The Program Committee Virtual Workbench

Ulf Nilsson
Department of Computer and Information Science
Linköping University
581 83 Linköping, SWEDEN

ulfni@ida.liu.se
http://www.ida.liu.se/labs/logpro/ulfni/

1 Introduction

We give a preliminary report and evaluation on the use of logic-based Internet tools in the organization of the program committee work of the International Logic Programming Symposium 1997. Several prototype systems (mostly written in SICStus Prolog 3#5 [1]) were developed to assist in:

- taking care of submissions,
- distributing papers to reviewers,
- managing the reviewing process,
- preparing data for the PC meeting, and
- notifying authors of acceptance/rejection.

The most important tools were the *PC chair workbench*—used by the program committee organizers to facilitate communication with authors and reviewers, and to monitor the reviewing process—and the *Reviewer's workbench*—used by the reviewers to monitor the status of their reviews, to provide statistics and to facilitate discussion on papers with conflicting reviews. By implementing the tools as CGI-scripts we were able to build semi-interactive environments with a very small effort; and users were able to use existing clients, such as Netscape and Internet Explorer, to interact with the tools.

The paper is organized as follows: In Section 2 we outline the principal architecture of the workbenches. In the following sections we discuss the various tools used at different stages of the program committee work; we show by means of snapshots what the interaction looked like and what kind of services were provided.[1] Finally we summarize the conclusions from our work; listing pros and cons of the tools and the underlying technology.

We assume prior knowledge in logic programming, basic knowledge in HTML [7] (in particular HTML forms) and the principles of the World-Wide Web [2].

[1] Most of the figures have been edited, partly not to disclose confidential information; partly for typographical reasons.

```
main :- write('Content-type: text/html'), nl, nl,
        write('<html>'), nl,
        write('<head><title>Test</title></head>'), nl,
        write('<body>Hello!</body>'), nl,
        write('</html>').

:- main.
:- halt.
```

Figure 1: Simple Prolog CGI-script for generating HTML code on the fly. The initial line declares the MIME-type of information being transmitted; in our case marked-up text.

2 Overview of the system

In this section we outline the basic architecture of the tools. For more information on how to use Prolog for Internet applications, see e.g. the PiLLoW library [4].

The workbench consists of a collection of Prolog programs, most of which are run under the Common Gateway Interface (CGI) [9]. The CGI is a way for the Web server to produce dynamic HTML code—instead of just dispatching an HTML file, the server may invoke programs (stored in special directories typically named `cgi-bin`) that output HTML code on the fly. Additional information can be transmitted to the program using environment variables. For instance, consider the following shell script:

```
#!/bin/sh
script=$HOME/prolog$PATH_INFO.pl
exec sicstus -l $script 2> /dev/null
```

If the script is named `pl` and a client requests the URL

`http://`*domain name etc.*`/cgi-bin/pl/test`

then the `pl`-script will be executed by the server. The environment variable `$PATH_INFO` will be set to `/test` by the server. Hence, the script invokes the Prolog system which consults the Prolog program `$HOME/prolog/test.pl`. If `test.pl` looks as in Figure 1, then the script will produce the output:

```
Content-type: text/html

<html>
<head><title>Test</title></head>
<body>Hello!</body>
</html>
```

The output is piped to the server and sent to the client that posted the original request.

As pointed out above, the server may send information to the CGI-script using environment variables. However, also a client (such as Netscape or Internet Explorer) may transmit data to the CGI-script using HTML forms. The form

provides a mechanism for sending attribute/value-pairs to the server. Forms may contain several types of objects, for instance, radio buttons, checkboxes, text fields, menus and submit buttons. Each object can be given an attribute name and the user may attach a value to it. The server relays this information to the script (which is be specified as an attribute of the `<form>`-tag). There are a number of libraries (e.g. PiLLoW [4]) for parsing the information and turning it into an association list. In our examples we use a predicate `html_get_data/1` to parse the information and the result is a list of attribute/value-pairs. This mechanism makes it possible to achieve a limited form of interactivity. The client may send data to the script via the server; the script processes the data and outputs a new form that, when completed, invokes the same script again. Note that the script does not preserve its state so either it has to pass its state to the client, or store the state in a file between invocations.

The program committee workbench consists of several software components described in the rest of this paper. Two components were particularly useful:

- the *PC chair* (PCC) *workbench* for accepting and distributing papers, for monitoring the process of reviewing and for notifying authors of acceptance or rejection. The PCC workbench was accessible only to the PC organizers.
- the *Reviewer's workbench* was accessible to all PC members. By means of this tool each PC member is granted access to the reviews of all papers assigned to the him. Each PC member can see (1) which of his reviews are missing; (2) evaluation data and the actual review from other reviewers of the papers assigned to the member; (3) the current ranking of all papers. In addition the tool facilitates a forum for textual communication where the PC members assigned to a paper may get into discussions in case of conflicting reviews.

The general scheme of both tools is as follows:

```
main :-
        html_get_data(Form),
        lookup(page, Page, Form),
        write('Content-type: text/html'), nl, nl,
        write('<html>'),
        write('<head><title>Title</title></head>'),
        write('<body>'),
        webpage(Page, Form),
        write('</body></html>').

webpage(a, Form) :-
        write('<form action="test" method="get">'),
        write('<input type="submit" name="page" value="b">'),
        write('</form>').
webpage(b, Form) :-
        write('<form action="test" method="get">'),
        write('<input type="submit" name="page" value="a">'),
        write('</form>').

:- main.
```

The script fetches the form data (a list a attribute/value-pairs). One of the attributes (in the example `page`) specifies what information to display. The script then outputs a common head- and body-part after which the script branches to the appropriate `webpage/2`. In most cases the `webpage/2` outputs a form that invokes the script recursively. In our example, the first definition outputs a form with a submit button that invokes the second definition, and the second definition produces a form that invokes the first definition.

3 Submissions

Although almost everyone has access to the Internet we decided to take a rather liberal view when formulating instructions for submission of papers. Hence, paper submissions were allowed but we encouraged sending a uuencoded, gzipped postscript file using email. It eventually turned out that everybody sent electronic versions. However, not everyone followed the instructions. We anticipated this and as a consequence the processing of papers was manual. This did take a considerable amount of time but it is not easy to see how to automate this step today. Due to the lack of standards for distributing texts many of the submissions (even those that came in postscript format) had to be "massaged" in order to print properly. Some problematic cases appeared but we eventually managed to get all submissions into printable postscript format. (Only one hardcopy had to be sent to the program committee members.) Each submission was to be accompanied by additional information:

- the names of all authors,
- the title of the paper,
- abstract,
- keywords,
- the email address of the contact person, and
- additional contact information.

For this purpose we used an HTML form accessible from the conference home page. The form was processed by a Prolog program and the information was stored in a paper-database with entries of the form:

`paper(`*No*, *Authors*, *Email*, *Title*, *Keywords*, *Abstract*, *Address* `).`

The first argument is a unique number assigned to each submission.

In parallel with receiving submissions, confirmations were sent out once the paper and the accompanying information were received. For this we used the PCC workbench. The interface of the tool (see Figure 2) consists of an HTML frame-set. The left frame (with dark background) is an HTML file that invokes three different Prolog scripts whose output show up in the right frame. Clicking on "Papers" enables the user to select one of the submitted papers (see Figure 2) after which the system displays data from the papers database (see Figure 3). The information consist of the *title* of the paper (this is a clickable link to the postscript version of the paper), *authors* and *email address* (also clickable). There are three buttons for sending letters of confirmation, rejection and acceptance. When clicking on the buttons the tool automatically generates a standard email letter addressed to one of the authors using the title- and email-fields from the paper database (see Figure 4). By displaying the email using an

Figure 2: The PCC workbench with the view of the submitted papers

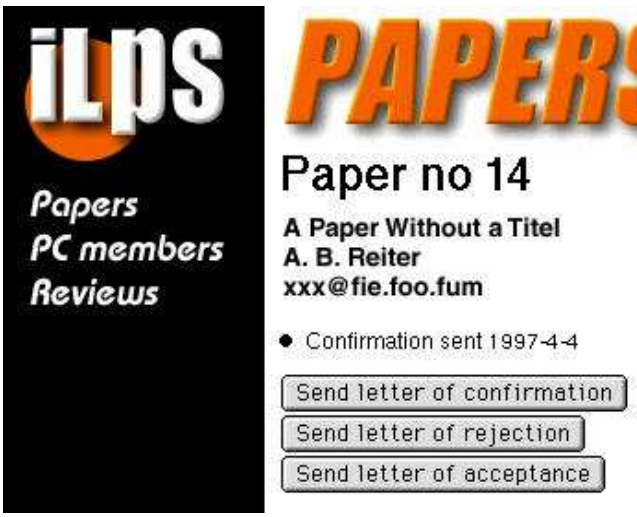


Figure 3: The PCC workbench presents information about individual papers and facilitates communication with the authors.

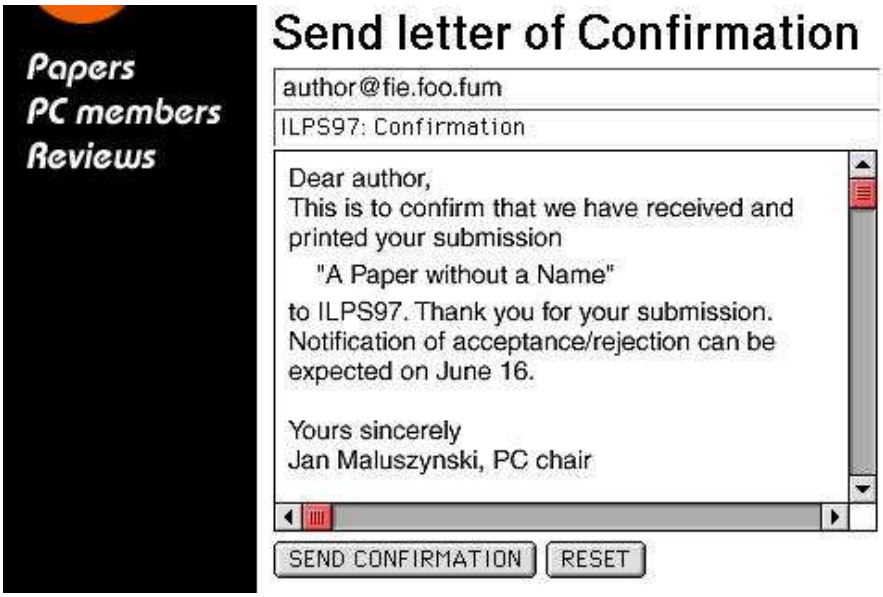


Figure 4: The interface for sending letter of confirmation to the authors of individual papers.

HTML form it is possible to edit the mail before sending it. All confirmations were stored in a special log-file to avoid multiple confirmations (see Figure 3).

4 Distribution of papers

After having received all submissions, each paper was assigned to five PC members. This work was partially automated by a CHIP V [5] program that matched keywords of the papers against each PC member's areas of expertise. The abstracts (stored in the paper database) were also made accessible to the PC members over the Web.

Distribution of papers to PC members was more or less automatic. We did not use the Web for this due to security problems—at the time we did not know how to prevent unauthorized access to papers, but in retrospect using the Reviewer's workbench had probably been a reasonably safe solution for this.[2] The distribution was made by a Prolog program using a reviewer's database of the form:[3]

`reviewer(`*Id*, *Name*, *Email*, ..., *List-of-papers* `).`

The first argument is a unique identifier of each PC member. The second argument is the full name of the PC member; the third argument the email address. Then follows some additional arguments (e.g. passwords) and finally the list of all papers assigned to the PC member.

[2] One complication is that PC members often use colleagues, or Ph.D. students to do some of the reviewing for them.

[3] The email was *composed* by the Prolog program, but to actually send the email we used the Unix systems library provided by SICStus 3.5 [1].

5 Monitoring of the reviewing process

Review forms written in LaTeX were available over the Web. The forms were collected by email and when the reviews came in they were manually checked and stored in a directory structure—one directory for each PC member. A Prolog program was regularly used to scan the directories, extracting numerical data from newly received forms and storing this information in a database with tuples of the form

`review(`*Id*, *File name*, *No*, *Overall*, *Confidence* `).`

The arguments are, in order: the unique identifier of the PC member (moreover the directory where the reviews were stored); the file name of the review; the number of the paper; the overall evaluation of the paper (an integer 0–5) and the confidence of the reviewer (an integer 1–3).

The databases previously described were used by the PCC workbench which turned out to be a valuable tool, in particular, for monitoring the reviewing process. By means of the tool we were able to check when reviews came in, we were able to easily spot missing reviews, and, most importantly, we were able to detect conflicting reviews (see Figure 5). The information provided by the PCC workbench was protected using the ACL facility (Access Control List) supported by the HTTP protocol [3]. Only the PC organizers had access to this tool.

The Reviewer's workbench turned out to be even more important; the tool provides a number of services to the PC members. (A similar, but more extensive, tool is developed by W. McCune for CADE-14 [6].) The tool is a single Prolog program (500 lines of code) able to invoke itself through HTML forms, as described in Section 2. When entering the system (by means of an individual password checked by the tool) the reviewer is presented with the menu in Figure 6(a)). From this interface (deliberately kept plain to reduce bandwidth) the PC member is provided three services:

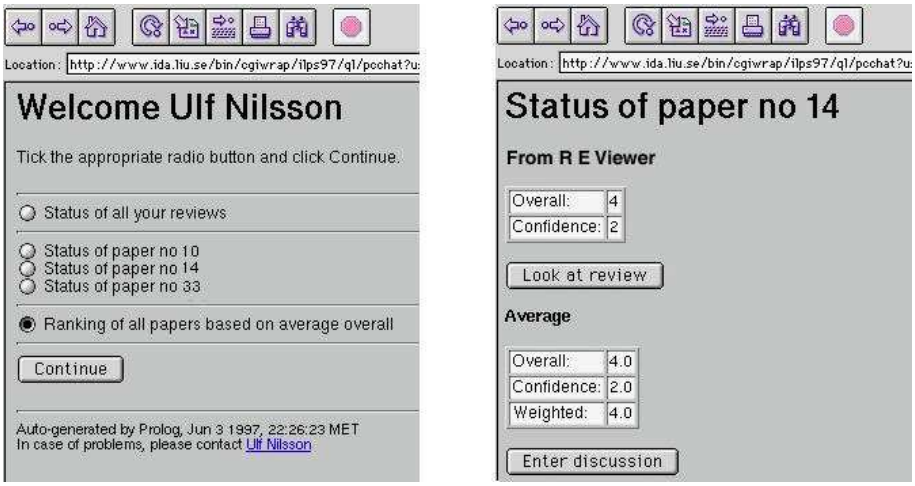
- he may check if reviews of papers assigned to him have been received or if they are still pending (see Figure 7).
- he may check the overall evaluation and confidence of other reviewers on all papers assigned to him (see Figure 6(b)). For each available review it is also possible to look at all the actual reviews of the paper.
- he may check the current ranking and average values of all papers (see Figure 8).

To facilitate discussion among PC members in case of conflicting reviews, the tool also provides a forum for discussion on individual papers. By clicking on the button "Enter discussion" in Figure 6(b), the tool generates the form in Figure 9. Here each PC member may read statements from other reviewers (in the example there are no previous statements). The PC member may also write his own statements and submit them to a database.

6 Notification

At the time of writing this, the reviewing process is coming to an end but the program committee meeting has not been held and authors are not yet notified

Figure 5: Mocked-up statistics for paper 14 in the PCC workbench. (In the example only one review is available.)



(a) (b)

Figure 6: Top level in the Reviewer's workbench (a) and statistics on individual papers (b).

of acceptance or rejection. The sending of notification is facilitated by the PCC workbench, much in the same way as confirmation (see Figure 4). That is, a standard (editable) letter is displayed and then dispatched. How to distribute review forms is still an open question. Since the reviews are written in LaTeX, our intention is to write a program that outputs and dispatches uuencoded, gzipped postscript versions of the reviews. (The file names of all reviews are stored in the review database and email addresses in the paper database, so this should be relatively easy.)

7 Conclusions

We have reported on the use of logic-based Internet tools for managing the work of the program committee of the International Logic Programming Symposium 1997. Several tools were developed to assist in:

- taking care of submissions,
- distributing papers to reviewers,
- managing the reviewing process,
- preparing data for the PC meeting, and
- notifying authors of acceptance/rejection.

The most important tools were the PC chair workbench—used by the PC chair to facilitate communication with authors and reviewers, and to monitor the reviewing process—and the Reviewer's workbench—used by all PC members to monitor the status of the reviews of each PC member, and to facilitate discussion on papers with conflicting reviews.

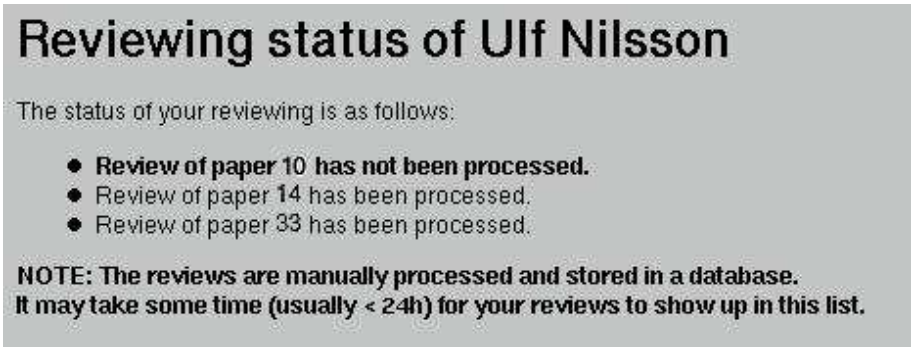


Figure 7: The Reviewer's workbench displays the status of the PC member's reviews

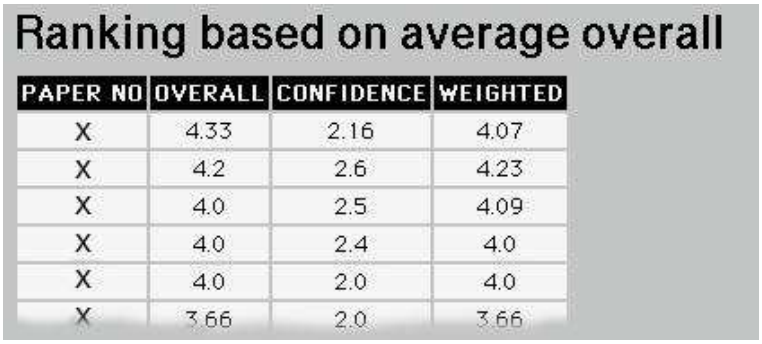


Figure 8: The ranking of all papers (excerpt).

The overall experience in developing and using the tools has been very positive. The time required to develop the prototypes was very short; both the PCC and Reviewer's workbench were developed in only a couple of days. In particular the Reviewer's workbench was of great use judging from the feedback from the program committee members.

In retrospect there are a couple of steps that we would probably have done differently, given a second chance:

- We decided not to use the Web to make submissions accessible to PC members; papers were rather distributed using email. This turned out to be unnecessarily complicated, both for the PC members and the PC organizers. A better solution had probably been to use the Reviewer's workbench for this, in which case each PC member could have been granted access to exactly those papers that he was responsible for. One complication, though, is that the PC member is not always the same person as the reviewer of the paper.

- A great deal of work was spent on managing incoming reviews. The form was written in LaTeX, but not all reviewers checked that the form passed through LaTeX, before submitting it. Another complication which we have

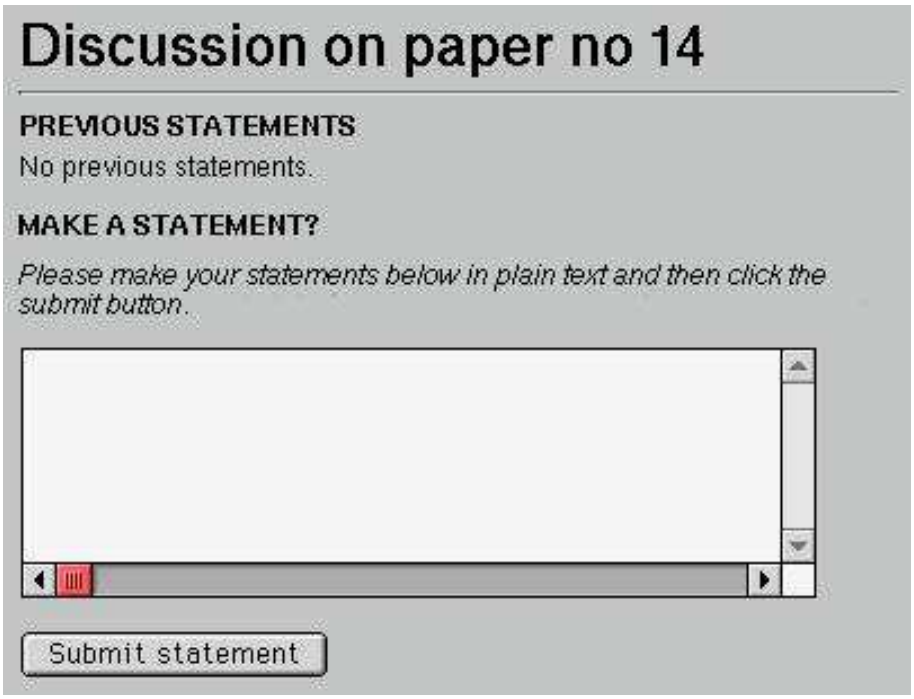


Figure 9: In case of conflicting reviews there is a forum for discussions (no discussion has started in the figure).

not yet resolved, is how to distribute the reviews to the authors. We plan to send out a postscript version using email, but it is not clear if everyone is able to read that. A better solution to the whole problem, had probably been to use an HTML form to submit reviews. With the aid of the Reviewer's workbench it had even been possible for the reviewer to edit reviews and re-submit them as a result of discussions on conflicting reviews. (See [6].)

- One shortcoming of the Reviewer's workbench was the lack of support to indicate updates since the last visit. PC members had up to 15 papers to review and browsing through all papers, looking for changes, was time-consuming. It would have been desirable to have some kind of indication in Figure 6(a) whether the status of a paper had changed or if there were new statements in the discussion forum.

Prolog turned out to be an excellent language for writing this kind of applications. As pointed out above it took only a couple of days to develop the tools, and a tool such as the Reviewer's workbench comprises only approximately 500 lines of code. However, in developing the tools we experienced two weaknesses of existing Prolog systems:

- Each time a script is run, the HTTP-server starts a Prolog process and loads a Prolog program. After the program has been run, the Prolog process dies. A more light-weight Prolog system would decrease the load on the

Web-server and make Prolog more competitive compared to existing script languages such as Perl. Another attractive solution is the concept of active modules described by Cabeza et al. [4].

- There is no good support in Prolog for accessing and storing persistent data. SICStus Prolog contains a database library, but it is a single user database (even for reading only, not more than one process can access the database). In Web applications, several clients may request simultaneous access to the same data or database. One solution may be to interface the Prolog system with an independent storage manager such as mini SQL [8], but a better solution probably is to extend the current database library to facilitate concurrent transactions.

A general problem when writing CGI-scripts is the lack of debugging facilities. Much initial time was spent on trying to figure out what went wrong when the client responded "Document contains no data". The situation improved when learning the most frequent sources of errors and learning to use print-outs to pin-point errors. Nevertheless, some kind of debugging facility would be very useful.

Acknowledgments

Thanks are due to the members of the program committee of the International Logic Programming Symposium 1997 for feedback and valuable suggestions. Financial support was provided by the Swedish Research Council for Engineering Sciences (TFR).

References

[1] J. Almgren et al. *SICStus Prolog User's Manual*. Swedish Institute of Computer Science, 1996. (Release 3#5).

[2] T. Berners-Lee, R. Calilliau, A. Luotonen, H.F. Nielsen, and A. Secret. The World-Wide Web. *Communications ACM*, 37(8):76–82, 1994.

[3] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945, IETF, 1995.

[4] D. Cabeza, M. Hermegildo, and S. Varma. The PiLLoW/CIAO Library for Internet/WWW Programming using Computational Logic Systems. In *Proc. of 1st Workshop on Logic Programming Tools for Internet Applications*, 1996.

[5] Cosytec. CHIP *System Documentation, Reference*, 1996.

[6] W. McCune. DEMO-97 Program Committee Web Pages for CADE-14. `http://www.mcs.anl.gov/home/mccune/pc-1.0/gate-demo.html`, 1997.

[7] D. Ragget. HTML 3.2 Reference Specification. TR REC–html32, World Wide Web Consortium, 1997.

[8] Hughes Technologies. Mini SQL. `http://sparcey.Hughes.com/`, 1996.

[9] W. Weinman. *The CGI Book*. New Riders, 1996.