SEPIA Programming Environment

Micha Meier
Philip Kay
Emmanuel van Rossum
European Computer-Industry Research Centre (ECRC),
Arabellastr.17, 8000 Munich 81, West Germany

Hugh Grant
ICL Information Technology Centre
South County Business Park, Dublin, Ireland

Abstract

We present the programming environment of the ECRC SEPIA system. SEPIA is an advanced Prolog system that allows integration of various extensions which may go beyond the logic programming paradigm. We describe here how its features were used to build its environment and what future work is planned to complete it.

1 Introduction

SEPIA (Standard ECRC Prolog Integrating Advanced Applications) [4] is a Prolog system developed at ECRC which serves as a basis for the integration of various extensions into a logic programming system. Its core is a WAM-based system with a fast incremental compiler, and a WAM emulator, extended by coroutining, event handling, modules, constructive negation and extendible data types.

Its architecture has been designed so that it allows various extensions to be integrated into it, among which are

- the constraint propagation system CHIP [6]
- the object oriented system PHOCUS [3]
- an interface to the INGRES database SEDUCE based on the EDUCE system [5]

- a recursive query answering component for deductive DBMS DedGin* [2] using the BANG file system [7]
- a windowing environment KEGI

This paper describes more in detail the KEGI environment for SEPIA and ongoing and future work on KEGI and SEPIA environment.

2 SEPIA Environment

The KEGI (Kernel ECRC Graphic Interface) system consists of three parts:

- The working environment, which is a windowing user interface to the SEPIA system.
- The 2D graphic package which is used to generate high quality graphic output.
- An object-oriented interactive graphic system based on ICL PCE [1].

2.1 Background

The KEGI project initially investigated the various tools which were available to provide working environments and interactive graphics for Prolog systems. PCE was obtained as being the most readily available and applicable for doing such a task. Conventionally PCE runs as a separate process and can be interfaced to any other process using a suitable connection. Such connections have been made to *Quintus*^{TM} Prolog, BIM Prolog, C Prolog and variants of LeLisp. Also a number of builtin predicates were developed using the CGI library to compliment the PCE system and to provide the Prolog programming with high quality 2D graphic output.

The SEPIA connection to PCE was implemented using external predicates written in C together with supporting Prolog predicates. However, the methodology adopted was in many respects different to that of the various systems mentioned above.

Conventionally the connection between PCE and the Prolog system is synchronous, any events that occur in the PCE windows will be passed to Prolog system and queued only when it is explicitly required. This has the disadvantage that if a user has any interaction with an application using PCE he cannot interact with the Prolog system via the keyboard or other applications at the same time. Equally if the Prolog system is servicing commands from the user input then he cannot interact with the PCE application.

Using SEPIA it was possible to overcome this handicap. SEPIA has the functionality to handle asynchronous events i.e signal interrupts. The connection between PCE and SEPIA uses a socket, this socket was opened in such a mode that SEPIA receives a signal as soon as there is some data available to be read in from the socket. When the user presses on a button, PCE writes a corresponding message on the socket which will cause SEPIA to interrupt whatever it was currently doing and invoke the interrupt handler to read the contents of the socket and call the corresponding predicate to service the button press. A further advantage was gained by the facility of using the socket connection to run SEPIA remotely and yet still have PCE display graphics on the users local machine. By this means PCE was able to provide graphic facilities to the SEPIA programmer.

Also, now that we had asynchronous handling of events it was possible to provide a user environment for SEPIA constructed using PCE. In this way the SEPIA environment was built. When SEPIA was invoked and the user called the goal **?- pce_env**, the PCE process was started, the communication socket was set up and a top level control window was created that contained a number of predefined and user definable buttons and pulldown menus. Such buttons were used to call builtin predicates such as **?- trace**, **?- ls** and **?- env** and pulldown menus were used for recalling the history of previous calls, displaying the names of Prolog files, current predicates, modules and avaliable libraries. Further windows could be opened for specific tasks such as to edit files edit windows, show the on-line manual, an interface to the SEPIA debugger and shell windows. All of these could be initiated from a button press on the control window.

Furthermore, the system could be modified by the user by providing a startup file or as an experienced user he could modify the environment to suit his requirements by directly modifying the Prolog library file which was compiled when the environment was initialised.

While very interesting from the theoretical point of view, the KEGI environment built up using the PCE system was not satisfactory from various reasons:

- The interface was slow and it used up much memory. This was due to the fact that an additional PCE layer was put between Prolog and the $Sunview^{TM}$ system.
- Since the PCE system was available only for the Sun^{TM} machines, it would have been difficult to port this environment to other machines, which is a necessary condition for ECRC prototypes, as it is sponsored by three major European companies who use different hardware.
- The system was not robust enough for controlling full user interaction.

To avoid these restrictions the emphasis of the KEGI development was changed. X11 had always been recognised as the future foundation for the system as this would guarantee hardware independence. However, at this time X11 was very much in its infancy. In particular for users of SEPIA the system was unstable and lacked many tools. Also the X11 toolkits from Athena, Ardent and HP were very unstable and had not yet reach maturity.

It was decided that the SEPIA working environment would be further developed using the SunView window libraries. The connection to the PCE system and the CGI library for user interaction and programming would also be maintained. All systems would be ported to X11 at a later date using the XView toolkit when this was made available and X11 more commercially robust.

2.2 Current State

The SunView implementation was developed using the many lessons learnt from the first phase of the project and has now a significant internal user base.

Figure 1: *kegitool* window with the on-line manual and editor window

The current features of the KEGI system include the following:

- Control panel, on line manual, editor windows and an interface to the debugger. This is now called *kegitool.*
- Full asynchronous interaction with the PCE system

Figure 2: A 2D demo program in *kegitool*

- CGI 2D graphic output

The kegitool control panel and SEPIA tty window are contained in the same frame. Interaction with SEPIA can be made either by the keyboard or by the mouse, button or menu input. Predicate calls are maintained in a history menu. Files can be displayed in a pulldown walking menu (in reality all the files on the whole network can be displayed in this walking menu).

Common actions can be carried out on selected files such as compiling, editing (in which case an editor of the user's choice is started), printing etc. Furthermore, the user can select text from anywhere on the screen and have this compiled automatically. The names of previously compiled files, known modules, user predicates and available libraries are maintained in further pulldown menus for user intercation. The user can initiate the debugger, on-line manual, define user buttons and quit or restart execution of the underlying SEPIA process via a number of buttons. Currently kegitool is being ported to X11.

Figure 3: A PCE demo program

The PCE system has been extended by the use of a graphic predicate layer to perform common user functionality. In total there are 62 predicates which amongst other things create warning or alert boxes, load icons and cursors, create and display trees and template windows. Objects can still be created and manipulated by using the predicates **new, send** and **get**. In this way SEPIA benefits from an object-oriented graphic system. Currently the PCE system is being ported to X11 using the Xview toolkit. It is anticipated that users will not have to make any changes to their Prolog code to use the new system.

The 2D graphic output library is a collection of 63 predicates to perform high speed colour output. This allows users to draw lines, solids text etc in many forms and colours. The 2D predicate library has been ported to directly use the X11 Xlib functions. Applications which were developed using the CGI implementation did not have to be changed.

The expected completion date for the X11-based SEPIA environment is

Figure 4: PCE on top of X11

at the end of 1989.

3 Debugging in SEPIA

As mentioned above, SEPIA has no interpreter, even the asserted clauses are compiled, and it is possible to use the conventional debugger on the compiled code without any restrictions. The advantage of such a scheme is that since there is no interpreter, the extensions that modify the Prolog system, e.g. the unification procedure, do not have to modify both the compiler and the interpreter, modifying the compiler is sufficient. The SEPIA incremental compiler is written in C for efficiency reasons which has proven to be a good choice as the compilation speed is by order of magnitude higher than for compilers written in Prolog. This makes the system more user-friendly and speeds up the development of programs.

Another, not less important reason to have a debugger on compiled code, is that the execution of the debugged code is much faster than it is the case with interpreted programs and it also uses less space. Here some comparisons of the speed of compiled (i.e. not debuggable), debuggable (but with the debugger switched off) and debugged code (in leap mode) for the *naive reverse* example on a (loaded) SUN-3/60:

System	**compiled**	**debuggable**	**debugged**	**slowdown**
Quintus Prolog 2.0	80k	2.6k	0.7k	114
SICStus Prolog 0.3.1	21k	1.3k	0.2k	105
SB-Prolog[1]2.3.2	26k	6.7k	0.9k	28
Sepia 2.2a	55k	42k	4.7k	11

It can be seen that Prolog systems that use an interpreter are very slow in the debug mode, whereas for a compiler-only system the slowdown is much smaller. Another disadvantage of interpreted programs is the space consumption, which can be much higher than for the same compiled programs and this can in fact prevent debugging large interpreted programs.

The KEGI interface to the debugger consists of one window which is split into several parts: the source file window, a command button panel, an output window and a control window which contains a series of buttons for each port and leashing mode.

The SEPIA debugger is a full four-port debugger, but it can display several additional ports:

- DELAY and WAKE
  The SEPIA debugger supports fully the coroutining features. When a procedure delays, its box is exited through the DELAY port and when it is woken again, the box is re-entered through the WAKE port. It is also possible to skip from the DELAY to the corresponding WAKE port.

- LEAVE
  This port is used to exit a procedure's box after a block exit (non-local jump) from the procedure or one of its children.

- NEXT
  This and the remaining ports do not show entering or exiting the box of a procedure but rather some actions inside it. The NEXT port is traced when a clause fails and the execution continues to the next clause of the same procedure. This port is missing in the original Byrd debugger and it makes debugging much easier when it is provided.

- CUT
  The CUT port is traced for all goals whose alternatives are discarded

[1]We do not consider here the SB-Prolog's facilities for tracing compiled code as they do not offer the functionality of a full four-port debugger and are not sufficient for sophisticated debugging

Figure 5: The debugger window of *kegitool*

inside a cut. This is a very convenient way for the programmer to see if the cuts are really necessary.

- UNIFY
  This port is traced at the end of the head unification so that the resulting bindings can be seen.

Apart from the standard leashing, the SEPIA debugger allows more detailed filtering of the tracing information, any port and any procedure can be either printed and the debugger stops there, only printed, or not shown at all. At the debugger prompt the user can issue many commands, e.g. various *skip* commands that allow to omit a part of tracing and resume it at a specified place which can be e.g. a goal with a specified invocation number, a specified level, intsantiation of a variable etc.

The debugger allows the user to look not only at the ancestor goals but also to any other previous goal, i.e. at the whole current execution tree. We plan to extend this feature with a graphical interface which would show the shape of the complete tree and allow to zoom on its parts using the mouse. Currently, when a port is printed in the debugger, the KEGI debugger interface shows the definition of the corresponding clause in its source file. This feature will be extended by showing precisely the position of the current subgoal in the source file.

4 Tools for Analysis and Performance Measurement

SEPIA has a statistics library file that extends the debugger to measure the activity at the procedure level. For every procedure the main ports are counted and they can be displayed in a table:

```
 PROCEDURE         #    MODULE    #CALL    #EXIT    #TRY    #CUT   #NEXT   #FAIL
is                /2    sepia_k       3        3       0       0       0       0
write             /1    sepia_k       6        6       0       0       0       0
move              /2    sepia        17       51      17       0      10       7
path              /4    sepia        17       16      17       0      15      17
fail              /0    sepia_k      22        0       0       0       0      22
!                 /0    sepia_k      43       43       0       0       0       0
opp               /2    sepia        51       51       0       0       0       0
safe              /1    sepia        51       34      51      23      28      17
not_member        /2    sepia       139       95     123      18     105      44
|TOTAL:     PROCEDURES: 20          368      320     210      43     158     109
```

This tool can be used both for low-level optimizations of the abstract machine and for user source-level optimizations.

By loading another system library file it is possible to produce statistics of the executed abstract instructions and store or add it into a file.

References

[1] A.Anjewierden. PCE-Prolog 1.0 reference manual. Technical report, University of Amsterdam, October 1986. ESPRIT Project 1098.

[2] A.Lefebvre and L.Vieille. Bases de donnees deductives et DedGin*. In *Proceedings of the AFCET Conference on Databases*, Paris, December 1988. invited paper.

[3] P.Dufresne D.Chan and R.Enders. PHOCUS: Production rules, horn clauses, objects and contexts in a unification-based system. In *Programmation en Logique, actes du Seminaire*, pages 77–108, Tregastel, France, May 1987.

[4] Micha Meier et al. SEPIA - an extendible Prolog system. In *Proceedings of the 11th World Computer Congress IFIP'89*, San Francisco, August 1989.

[5] J.Bocca. Educe: A marriage of convenience: Prolog and a relational dbms. In *Proceedings of the 3rd Symposium on Logic Programming*, pages 36–45, Salt Lake City, September 1986.

[6] M.Dincbas, P.Van Hentenryck, H.Simonis, A.Aggoun, T.Graf, and F.Berthier. The constraint logic programming language CHIP. In *International Conference on FGCS 1988*, Tokyo, November 1988.

[7] M.Freeston. The BANG file: a new kind of grid file. In *SIGMOD '87*, San Francisco, 1987.