place with reasonable speed and in reasonable comfort for the transcriber.

2.6. Data in electronic form: web-sourced corpora

Given that it is the cheapest and fastest way to collect data for a corpus, most corpora include large amounts of data which is already in electronic format. As mentioned earlier, with the current explosion of online information available, this task is becoming easier. In the remainder of this paper, the emphasis will be on building corpora using data from the web, but most of the discussion will also be relevant for other methods of data collection.

2.7. Web documents and the need for additional information about the documents

When a document is collected (the technical term inherited from librarians is *accession*) for inclusion in the corpus, various additional information about the text has to be recorded. This includes, for example, the source of the document, the date when the document was collected, and the person who collected it. In cases where the corpus includes monolingual documents in more than one language, the language in which the document is written also has to be recorded. This information is required for two reasons. Firstly, it is necessary for measuring progress and for other statistical procedures, e.g. to know how many files have been collected for each language, etc. Secondly, the information is required in order to know which tool to use in subsequent processing, for example part-of-speech tagging, as a different version of the tagging tool will need to be used for each language. Even for monolingual corpora, the specific information about the source of the document can be useful, e.g. for identifying and isolating texts belonging to specific varieties of the language. By recording the additional information at the time of data collection, we avoid the need for a more laborious and intensive retrospective categorization of texts at a later stage, when some of the information may no longer be readily available. In the case of domain-specific corpora, the texts can be immediately categorized according to predefined categories. One could argue that the recording of additional information at the time of data collection slows down the process of collecting data. This is true, but it saves a lot of time later, and the categorisation of texts is an important and necessary part of corpus documentation.

2.8. Data collection staff and the recording of additional information about web documents

The task of data collection is often carried out by relatively unqualified people, on low-paid short-term contracts, with little motivation, and little interest in the later stages of the corpus. When the corpus is built in a university, the people involved in collecting data are usually students. They work only few hours a week, and they have different backgrounds and working styles. Therefore the corpus builder must ensure that they all record the same additional information about the documents collected, and use the same format. As stated earlier, this information is essential both for progress measurement and statistical purposes, as well as for later processing. If the information is not in the same format, a lot of time is wasted in re-assembling it in a common format for subsequent operations which have to be implemented on the whole of the corpus.

2.9. Danger of data duplication

One danger particularly evident in building a corpus from web documents is the duplication of data. It is possible that two different data collectors may visit the same website and download the same documents. This causes a significant waste of time and effort in two ways: the time and effort of the second person downloading already collected data, and the additional time and effort expended in detecting and removing the duplicate documents. Detection is actually not so difficult: the document URL (Universal Resource Locator) will occur twice in the document list, signaling the likelihood of duplication.

2.10. Obtaining copyright permissions for documents collected

A major problem for all corpora, but especially for web-sourced corpora, is the obtaining of copyright permissions. Once web documents have been selected for inclusion in the corpus, permission is sought. It is necessary to keep a note of the documents for which permission has been obtained, and those for which it has not. Using their URLs, it is possible to cluster documents belonging to a particular website. In an ideal situation, the copyright permission is obtained before any data is collected from that site. In this way there is no danger that some of the collected data will have to be excluded from the corpus because of lack of authorization. However, from our recent experience, this is not a sensible option if the project has to be completed in a limited timescale. An added complication is that it often takes a long time to obtain copyright permissions from a site, and during this process a large number of messages may be exchanged between the corpus builders and the copyright owners. All these messages need to be stored safely, in case they need to be referred to later. In some cases, the copyright owners ask for a list of the files downloaded from their site. If this information is not stored in a format that allows it to be retrieved quickly, it is difficult to produce the list.

2.11. Original format of web documents

When a corpus is built using electronic documents from the web, the documents first need to be saved in their original format. This is necessary for two reasons: firstly, the original format often contains useful information which must be extracted for future use (e.g. the headers of the HTML documents may contain information about the author; keywords; the date when the document was produced; the language in which it was produced, etc); secondly, there may be no immediate and easy method to convert the documents from their original format to the desired format for the corpus (usually plain text format). From our recent experience, most of the documents currently on the web are in HTML format. The next most common format is plain text (or almost plain; with some form of markup, SGML or similar). Considerably fewer documents are in Microsoft Word (.DOC) or Portable Document Format (.PDF) formats. However this is only our experience, and may not necessarily be valid for other domains or document-types.

2.12. Format conversion

In most cases, the documents collected from the web need to be converted from their original format to a common format for the corpus. For input to various data processing programs, such as concordances, part-of-speech-taggers, etc, plain text format is probably the ideal corpus format. Once all the documents have been converted to a common format, various operational tools and statistical programs can be applied. Some of the information about the original formatting of the text (e.g. superficial features such as font, font size, typeface, and non-linguistic material such as statistical tables and graphical images, etc) is not required for linguistic analysis, therefore it can be removed from the corpus text and this reduces text size and makes the text more easily readable.

2.13. Corpus annotation

A corpus can be used as it is (as a collection of texts) for linguistic research, but it becomes more valuable with annotation. Depending on the purpose of the corpus, different linguistic features of the texts can be marked. In a very simple annotation scheme, only paragraph and sentence boundaries may be marked. This is regarded as minimal and obligatory annotation for most current corpora. The next level of annotation is part-of-speech tagging, where every word in the corpus is associated with a tag indicating its grammatical category or word class. Part-of-speech taggers have reached a fairly satisfactory level of accuracy for most purposes, and so part-of-speech tagged corpora are reasonably common and are available in many languages. A higher level of grammatical annotation is syntactic mark-up, where full or partial parsing trees are marked for each proposition. This level of annotation is rapidly developing, and parsed corpora are now fairly widespread. The meaning of each word in a text can be marked using semantic tags. However, this is a considerably difficult task, so very few semanticallytagged corpora are available as yet. The development of corpora annotated with discourse entities is in its infancy. Other linguistic features of the corpus texts can be marked, e.g. prosodic, pragmatic and stylistic features, but because of their intrinsic complexity and the difficulty of making them machine-tractable, these levels of annotation remain directions for future development.

2.14. SGML annotation

Current standard practice is to annotate corpus texts using an SGML annotation scheme. For each document selected for inclusion in the corpus, a program can apply the specific SGML markup which has been adopted. If such a program is not available, or its results are not accurate enough, the texts will have to be annotated manually, or at least semi-manually (e.g. using editing macros in a word-processor environment).

2.15. Special characters

In some cases, converting a document from its original format to plain text format can lose some useful information. For example, if the text contains special characters other than the ASCII character set (e.g. mathematical symbols), these might be lost during the conversion. Therefore, instead of converting a file to plain text, the text is converted to an SGML format in which the special characters are preserved and encoded in a standard manner, called "entity references" (e.g. &egrave; stands for è).

2.16. Multilingual corpora and representing different character sets

With the increasing interest in multilingual applications and environments, the need for flexible tools that can generalise across several languages is evident. The main concern in building a multilingual corpus is how to represent the characters which are specific to different languages, given that in standard ASCII set most of these characters are not represented. One solution is given by ISO 8859 which contains 10 extensions for different languages. However, in an extreme case, it is possible that the corpus may have languages in it which cannot be represented by a single extension. In this case, each document will have to have information attached to it about the extension used. Another solution is offered by Unicode, a character encoding system designed to support the interchange, processing, and display of the written texts of diverse languages of the modern world[1]. There is no finalised version of Unicode as yet, and most corpus software programs do not support it. If a corpus designer does decide to use Unicode, the tools to be used will need careful consideration. The third way of encoding non-ASCII characters is to use entity references to replace them, according to ISO8879.

[1] "Unicode… will make multilingual software easier to write, information systems easier to manage and international information exchange more practical." (Press release by Unicode Inc, Mountain View, California, Feb 19, 1991)

2.17. Regularly updated websites and the need for multiple visits

If texts from online newspapers or magazines are being used in the corpus, the website may need to be visited at regular intervals to download data. If the frequency of update of the website is known, an automatic procedure can be scheduled for this task. If not, a slightly more complex procedure will need to be devised to do the retrieval automatically, involving the comparison of documents collected previously with the current website documents. For most other websites, downloading will be manual and once only, or at irregular intervals. The risk of duplication still applies must be dealt with (see 2.9)

2.18. Structure of individual websites

Each website has its own structure, but some general rules can be established to categorize webpages, and classify them to be downloaded or not (e.g. image files with .GIF or .JPEG suffixes, and audio files with the .WAV suffix might be rejected). A general filter mechanism can be designed, which can be enhanced with specific filters for specific website features or special requirements of the corpus.

3. The proposed approach

3.1. A client-server corpus building architecture

In the previous section we presented the main steps in building a corpus. In order to complete the building of a corpus in a short timescale and with minimal costs, as many steps as possible have to be automated. In this section we present a client-server architecture which will help corpus builders in their tasks (Figure 1). This architecture was designed using our experience during the building of a written English business corpus from the web.

3.2. Clients and a customisable server

The proposed architecture consists of several clients which perform very specific operations like adding texts to the corpus, or retrieving texts from the corpus for format conversion or other processing, etc. The server will be designed using a modular architecture and users will be able to customize it according to their needs by adding specific modules.

3.3. Modular programming: functionality and interchangeability

The idea of developing independent program modules to perform specific tasks is very familiar to computer scientists. This technique is called modular programming. A module is a section of code which performs a function or a sub-function. Modules are usually stand-alone units. Thus, for instance, if a module is attached to a program, it should add only its own function to the program. Similarly, if the module is removed, the program should lose only the specific function which the module performs (however it is possible that in some cases, where one module depends on the results of another, other functions will become unusable as a result of removing a module). Another characteristic of program modules is their interchangeability. Two modules which perform the same function should be interchangeable without the user

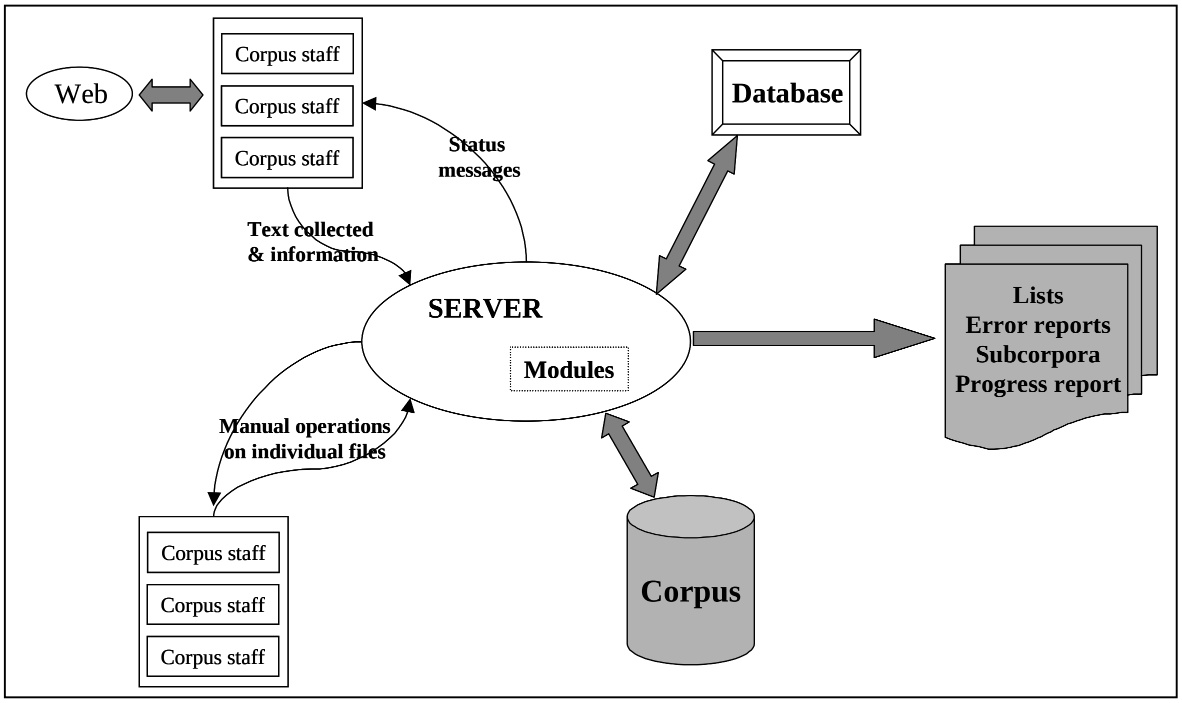

Figure 1. The architecture of the system

noticing any difference in the results. Differences can appear in performance parameters, such as running speed, memory used, etc, but this is quite normal, given that the implementations are different. The modules are very much self-contained units which interact in a very specific way with the program to which they are added.

3.4. Modular programming: ease of writing and maintenance, and reusability

Modular programming makes writing programs and maintaining them easier. Sections of programs can be written and tested independently. Moreover, modules written previously, which perform the required function, can be immediately included in the program. In this way, the time required for developing a new program is considerably reduced. In projects involving more than one programmer, modular programming becomes even more beneficial, if not essential; each programmer can develop one module at a time.

3.5. Program modules and off-the-shelf programs

Thus, the program will come with some predefined modules which perform very general operations, and users will be able to add other modules which perform specific operations. An example of a program module is one which converts an HTML document into plain text. This conversion can be done quite easily using an off-the-shelf program like *lynx*[2], but in our experience using the output directly is not a good idea, because it often contains a lot of information which is not needed for the corpus (e.g. for online newspaper sites, each page will have links to the main page, and maybe links to other related news items from the same day, or previous days, or even previous weeks). Instead, usually for each site, a filter has to be designed to remove any such unwanted information. For each site, there will also be a few files which cannot be converted using this filter, and for which a different one has to be designed.

During the format conversion phase of our recent corpus project, we learnt that the easiest procedure is to automatically convert the majority of files using a simple filter, and then to manually convert files which cannot be converted automatically. The identification of files which fail to be converted automatically is relatively straightforward. Either the number of words in the file drops dramatically (in a few cases, we even obtained zero-length files), or a simple search procedure reveals some control character sequences which should have been removed by the automatic conversion process.

3.6. The clients

The proposed architecture consists of different clients for performing very specific operations like adding texts to the corpus, or obtaining texts from the corpus for modification. For each task which cannot be done automatically, a client will be designed.

In most institutions, the corpus building project is only one among several ongoing projects, and the people working on it are only employed for a few hours per week. Therefore, it is rarely feasible for an individual to work on the same computer on every occasion, and consequently the files they are working on (whether during data collection or processing) will necessarily be stored on more than one machine. When the corpus

[2] *lynx* is a general-purpose distributed information browser for the World Wide Web which works on Unix based systems in text mode. It is highly configurable and can be easily used for converting large batches of HTML files simultaneously

building is completed, all the files which constitute the corpus must be available on one computer in order to carry out the final processing. So until that point is reached, someone has to periodically check all the computers to identify new data, and transfer it to the corpus processing machine. This operation takes time, and can introduce errors.

An alternative method is to use a specially designed client to collect information about files which are going to be added to the corpus, and send those files to a server program which stores both the information and the files. This approach has two advantages. Firstly, data collection staff are compelled to provide all the relevant information about the files. This information will include file names, locations, dates, languages, copyright holders, etc, and the program can ensure the uniformity and completeness of the information content and format. After the information has been entered, it is submitted to the server, which stores it in a database. Secondly, this method ensures that the version of the corpus on the server will always be up to date. The server can reject any file which has already been submitted on a previous occasion. Every time a file is added to the corpus, it is sent to the server, and for security reasons a copy of it is kept on the machine on which the client runs.

Whenever a file is collected from a new site, the server will ask for the copyright holder for the file. This may slow down the process of collecting data somewhat, but it will make the obtaining of copyright permissions easier. For a file from a site which has been visited before, the corpus staff can decide to use the information about the copyright holder from the previous visit, or to provide new information.

As mentioned earlier, it will not be possible to convert all the files using automatic methods. Therefore the original files will have to be obtained from the server, converted by the corpus staff, and then resubmitted to the server. Another client will be designed for this task. It is useful not only for converting the files to plain text format, but also for any other tasks which require human intervention (e.g. manual annotation).

Given that the corpus staff will work on different machines, some of which may run under Windows and others under Unix, the best language currently available for writing the programs is probably Java.

3.7. The server

The server software will carry out the basic operations of administering the corpus building process, and will be able to monitor the work of each member of the data collection team. The software will also maintain a database with various information relating to the project.

One set of information will relate to the files collected. It will include the name of the file, its source, the date when it was collected, the copyright holder, etc. Depending on the requirements of the project, more information can be stored for each file. The server will keep a configuration file which indicates which information is mandatory for each of the files collected. This configuration file will be sent to the client which is used for collecting data. The advantage of this approach is that all the people involved in collecting data will provide the same information. The corpus staff will submit this information at the time when the file is collected.

The configuration file will be also used by the database engine for storing the information about the files. As regards the choice of database engine, our current thinking favours ODBC (Open Database Connectivity), because it was ported on different platforms.

Another type of information to be stored in the database concerns copyright permissions. Our experience has shown that more than one message will usually be exchanged with the copyright holder, before permission is granted. All these messages will be stored in the database for future use. An automatic system for sending a standard copyright request could be designed, but we consider that this could be rather risky, and is likely to prove inefficient.

As mentioned earlier, different modules will be included in the server for performing automatic operations. These modules may be for converting files to plain text format, or for adding automatic annotation. Every time a file from the corpus is changed in any way, either by a module which performs an automatic operation, or manually by corpus staff, the changed version will be stored separately, without altering the original file. This is standard practice in corpus building, and ensures that we can always revert to a previous version of a file, or to the original file, at any time during processing. Files may get corrupted, or be deleted by accident, or a manual task may be poorly executed, so it is a worthwhile precaution to keep the immediately preceding version of a file, as only one process is then required to restore it to current status.

The server can be used for generating a corpus using a variety of different criteria. For example, if the corpus to be generated is required to consist of only certain categories of documents, only the files which belong to those specific categories will be retrieved. The information collected about each document can be used to automatically generate and insert an SGML header for each file.

Frequency lists, error reports, and progress reports can also be generated automatically by the server on request. These operations often require substantial processing time, and are therefore better performed during slack or static periods, for example overnight or at weekends, when the system is not being heavily used.

4. Conclusions

The use of corpora for a variety of linguistic and non-linguistic purposes has increased rapidly in the past few years, leading to a great demand for general and specific corpora. The process of corpus building is a new research area, which lacks standardisation and appropriate tools. In this paper we presented a highly customisable system for building and administering corpora, based on a client-server architecture that we hope will help corpus builders in their task. Given the current explosion of online information available and the recent experience gained in building corpora, the main emphasis in this paper was on building corpora from the Web, but most of the issues are relevant for any form of corpus construction.

5. Acknowledgements

Some of the recent experience of web-based corpus building was acquired during the Wolverhampton Corpus of Written Business English Project, which was co-funded by ELRA and the University of Wolverhampton.

6. References

Armstrong S., Church K.W., Isabelle P., Manzi S, Tzoukermann E. and Yarowsky D. (1999): Natural Language Processing Using Very Large Corpora, Kluwer Academic Publishers

Biber D., Conrad S. and Reppen R. (1998): Corpus Linguistics. Investigating Language Structure and Use, Cambridge University Press

Clear J., Fox G., Francis G., Krishnamurthy R., Moon R. (1996): COBUILD: The State of the Art, IJCL Vol 1 (2), John Benjamins, pp. 303-314

CORDIS, Community Research and Development Information Service

Cunningham H., Wilks Y. and Gaizauskas R. (1996) GATE – a General Architecture for Text engineering, in Proceedings of the 16th Conference on Computational Linguistics (COLING-96), Copenhagen

Davies M. (forthcoming): Creating Multi-Million Word Corpora from Web-Based Newspapers, paper given at the North American Symposium on Corpora, University of Michigan, 1999

Day D., Aberdeen J., Caskey S, Hirschman L, Robinson P, Vilain M. (1998): Alembic Workbench Corpus Development Tool, in *Proceedings of the First International Conference on Language Resource & Evaluation*, Granada, Spain, pp. 1021 – 1028

DeCristofaro J., Strube M. and McCoy K.F. (1999): Building a Tool for Annotating Reference in discourse, in *Proceedings of the Workshop on The Relation of Discourse/Dialogue Structure and Reference*, University of Maryland, College Park, USA, 21 June 1999.

ELRA, European Language Resources Association,
http://www.icp.grenet.fr/ELRA/home.html

Garside R., Fligelstone S and Botley S. (1997a): Discourse Annotation: Anaphoric Relations in Corpora in Garside R., Leech G. and McEnery A. (eds) *Corpus Annotation: Linguistic Information from Computer Text Corpora,* Addison Wesley Longman, pp. 66 – 84

Garside R, Rayson P. (1997b): Higher-Level Annotation Tools, in in Garside R., Leech G. and McEnery A. (eds) *Corpus Annotation: Linguistic Information from Computer Text Corpora*, Addison Wesley Longman, pp. 179 -- 193

Krishnamurthy R. (1992): Working Papers for Network of European Reference Corpora (Wordlists-WP5-87, Concordancing-WP5-88, Data Collection-WP6/WP7-57, Text Encoding-WP3-41) NERC, ILC, Pisa, Italy

LDC, Linguistic Data Consortium, http://www.ldc.upenn.edu

Leech G (1997): Introducing corpus annotation, in Garside R., Leech G. and McEnery A. (eds) Corpus Annotation: Linguistic Information from Computer Text Corpora, Addison Wesley Longman, pp. 1 –19

Mitkov R., Orasan C. and Evans R., (1999): The importance of annotated corpora for NLP: the cases of anaphora resolution and clause splitting, in *Proceedings of Corpora and NLP: Reflecting on Methodology Workshop*, TALN'99

Veronis J. and Ide N. (1996): Considerations for the Reusability of Linguistic Software available at http://www.lpl.univ-aix.fr/projects/multext/LSD/ LSD1.html