ASAP

IST-2001-38059

Advanced Analysis and Specialization for Pervasive Systems

Initial Web Site

Deliverable number:	D11
Workpackage:	Dissemination of Results (WP9)
Preparation date:	1 April 2004
Due date:	1 April 2004
Classification:	Public
Lead participant:	Univ. of Southampton
Partners contributed:	Tech. Univ. of Madrid (UPM)

Project funded by the European Community under the “Information Society Technologies” (IST) Programme (1998–2002).

1 Introduction

A project web page with basic information about the project has been available since the start of the project (see `http://clip.dia.fi.upm.es/Projects/ASAP`). Later, this web site has been extended to also include the latest software releases and documents delivered by the project. The development branches of the integrated tool and its components are also on-line (via the CVS and SubVersion version control systems), currently open only to the developers within the different groups participating in the project.

The project web site includes three main areas:

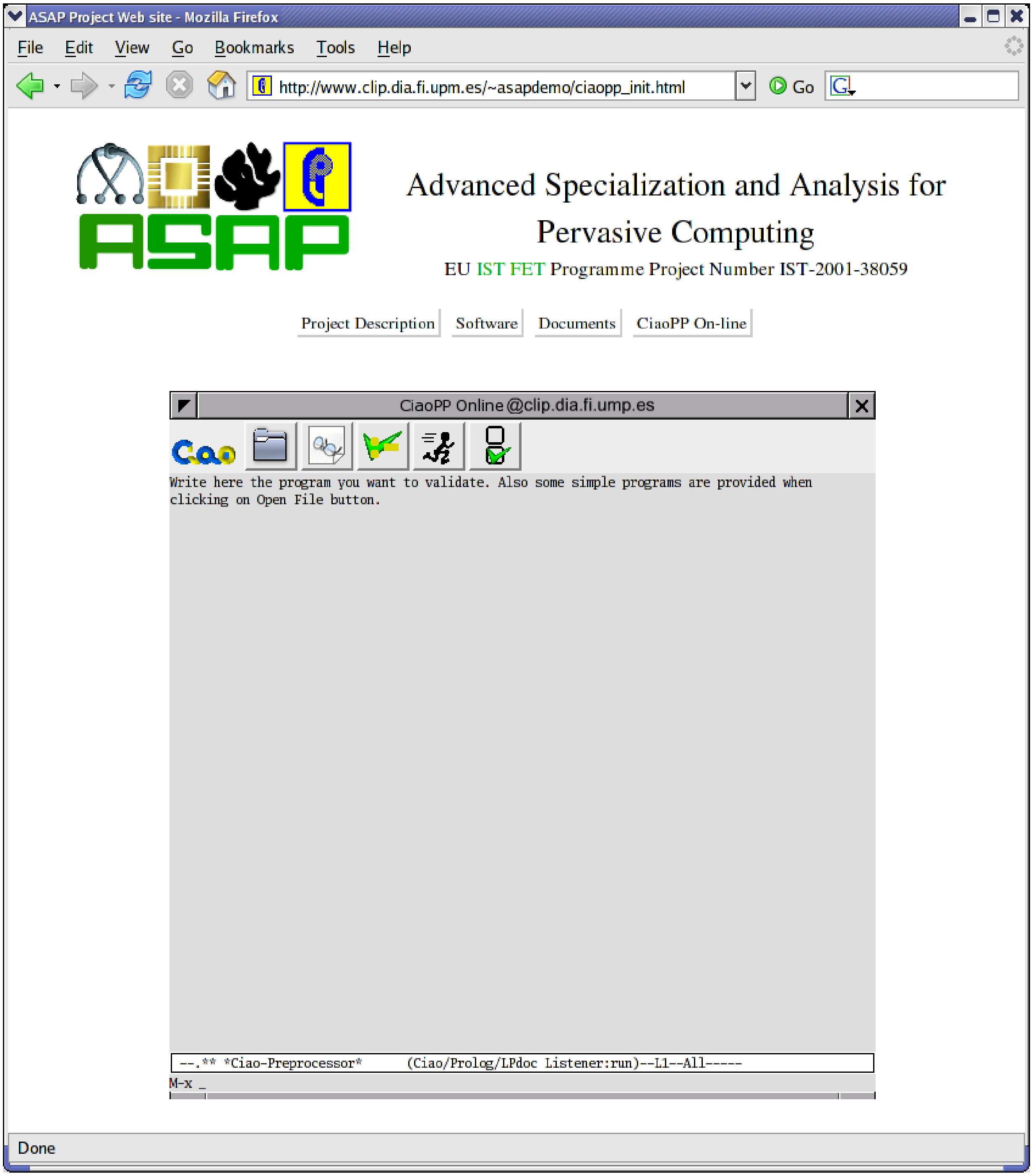
- The description of the project.
- The *Software* section which contains the first prototype tool (including manuals, downloadable installation packages, etc.), located at `http://clip.dia.fi.upm.es/Projects/ASAP/Software`
- The public deliverables produced by the Consortium in the context of the project in *pdf* format. These can be found at `http://clip.dia.fi.upm.es/Projects/ASAP/Documents`.

In this period we have added a fourth section in which we provide a first version of an on-line, web-based interface which allows performing some basic on-line testing and browsing of the capabilities of the tools. This allows interested parties to get a first flavor of their functionality and of the power of the program verification, validation and automated optimization techniques implemented, without having to go through the process of downloading and installing such tools. Some basic examples are also provided in order to simplify this process even further. In addition, the on-line interface allows users to upload their own programs for processing. The current on-line demonstrator allows to *analyze*, *optimize* and *check-assertions* (validate a program with respect to its assertions) with a default fixed option setting.

2 The on-line interface

Figure 1 shows the initial page of the web site of the first version of the demonstrator ("*CiaoPP Online*"). As mentioned before this page is now accessible from the main project web site. This on-line version of the interface to the tools has been implemented in Ciao using the PiLLoW web programming library [CH01].

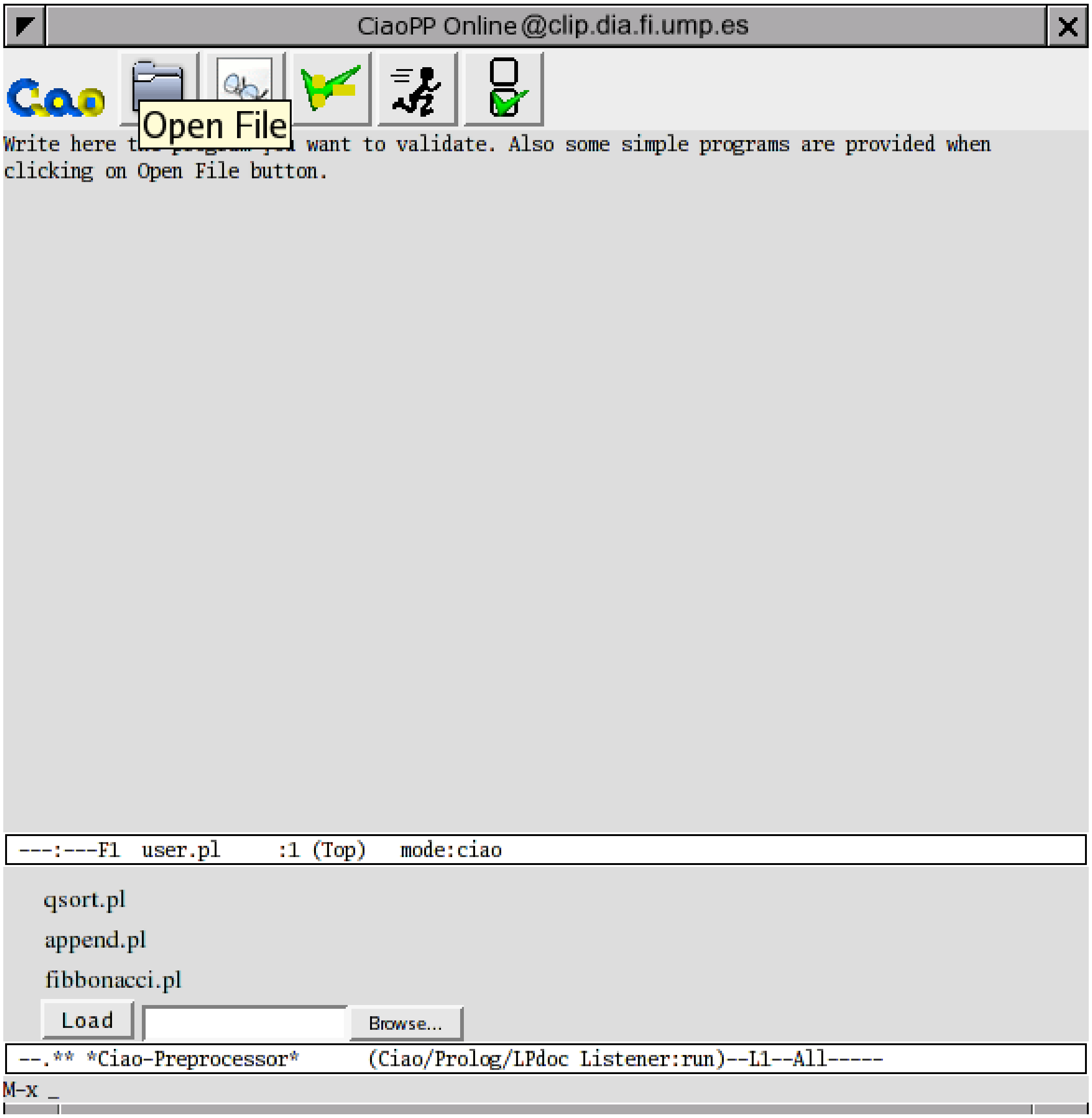
The user interface shown on the page consists of a toolbar and an area where the program to be processed can be entered and is displayed. Later we will see that this text area will be split


ASAP Project Web site - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.clip.dia.fi.upm.es/~asapdemo/ciaopp_init.html
Go
ASAP
Advanced Specialization and Analysis for Pervasive Computing
EU IST FET Programme Project Number IST-2001-38059
Project Description
Software
Documents
CiaoPP On-line
CiaoPP Online @clip.dia.fi.ump.es
Done

```
Write here the program you want to validate. Also some simple programs are provided when
clicking on Open File button.
```

```
--.** *Ciao-Preprocessor*      (Ciao/Prolog/LPdoc Listener:run)--L1--All-----
M-x _
```

Figure 1: Initial web page of the CiaoPP Online interface.



```
Write here the program you want to validate. Also some simple programs are provided when
clicking on Open File button.
```

```
---:---F1  user.pl      :1 (Top)    mode:ciao
```

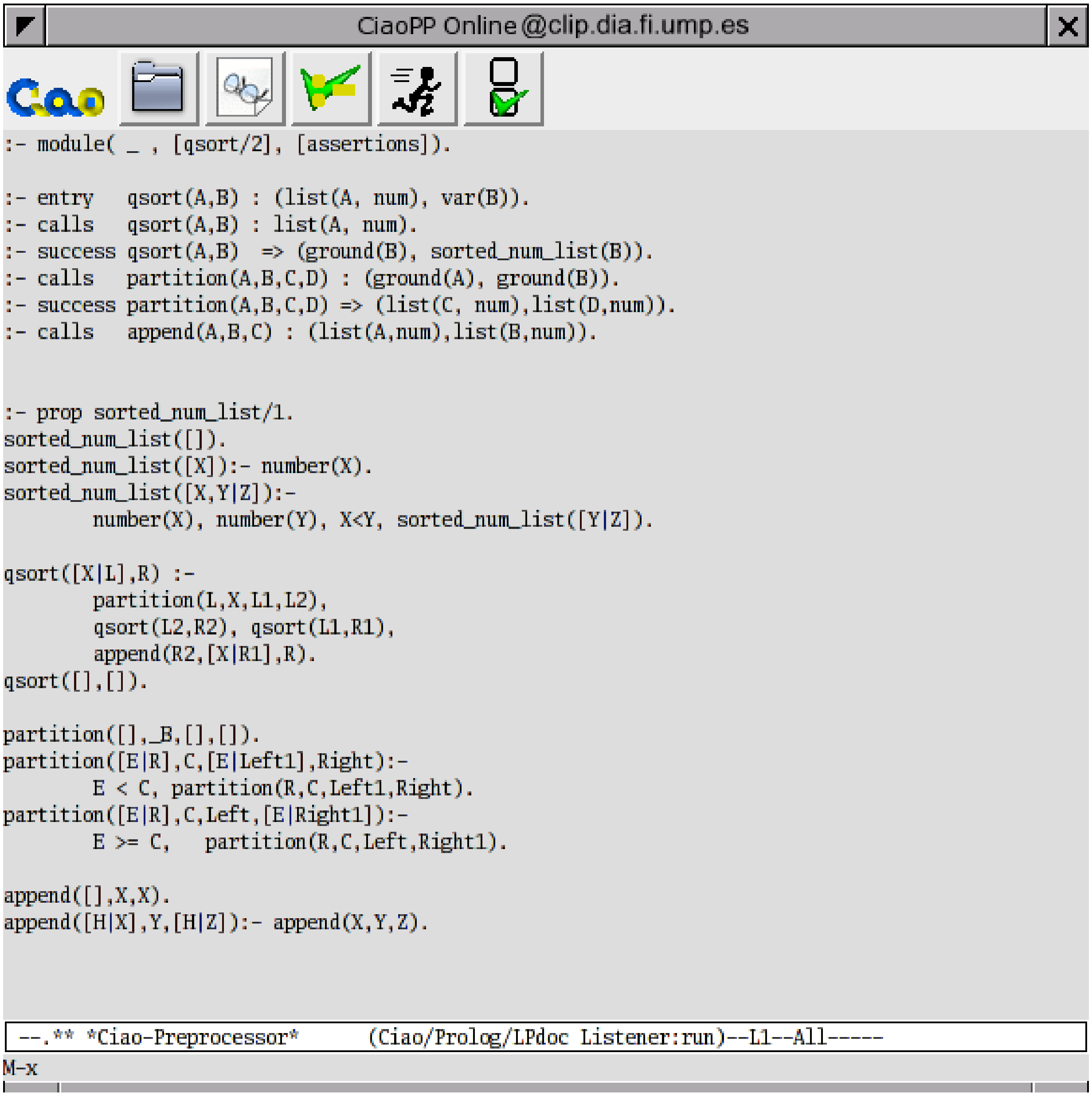
qsort.pl
append.pl
fibbonacci.pl

```
--.** *Ciao-Preprocessor*       (Ciao/Prolog/LPdoc Listener:run)--L1--All-----
M-x _
```

Figure 2: The interface to upload a file or load a predefined program.

in two parts, after the current program is already processed, which will allow also showing the error messages and the output from CiaoPP.

In addition to the user being able to write (or paste) the program in the text area above mentioned, the interface provides an *Open File* button in the toolbar which allows uploading a program from the user computer and also loading some simple predefined programs that are provided as examples and that allow to easily test the application. In Figure 2 we show the web interface after this button is pressed.



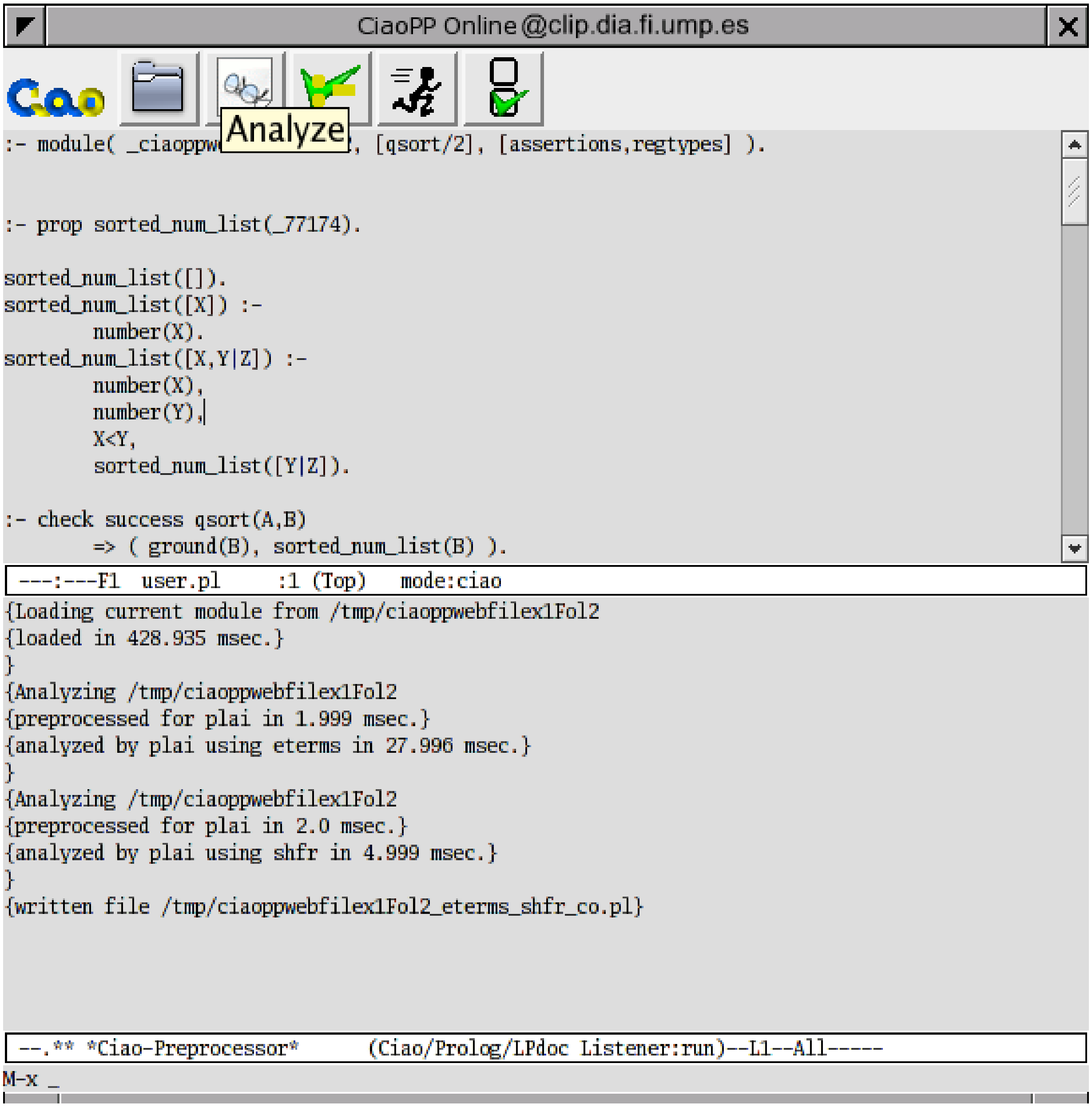
```
:- module( _ , [qsort/2], [assertions]).

:- entry   qsort(A,B) : (list(A, num), var(B)).
:- calls   qsort(A,B) : list(A, num).
:- success qsort(A,B)  => (ground(B), sorted_num_list(B)).
:- calls   partition(A,B,C,D) : (ground(A), ground(B)).
:- success partition(A,B,C,D) => (list(C, num),list(D,num)).
:- calls   append(A,B,C) : (list(A,num),list(B,num)).


:- prop sorted_num_list/1.
sorted_num_list([]).
sorted_num_list([X]):- number(X).
sorted_num_list([X,Y|Z]):-
        number(X), number(Y), X<Y, sorted_num_list([Y|Z]).

qsort([X|L],R) :-
        partition(L,X,L1,L2),
        qsort(L2,R2), qsort(L1,R1),
        append(R2,[X|R1],R).
qsort([],[]).

partition([],_B,[],[]).
partition([E|R],C,[E|Left1],Right):-
        E < C, partition(R,C,Left1,Right).
partition([E|R],C,Left,[E|Right1]):-
        E >= C,   partition(R,C,Left,Right1).

append([],X,X).
append([H|X],Y,[H|Z]):- append(X,Y,Z).
```

```
--.** *Ciao-Preprocessor*      (Ciao/Prolog/LPdoc Listener:run)--L1--All-----
M-x
```

Figure 3: The interface when a program has been loaded.

Once the user has loaded the program to be processed in the text area using the previously mentioned means (a situation shown in Figure 3), he/she can use the rest of the buttons of the toolbar to perform the following actions on the program in the text area:

Analyze *Analyzes* the program using a set of predefined options. These options has been selected to provide processing and output suitable for most users.



```
:- module( _ciaoppw                , [qsort/2], [assertions,regtypes] ).


:- prop sorted_num_list(_77174).

sorted_num_list([]).
sorted_num_list([X]) :-
        number(X).
sorted_num_list([X,Y|Z]) :-
        number(X),
        number(Y),
        X<Y,
        sorted_num_list([Y|Z]).

:- check success qsort(A,B)
        => ( ground(B), sorted_num_list(B) ).
```

```
---:---F1  user.pl     :1 (Top)    mode:ciao
```

```
{Loading current module from /tmp/ciaoppwebfilex1Fol2
{loaded in 428.935 msec.}
}
{Analyzing /tmp/ciaoppwebfilex1Fol2
{preprocessed for plai in 1.999 msec.}
{analyzed by plai using eterms in 27.996 msec.}
}
{Analyzing /tmp/ciaoppwebfilex1Fol2
{preprocessed for plai in 2.0 msec.}
{analyzed by plai using shfr in 4.999 msec.}
}
{written file /tmp/ciaoppwebfilex1Fol2_eterms_shfr_co.pl}
```

```
--.** *Ciao-Preprocessor*      (Ciao/Prolog/LPdoc Listener:run)--L1--All-----
M-x _
```

Figure 4: The interface after the *Analyze* button is pressed.

Check Assertions If the program has assertions written in the Ciao Assertion Language [PBH00], this process tries to prove whether each assertion can be reduced to true or false at compile-time, i.e., if the program can be statically *verified* or *bugs* can be statically detected. The results of the process are reflected in the output portion of the display. If an assertion cannot be proved or disproved, then it is printed as a *check assertion*, which can then be optionally transformed to a run-time check in a later process (this currently needs the use of Customize Options –see below).

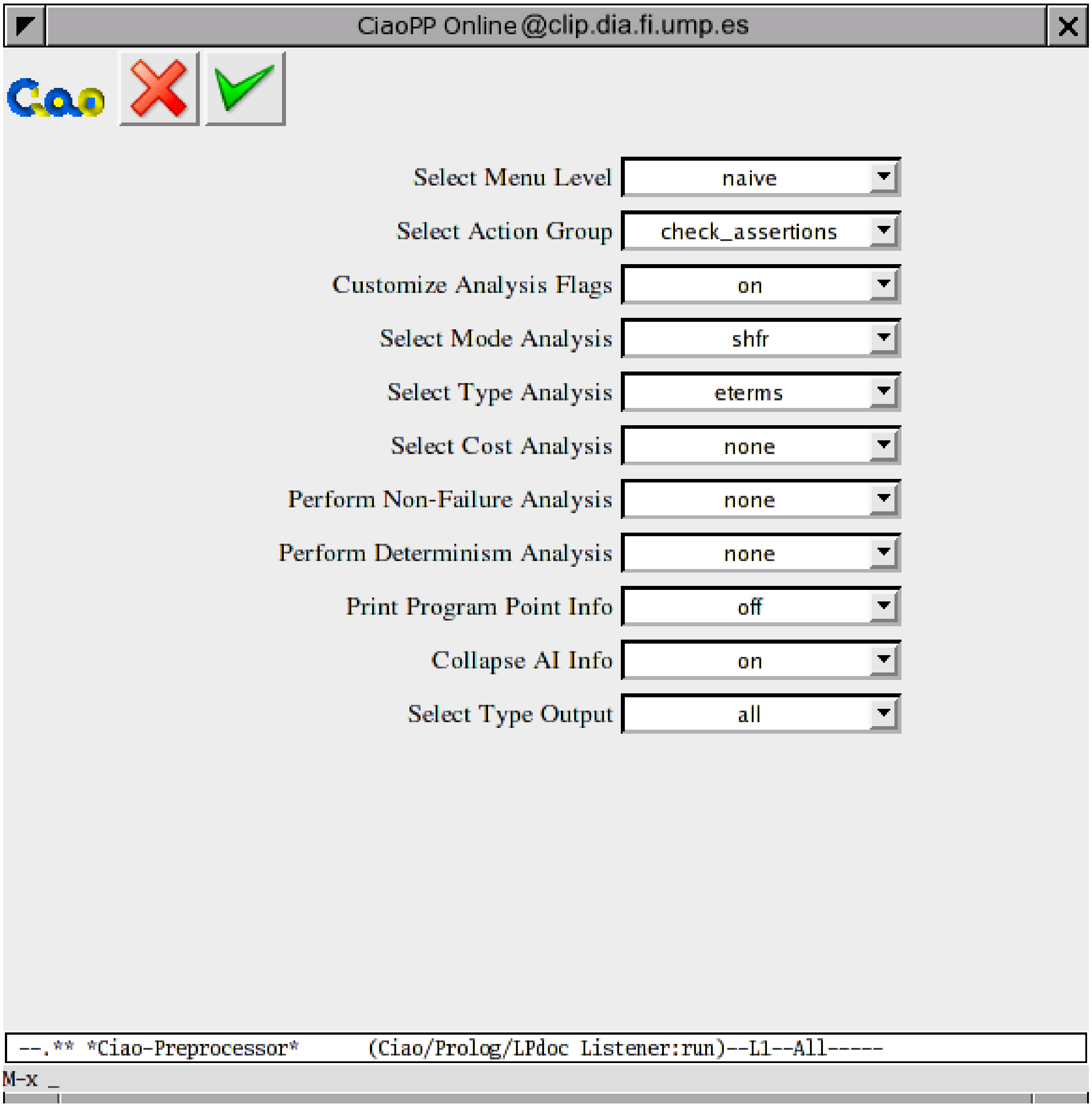
Optimize Tries to *optimize* the program using predefined options, which are aimed at producing acceptable results in most cases.

Customize Options This button provides the possibility of changing the options that are taken into account in the previous cases. The number of options is very large, since the tool has a very large set of capabilities. To simplify the process there are two levels of options: novice and expert.

Figure 4 shows the interface layout after the *Analyze* button is pressed. The text area of the interface has been split into two parts. The upper part contains the result program after it has been analyzed, where, for example, some assertions expressing the results of analysis have been added. The lower part displays the messages printed during the analysis process, which include possible error or warning messages. Note that the code in the upper part can at this time still be processed further using the other buttons available.

The *Check Assertions* and *Optimize* buttons produce analogous interface layout and results. Therefore, they will not be discussed further.

Figure 5 shows the interface as it looks after the *Customize Options* button is pressed. The text area has been replaced by a list of option names with a corresponding control box that allows the selection of the possible values for each option.



Select Menu Level	naive
Select Action Group	check_assertions
Customize Analysis Flags	on
Select Mode Analysis	shfr
Select Type Analysis	eterms
Select Cost Analysis	none
Perform Non-Failure Analysis	none
Perform Determinism Analysis	none
Print Program Point Info	off
Collapse AI Info	on
Select Type Output	all

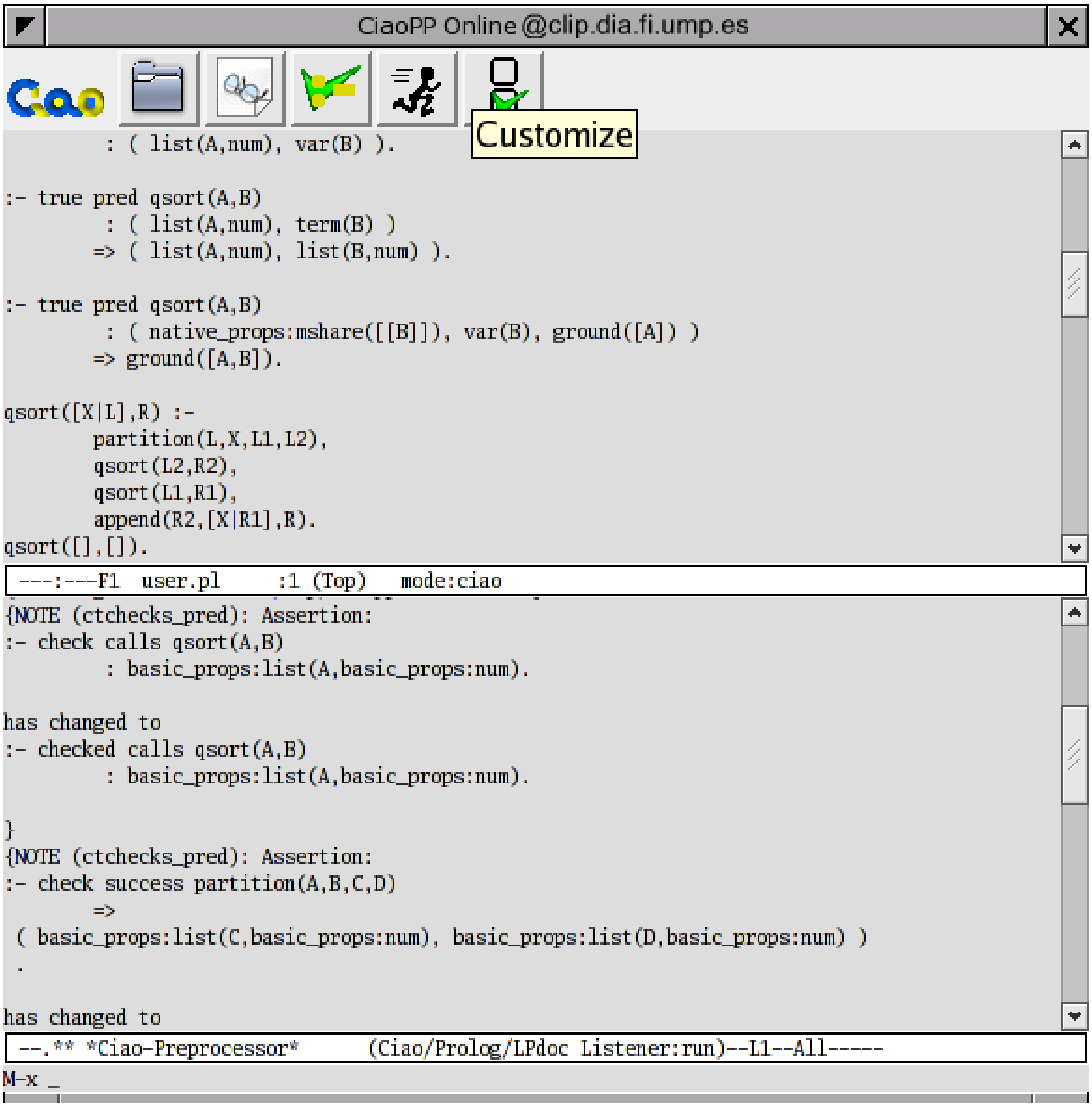
```
--.** *Ciao-Preprocessor*      (Ciao/Prolog/LPdoc Listener:run)--L1--All-----
M-x _
```

Figure 5: The interface to customize options.

An interesting and non-trivial characteristic of this option menu is that it is dynamic: because of the dependencies between options changing the value of an option can produce changes in the options below the selected one, such as hiding existing options or showing new ones. The toolbar changes as well to provide buttons for accepting or canceling the modified options.[1] This dynamic behaviour is implemented in JavaScript. The JavaScript program is generated automatically taking as input the internal (Prolog) configuration files that define de options and encode the dependencies among these options based on the capabilities of the tools. This has the advantage that it ensures that both the standard interface and the Web interface to the tools change automatically and identically as new options and capabilities are added to the toolset.

After accepting the modified options, the action selected in the *Select Action Group* option will be performed (as shown in Figure 6), using the program which resided in the text area before pressing the *Customize Options* button.

[1] The options and their possible values are explained in detail in the CiaoPP Reference Manual [BLGPH04].



```
         : ( list(A,num), var(B) ).

:- true pred qsort(A,B)
         : ( list(A,num), term(B) )
        => ( list(A,num), list(B,num) ).

:- true pred qsort(A,B)
         : ( native_props:mshare([[B]]), var(B), ground([A]) )
        => ground([A,B]).

qsort([X|L],R) :-
        partition(L,X,L1,L2),
        qsort(L2,R2),
        qsort(L1,R1),
        append(R2,[X|R1],R).
qsort([],[]).
```

```
 ---:---F1  user.pl      :1 (Top)    mode:ciao
```

```
{NOTE (ctchecks_pred): Assertion:
:- check calls qsort(A,B)
         : basic_props:list(A,basic_props:num).

has changed to
:- checked calls qsort(A,B)
         : basic_props:list(A,basic_props:num).

}
{NOTE (ctchecks_pred): Assertion:
:- check success partition(A,B,C,D)
        =>
 ( basic_props:list(C,basic_props:num), basic_props:list(D,basic_props:num) )
 .

has changed to
```

```
 --.** *Ciao-Preprocessor*       (Ciao/Prolog/LPdoc Listener:run)--L1--All-----
M-x _
```

Figure 6: The interface after customizing options.

3 Conclusions

The prototype on-line interface provided is already an important step towards making the ASAP toolset available on-line. This will in turn strongly contribute to the objective of dissemination of the results of the project. In the remaining of the project, the functionality of the current version of the on-line interface will be enhanced in several ways, covering the capabilities of all the tools in the tool set as they are progressively improved.

References

[BLGPH04] F. Bueno, P. López-García, G. Puebla, and M. Hermenegildo. The Ciao Prolog Preprocessor. Technical Report CLIP1/04, Technical University of Madrid (UPM), Facultad de Informática, 28660 Boadilla del Monte, Madrid, Spain, January 2004.

[CH01] D. Cabeza and M. Hermenegildo. Distributed WWW Programming using (Ciao-)Prolog and the PiLLoW Library. *Theory and Practice of Logic Programming*, 1(3):251–282, May 2001.

[PBH00] G. Puebla, F. Bueno, and M. Hermenegildo. An Assertion Language for Constraint Logic Programs. In P. Deransart, M. Hermenegildo, and J. Maluszynski, editors, *Analysis and Visualization Tools for Constraint Programming*, number 1870 in LNCS, pages 23–61. Springer-Verlag, September 2000.