
LPdoc
A Documentation Generator
for (C)LP Systems

Manuel Hermenegildo

herme@fi.upm.es

*School of Computer Science
Technical University of Madrid (UPM)*

CL2000

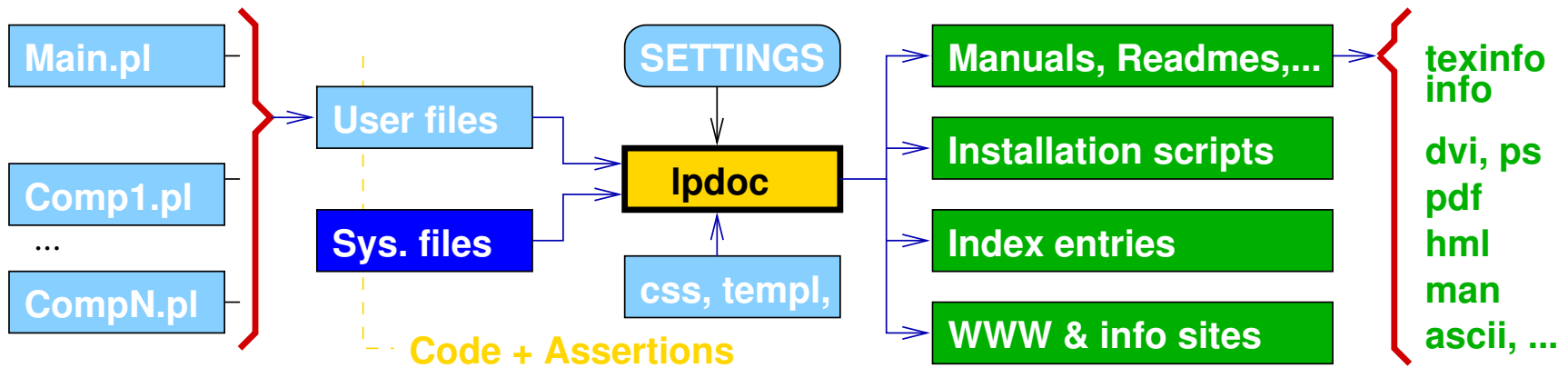
Imperial College, UK, July 28, 2000

(Work supported in part by Projects EDIPIA/ELLA/DiSCiPI)

Introduction / Motivation

- Writing and, specially, maintaining program documentation is hard
→ automate process as much as possible.
- Objectives:
 - ◇ Keep documentation close to source
(easy to keep in sync with the program – “Literate Programming”).
 - ◇ Be able to *reuse* typical program documentation.
 - ◇ Integrate closely with assertion language used in debugging/verification.
 - ◇ Produce useful documentation even if no comments or assertions in program.
 - ◇ Integrate in program development environment (e.g., version control system).
 - ◇ Allow complex manuals (indices, images, citations from BiBTeX dbs, etc.).
 - ◇ Support many output formats.
 - ◇ Perform several related tasks (e.g., construction of distribution sites).
 - ◇ Allow text reuse in multiple places (e.g., manuals, readmes, distribution sites, lists of manuals and sw packages, announcements, installation scripts, ...)
 - ◇ Be largely (CLP) platform-independent and modular.

Overall operation



- Can be done via menus from emacs interface.
- Or manually:
 - ◇ Creating manual:
 - * Edit SETTINGS file
 - * `lpdoc format` (dvi, ps, html, ...)
 - ◇ Viewing manual: `lpdoc dviview`, `lpdoc htmlview`, ...
 - ◇ Installing manual: `lpdoc install`
 - ◇ + cleanup, etc.

Inputs

- Basic types of input files:
 - ◇ Files to be documented (possibly including assertions and comments).
 - ◇ Used but not documented (library) files (e.g., system and user libraries: types, properties, reexports, etc.).
 - ◇ SETTINGS, template files, HTML style (css files), etc.
- SETTINGS:
 - ◇ Determines main file and components.
 - ◇ Defines the paths to be used to find files (independent of the paths used by the LPdoc application itself).
 - ◇ Selects indices (predicates, ops, declarations, properties, types, libraries, concepts, authors, ...), options, etc.
 - ◇ Defines location of BiBTeX file(s), HTML styles, etc.
 - ◇ Defines document installation location, WWW site, etc.

Assertions

- Assertions:
 - ◇ Written in the Ciao assertion language.
 - ◇ Declarations, used to:
 - * state general properties, types, modes, exceptions, ...
 - * of certain program points, predicate usages,
 - ◇ Includes standard compiler directives (`dynamic`, `meta_predicate`, etc.).
 - ◇ Have a certain qualifier: `check`, `true`, `trust`, ...
 - ◇ Can include documentation text strings.
- LPdoc understands assertions natively and uses them to generate the documentation.

Assertions (Contd.)

- Examples – pred:

```
:- pred qsort(X,Y) : list(X) => sorted(Y)
    # "@var{Y} is a sorted permutation of @var{X}."
```

- Examples – prop, regtype:

```
:- prop sorted(X) # "@var{X} is sorted."
sorted([]).
sorted([_]).
sorted([X,Y|R]) :- X < Y, sorted([Y|R]).

:- regtype list(X) # "@var{X} is a list."
list([]).
list([_|T]) :- list(T).
```

Comments

- Declarations, typically containing textual comments:

```
:- comment(CommentType, CommentData).
```

```
(also: :- doc(CommentType, CommentData).)
```

- Examples:

```
:- comment(title, "Complex numbers library").
```

```
:- comment(summary, "Provides an ADT for complex numbers.").
```

```
:- comment(ctimes(X,Y,Z), "@var{Z} is @var{Y} times @var{X}.").
```

- Markup language, close to LaTeX/texinfo:

- ◇ Syntax: *@command* (followed by either a space or `{}`), or *@command{body}*.

- ◇ Command set kept small and somewhat generic, to be able to generate documentation in a variety of formats.

- ◇ Names typically the same as in LaTeX.

- ◇ Types of commands:

- * Indexing and referencing commands.

- * Formatting commands.

- * Inclusion commands, etc.

Structure of generated documents

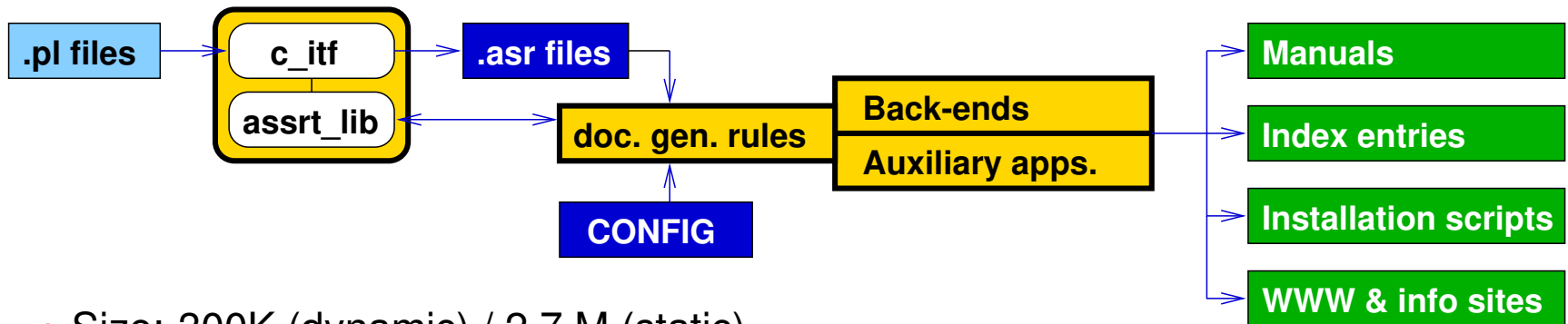
- Overall structure:
 - ◇ Single file → simple manual without chapters.
 - ◇ Multiple files:
 - * Main file gives title, author(s), version, summary, intro, etc.
 - * Other (“component”) files are chapters and appendices.
- Chapters:
 - ◇ If file does not define `main` → assumed *library*, *interface* (API) documented. else → assumed *application*, *usage* documented.
 - ◇ Structure:
 - * Chapter title/subtitle (or file name if unavailable).
 - * Info on authors, version, copyright, ...
 - * Chapter intro.
 - * Interface (usage, exports, reexports, decls, ops, modules used, ...).
 - * Documentation for decls, preds, props, regtypes, multifiles, modedefs, ...
 - * Bugs, changelog, appendices, ...

Documentation of predicates, props, etc.

- If no declarations or comments:
 - ◇ One line stating predicate name and arity
(useful: goes to index → automatic location, automatic completion).
 - ◇ If property or regtype: source code (often best description).
- Comments for the predicate/property/regtype...
- All assertions, described in textual form (unless stated otherwise).
- `pred` assertions documented as “usages”.
- Comments associated with `pred` assertions used to describe the usages.
- Syntactic sugar (e.g., modes) can be documented as is or expanded.
- The text in properties is *reflected* into the predicates which use such properties (also if property is imported from another module).

Architecture and Implementation

- Standalone application (Ciao standalone executable).
- Uses the Ciao generic modular program processing library (see the paper on the Ciao module system):
 - ◇ We want to be fully modular and incremental.
 - ◇ To support syntax extensions (ops, expansions, ...) the task requires a full reader, precise module visibility, etc.
- System is indeed quite incremental (vital for, e.g., the Ciao manual).



- Size: 300K (dynamic) / 2.7 M (static).
- 11K lines Prolog + 12K lines from Ciao libraries + 1K misc (html/css, BiBTeX, ...).

Comparison with other systems

- We are not aware of other systems with the capabilities of LPdoc.
- Some systems for pure “literate programming” in LP.
 - ◇ Quite useful, but almost all text must be written manually.
 - ◇ LPdoc goes much further and is much more automatic (“knows at least as much as the compiler”).
- Some automatic documenters with more limited capability (e.g., Icon, Perl, ...).
- Closest system is javadoc (developed in parallel with LPdoc):
 - ◇ Nicely formatted HTML manuals.
 - ◇ Also uses information typically available to the compiler.
 - ◇ Allows inclusion of textual comments in HTML format.

Disadvantages:

- ◇ Assrt. lang., treatment of props, markup, output formats, etc. richer in LPdoc.
- ◇ Perhaps too tied to HTML.
- ◇ Cannot show source code, as LPdoc.
- ◇ (+ the obvious one: tied to Java).

Conclusions

- In use at CLIP since late 1996 (and elsewhere) → some user experience.
- Very good for reference manuals in general. Also for “internals” manuals.
- Most satisfactory for libraries (highest quality documentation with least effort).
- Somewhat stilted for user’s manuals, but still useful.
- Much easier to maintain documentation up to date.
- With practice, one can with moderate effort write assertions and comments that:
 - ◇ document the program code,
 - ◇ produce a manual documenting the use of the code,
 - ◇ greatly improve the debugging and maintenance cycles (verification).

Writing assertions/comments more likely if effort pays off in several ways!

- All CLIP software manuals, web sites, etc. currently produced using LPdoc.
- Can be downloaded freely from <http://www.clip.dia.fi.upm.es/Software>.
- Can be adapted to other (C)LP systems and output formats.

System Demo

- LPdoc
- The Ciao preprocessor – Ciaopp
 - ◇ No assertions or comments.
 - ◇ Add assertions, comments.
 - ◇ Generate dvi, view
 - ◇ Add citations.
 - ◇ Generate html, view; info, view
 - ◇ Add a figure. View in several formats.
 - ◇ Manual cleanup for distribution/installation.
 - ◇ Visit Ciao manual, show help on current symbol.
 - ◇ Visit WWW site, collection of manuals.
 - ◇ Style sheets.

SYSTEM DEMO