Intranet learning tools for NLP

William J BLACK
Centre for Computational Linguistics, UMIST
PO Box 88, Manchester
M60 1QD, United Kingdom
bill@ccl.umist.ac.uk

Simon HILL
Department of Computation, UMIST
PO Box 88, Manchester
M60 1QD, United Kingdom,

Mahmoud KASSAEI
Centre for Computational Linguistics, UMIST
PO Box 88, Manchester
M60 1QD, United Kingdom
mahmoud@ccl.umist.ac.uk

Abstract

This paper describes experience with the developed of tools for CL education using Java. Some are standalone Java applets and others are clients which connect to a parsing server using a LISP-based backend. The principal benefits are platform independence and reusability rather than world-wide web access, although intranet technology reduces the need for special purpose labs.

Introduction

Networked computers can be used to support learning in various ways. In computational linguistics, the predominant pattern of use is twofold: Learning materials are distributed using hypertext, and laboratories are conducted in which students work directly with computational linguistics processors such as parsers and generators.

The 'authorware' approach to developing learning materials has not been popular in the teaching of computational linguistics because of the extensive labour involved in encoding content. Since CL is all about the use of powerful general mechanisms and expressive formalisms, the idea of writing learning materials using less expressive tools has little appeal.

However, the new technologies of the internet make it easier to combine media to produce integrated learning environments in which pedagogical materials can be intimately connected to mechanisms and resources. Using such approaches can produce payoffs whether or not distance learning is involved. A better integrated set of resources for laboratory activities makes fewer demands on support staff such as graduate demonstrators. The ability to encapsulate mechanisms and tools in applets also means that the need to maintain special purpose laboratories is diminished, and it is also possible to promote CL to potential students in schools.

This paper reports experience with the use of web browsers to provide practical activities to an introductory class of computational linguistics students. We concentrate on the tools developed locally, although we make use of others where appropriate. Much of the discussion focuses on what is possible with the constraints imposed by current network software.

1 Learning Objectives

Behind the practical work reported in the rest of this paper is an assumption that introductory CL education should provide learning environments both for the linguistic and computational aspects of CL.

For the former aspect, a simplified grammar development environment is required; for the latter, an interactive exploratory tool which can step through processes like constructing a derivation, applying a specific search algorithm, relating the data structures to the representations produced as the result of analysis.

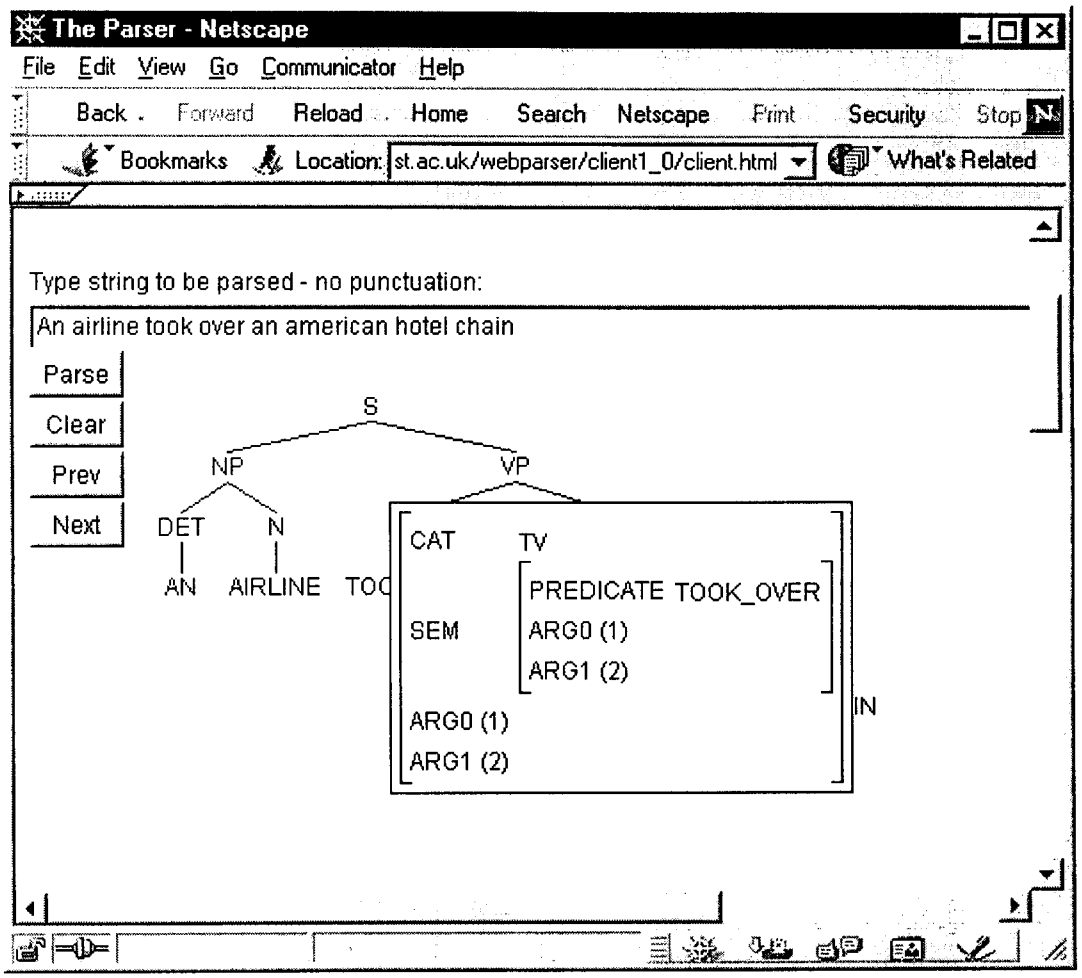

Figure 1 The client-server parser running in Netscape browser

2 Tools for exploring grammars and linguistic representations

In the preceding paragraph, we contended that *simplified* grammar development environments are required. Why not the real thing, like the Alvey Tools, Geppetto, LFG workbench, Pleuk, ProFit, ALEP et al? The target audience is the introductory CL student, either in the first year of a study programme or taking a CL module as part of a cognate discipline. Tools used by researchers are for later in the course, if we manage to retain the interest of the students, which we will only do if we make CL tools as accessible as the generality of IT applications.

Graphical user interfaces are essential: to give a student a graphics workstation and have them interact with Prolog in a terminal window conveys an unfavourable impression.

Some excellent learning software for CL is available, such as the Linguistic Instruments tools for CFGs DCGs, PATR and categorial grammars. Their main drawback is that they are tied to the MacOS platform.

2.1 Software platforms for portability

There are implementations of the programming languages considered suitable for NLP that hava graphical user interface (GUI) development tools, but unfortunately these are not standard. The ideal would be to use programming

languages and GUI development tools that are available on different computer systems. The two most widely known platform-independent GUI development tools are Tcl/TK and Java.. Of the two, Tcl/TK is simpler, but Java has the crucial advantage that it can 'run anywhere' in a web browser, whereas for Tcl/TK, the user has to obtain a plug-in.

Integrating tools with teaching material

Another use of Java running in a browser is to embed the interactive elements into hypertext, allowing a close linkage of textual learning materials with practical activities. As well as using Java to provide run-anywhere programs, teaching materials can be enhanced by applets which display linguistic analyses graphically, and even have the displayed information open to manipulation.

The grammatical resources used are the grammars in PATR-II as presented in Gazdar and Mellish, 1989.

The client-side tree and AVM-drawing programs illustrated in Figure 1 can be used independently of the server discussed below, to produce animated teaching notes. The data to be displayed and manipulated in this applet is specified in applet's parameters, so it is possible to use it to illustrate different analyses at different parts of an educational hypertext. There is one structured string parameter which encodes the tree, and one further parameter for each node, which encodes the content of the respective AVMs.

The *Thistle* tree-editing suite (Calder, 1998) is a well-developed interactive tool for working with linguistic representations such as trees and AVMs is a more sophisticated alternative. However, the tree-drawing program described is only a part of a more sophisticated mechanism which links the linguistic information displays to on-line parsing.

3 On-line Parsing

Having chosen to use Java for the development of graphical displays of linguistic data, we have to consider what is the most appropriate engine for the analysis or generation behind them. One possibility would be to re-write the code for those algorithms in Java, but this ignores the possibility of re-using existing programs written in Prolog or LISP, which are documented in various textbooks. These implementations are more established than existing Java-based parsers, which have not so far featured in published learning materials.

There are several practical ways in which a Prolog or LISP parser can have its output displayed graphically in a browser. One is to invoke the parser from within a CGI script on a web server. This strategy has been adopted by Ramsay (1999) for presenting the *Parasite* project on the WWW.

A disadvantage is that each request involves the overhead of starting a new Prolog process, and a consequently inflated response time, as well as complex arrangements to maintain dialogue state information.

Client-server parsing

In the CCL webparser system, the LISP-based parsing program acts as a server which accepts socket connections from the Java applet that handles the display.

The intention was that the LISP parser should be a black box, so we elected to build th server using *Expect*. This is an extension of Tcl/TK (Libes, 1995) that is specially designed to 'automate' interactive programs.

The Expect program spawns a LISP process and then controls it by simulating the user with its *send* and *expect* commands. It uses its *send* command to load linguistic resources, and then it opens a server socket, awaiting requests from networked clients.

When socket connections are accepted from users' browser clients, the expect program passes these on to the LISP parser program, and awaits the response, which it passes back to the client.

The advantage of using Expect as an intermed-iate layer is that it enables the LISP process to react to different client programs without having to restart to serve each of them. It also lets the server save the results of a parse, e.g. a

chart, and let the user ask for information that is stored in chart edges for some time after the initial parse was done. When a parse request is processed, a reference number is generated and the chart is cached, indexed by that number. The reference numbers are notified to the client as part of the message summarising the result of the analysis.

The client-server protocol

A simple protocol has been defined for communication between the client and the server, as shown in Table 1. The client prefixes each request by one of the keywords *parse, tree,* and *avm.*

Table 1 Client-server protocol for web parser

Client request	Server response
parse word*	parsecount p# num
tree p# tree#	showtreelp#ltreenode*
avm p# edge#	showavmledge#ldagnode*/ dagedge*

Key: p# reference number of the parse request
tree# nth analysis produced in parse p#
edge# nth edge from the chart of parse p#
treenode is a triple *node#, parent, label*
dagnode is a number
dagedge is a triple *from,to,label*

Figure 1 illustrates this client-server system running in a Netscape browser.

Educational use of the client-server parser

The version displayed is under development, and still lacks some features it would need to be really useful in an educational context. The most important practical requirement is for the user to be able to modify the grammars and lexicons that the system uses. To make the program useful for reinforcing different stages in a course, the user has to be able to select alternative grammars, and to be able to edit his/her own. This is inconvenient to implement when the interface is an applet, because an applet is not normally permitted to save programs on the local machine. However, we expect to have a workaround for this in place for the next academic year.

Being able to display the results of parsing in the form of conventional diagrams is an advance on textual interaction. When the student is working on a grammar, the displays produced give feedback which is much more readily understood than textual output.

Students in the later years of the CL course have been able to use the system to visualise the results of parsing with grammars under development in another window. The planned file-management facilities will enable the program to be used regularly with the target users (students new to CL) in the next academic year.

Tools to support the understanding of CL processing

Understanding grammar notation is only part of what a CL student needs to learn in practical classes. Accordingly, we have started to develop a suite of tools that animate parsing algorithms using Java only, without the need for a server.

The first of these was developed in a day, and lets the student step through the construction of a derivation, either top-down or bottom-up. This program is illustrated in Figure 2. It has been extended so users can supply their own rules. This program has been successfully deployed with first-year students already and appears to have enhanced their understanding of these basic concepts of formal language theory and parsing algorithms. The same program can be used, with different data, to illustrate search algorithms in general.

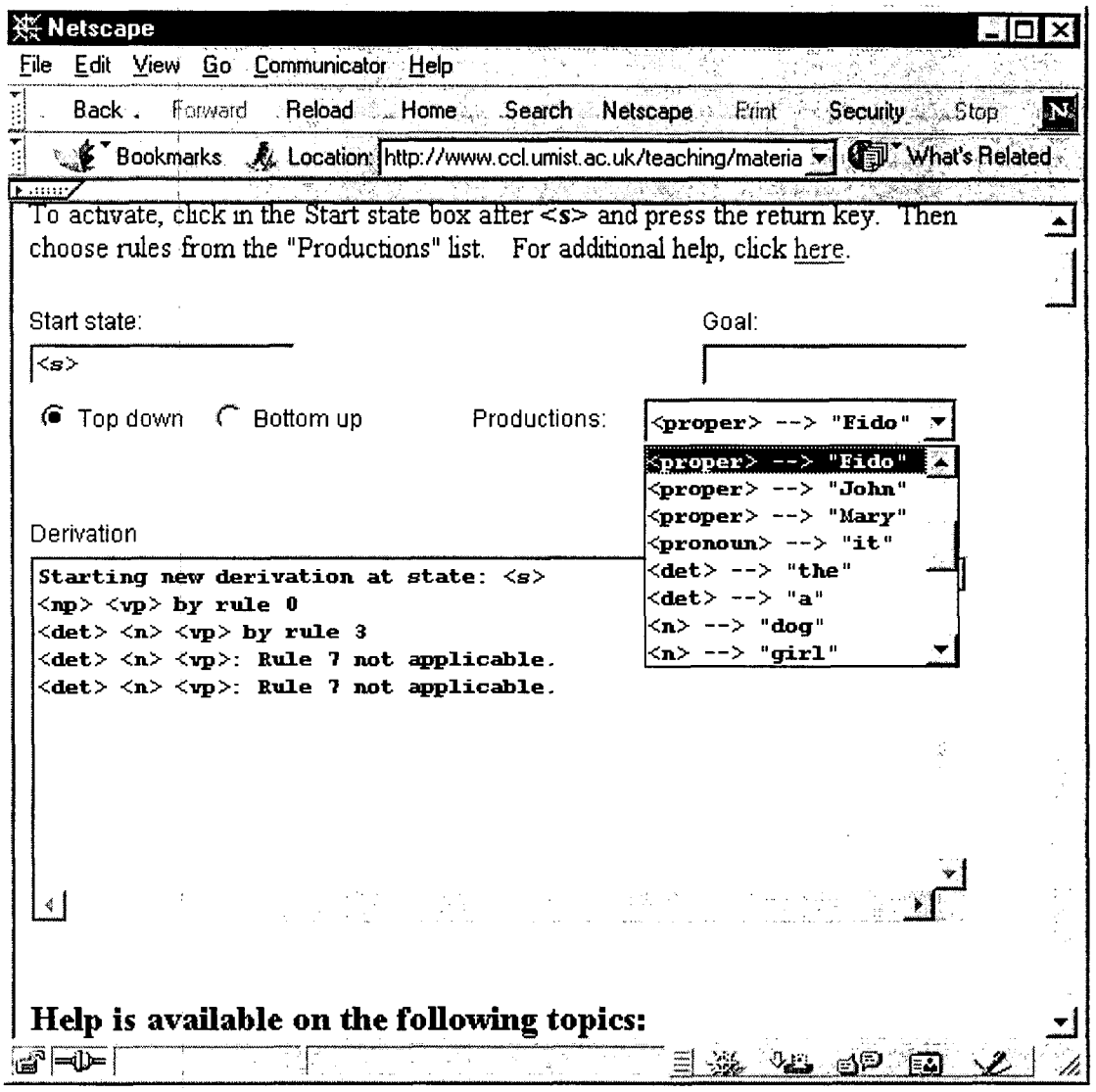

Figure 2 Interactive tool for constructing derivations

We are modifying this program so that it can show the agenda at each step of an automatic derivation to enhance the student's understanding of search algorithms in general. At some point, this program will be integrated with the graphical presentation elements.

Discussion

The software described in this paper is still under development. It has been used by students, but not yet extensively. It will be difficult to get objective information on what difference such accessible interactive learning tools make to the students' learning, but a study of usability will be conducted during the next academic session.

The main advantage (for the teacher) in using a WWW-based environment for delivering natural language processing practical work to students is that once developed, the laboratory needs less specialist provision and staffing than before. Students can also work in their own time in campus-wide computer facilities without having to have NLP software installed.

Planned developments

The software described is at a rather immature phase of development, but most of the hard work has been done. Planned future developments after the file management facilities are completed include:

- graphical viewers for the chart in chart parsing
- dependency tree viewers
- discourse representation viewers
- incremental tracing of generation algorithms

Also on the agenda is to make on-line access to a range of well-known NLP programs an integral feature of all the teaching materials for introductory NLP. Using the same *Expect*-based meachnism, we can put user interfaces around taggers, morphological analysers, dictionaries and corpus-analysis programs and link to them all with hypertext.

Conclusion

We have described recent work on the implementation of student-oriented tools for natural language processing.

Three kinds of tool have been developed, which complement those available elsewhere. Firstly a parameterisable applet which enables the lecturer to incorporate syntax trees with embedded attribute-value matrices into hypertext teaching materials. Second, a portable HTML-Java interface to a parsing server residing on a departmental intranet server. Finally, we have developed a tool that lets the student explore the process of analysis step-by-step, to reinforce understanding of the basic algorithms for NLP.

Both the second and the third tool are now being enhanced to enable them to be used by students to develop their own resources, and effort is also under way to complete graphical viewers for a more complete inventory of linguistic representations.

The client-server method of constructing an on-line parser with a user interface is an attractive approach because it allows us to re-use existing tools, for example those which are featured in teaching materials, such as Gazdar and Mellish (1989). The *Expect* plus *Java* technology provides a good solution for developing user interfaces for local use; the possibility of deploying these within hypermedia provides an additional opportunity to package the practical work within course materials.

Tools used

The client-server program was developed using JDK1.1.7, Allegro Common Lisp™, version 5.0, Expect version 1.5, and the Apache Web server, under Solaris™ 2.6. The client programs have been tested on Netscape 4.0 and 4.5, and on Internet Explorer™ 4.0. It is planned to verify that the LISP portion can run under a freely available LISP, with a view to making the tools available to anyone interested in using them.

Availability

The client-server parser can be tried out at http://bermuda.ccl.umist.ac.uk/webparser/client1_0/client.html
When the software is available for download it will be announced at http://www.ccl.umist.ac.uk.

Acknowledgements

Simon Hill was supported by a EPSRC studentship, and some further financial support has been provided by ELSNET....

References

Calder, J. (1998) How to build a (quite general) linguistic diagram editor. In *Thinking with Diagrams* (TwD98), Aberystwyth, UK.

Gazdar, G. and Mellish, C.S..(1989) *Natural Language Processing in LISP*. Reading, MA: .Addison Wesley

Libes. D. (1995), *Exploring Expect*. Cambridge, MA: O'Reilly,

Ramsay, A.M. (1999) Weak lexical semantics and multiple views. In H.C. Bunt and E.G.C. Thijsse, Eds, *3rd International Workshop on Computational Semantics*, pages 205--218, Tilburg, The Netherlands.