A Hypertext Based Environment to Write Literate Logic Programs

Pierre Deransart (INRIA-Rocquencourt/France)
Roberto da Silva Bigonha
Patrick Parot (CSN/France)
Mariza Andrade da Silva Bigonha
José de Siqueira

Computer Science Department
Universidade Federal de Minas Gerais
Brazil

Abstract

Hyperpro is an experimental hypertext programming environment for Prolog based dialects and its application to some logic program development according to a logic programming methodology. The genericity of the tool makes it easily adaptable to other logic programming languages and to other applications in the field of logic programs development, in particular to handle logic programs with constraints.

1 The Software Documentation Problem

Software documentation is a perennial problem and a very important issue which still deserves research efforts. Almost all software systems evolve along their lifetime and thus require continuous maintenance, which is usually expensive and difficult to accomplish because most systems are poorly documented. Indeed, software documentation is often noexistent, incomplete or out-of-date.

Information contents of software documentation are naturally redundant, since the main purpose of any documented software is to provide at least two alternative views of the same material: the program view for the machine and the text view in a literate style for human consumption. The text view must also be organized in a way to provide different levels of abstraction of the documented software in order to help the understanding of large systems.

It is also a fact that program documentation tends to become a large collection files, which is prone to discourage programmers to keep the text part of the documentation up to date to the corresponding software. A good documentation system should then provide way to check automatically inconsistency of this kind, perhaps by creating strong ties between the documentation and its related pieces of programs.

2 The HyperPro System

We have designed a well-suited document generic structure for logic programs according to a recommended methodology in Logic Programming. With few alterations the prototype could be applied to other programming languages as we will see in the conclusion. The associated generic presentation has been designed according to criteria relative to the nature, the importance, the expected position of the elements in a program document, etc. Use of the prototype allows us to assess results that we present further, as well as evaluating the

impact and the relevance of our technique relative to "directed-methodology programming" in the process of logic program developments.

HyperPro is based upon the Grif-Thot [3, 4] tool, a powerful structured editor that includes hypertext features. It offers a way to handle two basic aspects: text editing and CLP programming. For text editing it uses the Thot system.

A HyperPro program is in fact a Thot document written in a report style. It also contains specific paragraphs which correspond to relation definitions. Their format reflects strictly the methodology required for CLP program development. The methodology is based on the works described in [20, 2, 1]. It uses simple basic principles such as in CLP the program unit is a packet of clauses characterizing a relation. Thanks to the declarative aspect of relational programming a relation definition may be understood just looking at the clauses and the informal definitions of the predicates used in the bodies of the clauses. The nature of the comments is obviously important because it must bring a redundant but different information. For such purpose different kinds of informations must be provided, which are precisely defined in the methodology. On the other side, the text editing system must provide facilities to navigate inside the program and its comments.

A whole program document is visualized in the integral document view. Other views may be specified in the generic presentation. Different conversion schema may be defined to export a program into different specific formalisms (*e.g.* LaTeX). Document exportation may be achieved upon views to collect information that can be used as input of various external systems as a Prolog evaluator, a spell checker, a theorem prover system, etc. The Application Program Interfaces (API) provided by Thot allows us to develop our specific applications that potentially could act on the editing document. The Thot toolkit is a comprehensive set of editing functions (written in C) that can be used for building the previously mentioned applications ; such functions perform operations on structured documents through the UNIX X-window environment.

A program (document) can be seen from different perspectives called views: each of them, specified in the generic presentation, is a way to visualize exclusively specific elements of the generic structure that are relevant for the programmer during a given stage of the development process ; for instance the user might want to focus on the clauses part of the program, or on the assertions or comments parts. Automatic synchronization of views allows the programmer to navigate on its document by pointing or selecting some chunks in any views. This aspect may be very relevant for large programs and facilitates "real-time" information retrieval. The user can work (write) in a specific view instead of editing the integral document since the editor itself will achieve real-time up-date of all the other views (as well as the integral document view). All the views can be opened simultaneously. Such features enhance flexibility and facilitate the program development process. Four kinds of views have been specified in HyperPro:

- Program view : allows to visualize exclusively the clauses parts (predicate definition) of the integral document.
- Comment view : allows to visualize exclusively the comments parts relative to the predicate definitions.
- Assertion view : allows to visualize exclusively the assertions parts relative to the predicate definitions.
- Typing view : allows to visualize exclusively the typing parts relative to the predicate definitions.

The program document model we contains relation definitions (elements of the generic structure) where the same predicate may be defined by several versions. The prototype offers possibilities to put links according two ways: 1)to point the current version of a given predicate definition in a relation definition block. 2)to relate a use of a predicate to its whole definition (that contains all the information about the considered predicate such as comments, assertions and predicate type.

The Hyperpro system allows the user select a chunk of program (packet of clauses) and requires to test it. After clicking on a special menu item the considered chunk is saved on a file, the system opens a window, calls a Prolog evaluator and loads the previous file. Then the user can perform any tests he wants within the Prolog evaluator. A possibility to load automatically all the current predicate definitions that often are necessary to test a part of code is beeing studied.

3 Main Results and Conclusions

The Hyperpor prototype allows to create and to elaborate homogeneous program documents according to the chosen programming methodology. This is a great advantage to control efficiently the different stages of the software developments and notably during the maintenance stage. Differents presentations can be available and the user may carry out some customizations to match its specific needs or tastes.

Hypertext features as hyperlinks allows the user to follow predicate definitions and to retrieve the current version of a predicate definition. Very useful to perform some tests when there exists several implementation versions of the same predicate definition.

The HyperPro approach allows texts, clauses and assertions to share a single document helps keeping consistent the software documentation throughtout their lifetime. Inconsistencies are not automatically checked, but we expect that the fact that the Prolog clauses and their corresponding explanatory texts are tigh together encourages maintenance people to keep software changes well documented.

Documentation of software must also be organized in a way to provide different levels of abstraction of the documented software in order to help the understanding of large systems. The synchronized viewing facilities provided by Thot, and used in the HyperPro system, permit the definition of levels of abstraction, which are the key to build large, understandable and maneageable documents. Furthermore, the view windows implemented in HyperPro also accept updating operations of their contents with automatic reflection in all others.

Literatur

[1] Pierre Deransart and Gérard Ferrand, *An Operational Formal Definition of Prolog: a Specification Method and its Application* , New Generation Computing 10 (1992), 121-171, 1992.

[2] Deransart, Pierre and Małuszyński, Jan, The MIT Press, *A Grammatical View of Logic Programming*, novembre, 1993.

[3] Quint, V. and Vatton, I., *Grif: an interactive System for structured Document Manipulation*, Proceedings of the International Conference on Text Processing and document Manipulation, 1986, November, 200-213, Cambridge University Press.

[4] Quint, V., *The Thot user manual*, Internal report, INRIA-CNRS, 1995.

[5] Abdelali Ed-Dbali and Pierre Deransart, *Software Formal Specification by Logic programming*, Logic Programming Summer School, Zurich, N. E. Fuchs and G. Comyn, 1992, Zurich, Suisse, Springer Verlag, LNAI, 636, 278–289, September.

[6] Deransart, Pierre and Ed-Dbali, Abdelali and Cervoni, Laurent, Springer Verlag, *Prolog, The Standard; Reference Manual*, 1996.

[7] Henrard, J. and Le Charlier, B., *FOLON: an Environment for Declarative Construction of Logic Programs*, PLILP'92, Leuven, Belgium, 217–231, 1992, August 26–28.

[8] Furuta, R. Quint, V. and André, J., *Interactively Editing Structured Documents*, Electronic Publishing, 1988, 1, 1, 19–44, April.

[9] Siqueira, J. de, *SEQUOIA: a theorem prover for counter model construction*, XVth conference of the Chilean Computer Science Society, Arica, Chile, 1995, August.

[10] Knuth, Donald D., *Literate Programming*, The Computer Journal, Vol. 27, No. 2, 1984, pp. 97-111.

[11] Knuth, Donald, *Literate Programming*, CSLI lecture notes, Stanford, CA, Center for the study of language and information, 1992, 27, 349–358.

[12] Quint, V. and Vatton, I., *Hypertext aspects of the Grif structured editor : design and applications*, Rapports de Recherche #1734, INRIA Rocquencourt, 1992, July.

[13] Quint, V., *Les langages de Grif*, Internal report (in french), INRIA-CNRS, 1992, May.

[14] Ramsey Norman, *The noweb Hacker's Guide*, Departament of Computer Science, Princeton University, September 1992 (Revised August 1994).

[15] Ramsey Norman, *Literate-Programming Tools Can be Simple and Extensible*, Departament of Computer Science, Princeton University, November 1993.

[16] Ramsey Norman, *Literate Programming Simplified*, IEEE Software, V.11(5), 97-105, September 1994.

[17] Ramsey Norman, *Literate Programming: Weaving a language-independent Web* , Communications of the ACM, 32(9): 1051-1055, September 1989.

[18] Richy, H., *Grif et les index électroniques*, INRIA Rocquencourt, 1992, October .

[19] Rizk, A. Streitz, N. and André, J., *Hypertext : concepts, systems and applications*, Proceedings of the European Conference on Hypertext, 1990, November, University Press.

[20] Renault, Sophie and Deransart, Pierre, *Design of Redundant Formal Specifications by Logic Programming: Merging Formal Text and Good Comments* , International Journal of Software Engineering and Knowledge Engineering, vol 4, No. 3, 1994, 369–390.

[21] Thimbleby, H., *Experiences of 'Literate Programming' using Cweb(a variant of Knuth's Web)*, The Computer Journal, Vol. 29, No. 3, 201-211, 1986.