Transactions on Information and Communications Technologies vol 10, © 1995 WIT Press, www.witpress.com, ISSN 1743-3517

Prolog as an intermediate metafile format within the software engineering life cycle

M.P. Lee
Computing Information Systems Management Group, School of Defence Management, Royal Military College of Science, Cranfield University, Shrivenham, Swindon, Wiltshire, SN6 8LA, UK

Abstract

This paper seeks to argue that Prolog can act as an intermediate metafile format within the software engineering lifecycle. It begins by considering graphical metafiles and then goes on to look at a computer-aided software engineering (CASE) tool which already uses Prolog as such a metafile. This concept can however be taken further. The rest of the paper looks at how Prolog can be used both within and between the various stages of the software engineering lifecycle.

1 Introduction

Prolog has traditionally been seen as a fifth generation programming language (5GL) for artificially intelligent (AI) applications. This view has been promulgated by a series of textbooks which have concentrated on AI examples with little coverage of problem solving, program design and commercial applications.

The recent appearance of books by Goble (1989), on structured systems analysis through Prolog, and Malpas (1987), on Prolog as a relational language, have started to redress the balance. In fact Prolog has several of the attributes of a fourth generation programming language (4GL). In particular it is a relational query language with a built-in database. The relationship of Prolog to databases has led to the publication of a number of books (Gray 1984, Gray & Lucas 1988, Li 1984, Lucas 1988 and Marcus 1986) and a large number of papers.

Prolog certainly provides a contrast to traditional third generation languages (3GLs). It is declarative rather than procedural, interpreted rather than compiled, symbolic rather than numeric. It is pattern driven rather than control directed, recursive rather than iterative, and search intensive rather than computation intensive. Finally, it is polymorphic, bidirectional and deductive (Lee et alia 1991).

This paper seeks to argue that Prolog can act as an intermediate metafile format within the software engineering lifecycle. It begins by considering graphical metafiles and then goes on to look at a computer-aided software engineering (CASE) tool which already uses Prolog as such a metafile. This concept can however be taken further. The rest of the paper looks at how Prolog can be used both within and between the various stages of the software engineering lifecycle.

2 Metafiles

In the early days of computer graphics a program would produce output on just one hardware device. However with the development of various graphical displays and hardcopy devices it made sense to separate graphical programs from output devices.

This was achieved by making the drawing program into a device independent "pre-process" which wrote its output into an intermediate metafile which could be post-processed for a particular target device.

This meant that any new display device could be used with existing programs merely by writing a driver which translated from the logical metafile format according to the physical device characteristics. Perhaps the best example of a graphics metafile was provided by GHOST (Prior & Sutherland 1982).

Today metafiles can be classed into two categories depending on whether they are proprietary to one company (de facto standards) or by some independent body (de iure standards). Manufacturer independent examples include the Computer Graphics Metafile (CGM) and the Initial Graphics Exchange Specification (IGES), while proprietary formats include PostScript (Adobe Systems), Windows Metafile Format (WMF, Microsoft Corporation) and Drawing eXchange Format (DXF, AutoDesk). Lee & Soper (1992) consider the Windows Metafile Format as a tool for software integration, whilst Francis (1991) provides a good comparison of CGM and PostScript.

PostScript, like Prolog, is a programming language. However it is not usually programmed by a human, but instead acts as a concise communication medium between computers and devices. In some respects this is also true of the

Structured Query Language (SQL). Increasingly SQL is not being used as an end-user or even a programmer query language, but instead is acting as a concise communication medium between client and server computers over networks.

Valdes (1991) discusses the emergence of this class of small programming languages with large underlying software 'engines' (table 1).

Table 1: Small languages with large engines

Language:	Engine:
Prolog	Backward chaining inference (Bratko 19986)
Postscript	Interpreter in a laser printer (Francis 1991)
SQL	Data Base Management System
Lotus macros	1-2-3 Spreadsheet (Soper & Lee 1990)
AutoLisp	AutoCAD Computer Aided Design Graphics
Unix shells	Unix operating and file management system

Prolog is a relatively small programming language since it does not have keywords such as 'begin', 'end', 'var', 'while', 'then' or 'else'. It has no type declarations such as integer or array. The only data structures provided are static records (stored as facts) and dynamic lists. Lastly the language does not provide control structures like for/do or while loops.

In many respects Prolog is like a expert system shell since it provides a user interface (albeit command driven) to an underlying backward chaining inference engine with built-in pattern matching. Initially the system is empty of programs and data.

Already at least one product, MacCadd, has adopted Prolog as a graphical metafile, to which we turn next.

3 MacCadd

MacCadd is a Computer Aided Software Engineering (CASE) tool which produces a variety of design diagrams on Apple Macintosh computers. In particular it supports data flow diagrams (DFD), state transition diagrams (STD), structure charts (SC) and entity relationship diagrams (ERD). Furthermore it can be customised, via a variant definition file (VDF) facility, to meet specific diagramming conventions (Patterson 1990).

Table 2: MacCadd Diagram Prolog Facts

Prolog facts:	Definition:
a (arrow_id, source_symbol_id, destination_symbol_id, arrow_type_id, top_left_x, top_left_y, bottom_right_x, bottom_left_x).	Arrow
c ("Text").	Comment
d (symbol_id, "Diagram register name").	Diagram
f (arrow_id, font_number, style_bits, font_size, "Arrow text").	Flow
m (parent_symbol_id, factor1, factor2, top_left_x, top_left_y, box_visible_bit).	Magnification
o (diagram_type_id, parent_symbol_id, num_horizontal_pages, num_vertical_pages, page_size, orientation_bit).	Orientation
p (arrow_id, sequence_number, pivot_centre_x, pivot_centre_y).	Pivot
s (symbol_id, parent_symbol_id, symbol_type_id, top_left_x, top_left_y, bottom_right_x, bottom_left_x).	Symbol
t (symbol_id, font_number, style_bits, font_size, "Symbol text").	Text
v ("Variant Definition File name").	VDF name

MacCadd ensures that the syntax of a given diagram type and its underlying rules are enforced. It has an integral data dictionary whose contents can be output as a series of Prolog clauses (Jones 1988). This metafile format has three advantages: first it is human-readable and can therefore be verified by eye;

secondly it is in straight ASCII format for ease of portability to other computers and thirdly it can be readily analysed further either directly by using Prolog or else by using programs written in another language.

MacCadd stores a diagram as a series of Prolog facts. Each fact has a single letter mnemonic predicate name with various arguments (table 2).

As an example consider:

```
p (a00001,2,180,90).
```

where:

p	indicates a pivot clause,
'a0001'	is an arrow identifier (ID),
2	indicates that this is a second pivot on the arrow,
180	is the X coordinate of the pivot position,
90	is the Y coordinate of the pivot position.

The utility of MacCadd is a powerful argument for using Prolog as a graphical design metafile, however Prolog can be used both within and between other stages of the software engineering lifecycle apart from design (sensu stricto). We now go on to consider Prolog's utility for requirements analysis, specification, design, implementation, testing and maintenance in turn.

4 Prolog for requirements analysis

Prolog has two main strengths in the requirements analysis phase of the lifecycle: it can deal with natural language directly (Geetha & Subramanian 1990) and it can be used for rapid prototyping (Bratko 1989). The first aspect is supported by both the ease of coding of grammar rules into Prolog and the presence of a built-in pattern matcher. These allow subsets of any language, including English, to be parsed directly by a program.

Jay (1991) has used a Backus Naur Form (BNF) compiler, written in 270 lines of Prolog and utilising its definite clause grammar (DCG) facility, to parse AutoCAD's drawing exchange files (DXF) and then convert them into Prolog. Since DXF is a de facto industry standard metafile, this then allows Computer Aided Design (CAD) packages other than just MacCadd to produce output in the form of Prolog facts.

Rapid prototyping can be a powerful tool in helping to elicit users' requirements (Lee et alia 1990). Its use with Prolog has been advocated by various authors (for example Lea & Chung 1990, Rueher 1987 and Leibrandt & Schnupp 1984).

5 Prolog for specification

In some respects rapid prototyping is the antithesis of the lifecycle approach, but by turning a specification into an executable program they become complementary. In particular, Ramirez et al (1990) show how formal, mathematical specifications of data models can be directly implemented in Prolog.

Similarly, Lazarev (1989) shows how data flow diagrams (DFDs) can be directly mapped into an executable specification written in Prolog. He argues that Prolog's declarative nature is ideally suited to the analysis phase while it's procedural capabilities are well suited for software implementation by making a specification executable.

So far as the next two stages of the software engineering lifecycle are concerned, design and implementation, little more needs to be said. This paper has already discussed MacCadd as a CASE tool for software design and Prolog is, first and foremost, an implementation language. Suffice it to say that Prolog output from the specification stage can be directly used for design and code purposes.

6 Prolog for testing and maintenance

Some work has already taken place on the role of Prolog in the testing phase of the software engineering lifecycle. For example, Roper & Smith (1987) present a Prolog program for testing Jackson Structured Programming (JSP) designed programs and Singleton & Farris (1988) describe a Prolog system for software configuration management.

An example of Prolog used for maintenance purposes is by Pau & Kristinsson (1990) who describe a software maintenance system written in Prolog which assists in reverse engineering, documentation and error management.

7 Disadvantages of Prolog

Although this paper has argued for the increased use of Prolog both within and between the phases of the software engineering lifecycle, it is necessary to bear Prolog's deficiencies in mind. In particular Prolog suffers from a total lack of built-in modularity. All procedures exist in a global name space, even if all variables are entirely local to each statement. Prolog has not yet been standardised, but the National Physical Laboratory (NPL) is currently working on this point.

In terms of paradigms Prolog supports logic-based, relational and declarative styles of working, but not functional or object-oriented approaches. The latter

can be circumvented by using products such as LPA's Prolog++ with object-oriented extensions. Prolog can also be criticised for its poor debugging facilities, which are part of its currently inadequate support environment.

8 Further Research

Work is still continuing on the various uses of Prolog in the integration of the phases of the software engineering lifecycle. Another area for further research is in the association of Prolog with office automation and personal productivity software. This paper has already discussed the links that Prolog has with databases, text processing and computer graphics. Goble (1989) cleverly illustrates its relationship to spreadsheets (p 100-104) and statistics (p 127-130). This whole area, and the use of Prolog as a fourth, rather than fifth, generation programming language for commercial, rather than artificial intelligence, applications requires further study.

9 Conclusion

This paper has sought to argue that Prolog can act as a metafile format within and between the various stages of the software engineering lifecycle. In particular it has looked at MacCadd as an example of successful use of this concept. However the idea can be extended from design (sensu stricto) proper to design (sensu lato) in the large.

10 Acknowledgements

This paper is published with the permission of the Principal of the Royal Military Ciollege of Science (RMCS). This work developed out of an undergraduate project (Leach 1989). I am grateful to an anonymous referee and to Adrian Kent, Alan Harrison & John Pryce of RMCS for commenting on this paper.

11 References

Bratko I 1986 "Prolog programming for Artificial Intelligence" Addison-Wesley, 0-201-14224-4.

Bratko I 1989 "Fast prototyping of expert systems using Prolog" pp 69-88 in Guida G & Tasso C (eds) "Topics in expert system design" Elsevier, 0-444-87321-X.

Francis A H 1991 "Computer Graphics Metafile versus Postscript - 'horses for courses'" Computer-Aided Design 23, 4, 297-302.

Geetha T V & Subramanian R K 1990 "Representing natural language with Prolog" IEEE Software 10, 3, 85-92.

Goble T 1989 "Structured systems analysis through Prolog" Prentice Hall, 0-13-853581-7.

Gray P M D & Lucas R J 1988 "Prolog and databases: implementations and new directions" Ellis Horwood, 0-7458-0371-7.

Gray P M D 1984 "Logic, algebra and databases" Ellis Horwood.

Jay C 1991 "A BNF compiler for Prolog" AI Expert 6, 1, 35- 39.

Jones J 1988 "MacCadd, an enabling software method support tool" pp 132-154 IN Harrison M D "People and computers" Cambridge University Press, 0-521-33259-1.

Lazarev G L 1989 "Executable specifications with Prolog" Dr Dobb's Journal 14, 10, 61-68.

Lea R J & Chung C G 1990 "Rapid prototyping from structured analysis: executable specification approach" Information & Software Technology 32, 9, 589-597.

Leach A P 1989 "Using Prolog as an intermediate metafile format in the software engineering life-cycle" Unpublished BSc dissertation, Royal Military College of Science.

Lee M P, Darling M W M, Peacock D & Jeffreys S 1990 "Simulating user interfaces with dBASE III+" pp 181-195 IN Life M A, Narborough-Hall C S & Hamilton W I (eds) "Simulation and the user interface" Taylor & Francis, 0-85066-803-4.

Lee M P, Harrison A & Kent A E 1991 "Group projects for the software engineering of knowledge-based systems" pp 95-107 IN King G A (ed) "Proceedings of the First National Conference on Software Engineering in Higher Education" Southampton Institute, 1-874011-00-1.

Lee M P, Jeffreys S & Peacock, D 1989 "dBASE as a first programming language" Collegiate Microcomputer, 7, 2, 111-116.

Lee M P, Pryce J D & Harrison A 1994 "Prolog as a first programming language" pp 275-281 IN King G A et alia (Eds) "Proceedings of the First International Conference on Software Engineering in Higher Education" Southampton Institute, 1-85312-289-0.

Lee M P & Soper J B 1992 "Integrating and enhancing DOS programs using Windows 3" Collegiate Microcomputer 10, 4, 248- 254.

Leibrandt U & Schnupp P 1984 "An evaluation of Prolog as a prototyping system" pp 424-433 IN Budde R et alia (eds) "Approaches to prototyping" Springer-Verlag, 0-387-13490-5.

Li D 1984 "A Prolog database system" Research Studies Press 0-14-196153-7.

Lucas R 1988 "Database applications using Prolog" Ellis Horwood, 0-7458-0356-3.

Malpas J 1987 "Prolog: a relational language and its applications" Prentice Hall, 0-14-026373-9.

Marcus C 1986 "Prolog programming: applications for database systems, expert systems and natural language systems" Addison Wesley, 0-201-14647-9.

Patterson G 1990 "MacCadd v 5.01 - a review and assessment" Monitor 1, 1, 11-16.

Pau L F & Kristinsson J B 1990 "SOFTM: a software maintenance expert system in Prolog" Software maintenance: research & practice 2, 87-111.

Prior W A J & Sutherland R J 1982 "The GHOST-80 interactive metafile" p 217-224 IN Vandoni.

Ramirez R G, Choobineh J & Dattero R 1990 "Using logic programming for formal specification and validation of data models" Information & Management 19, 101-112.

Roper R M F & Smith P 1987 "A software tool for testing JSP designed programs" Software Engineering Journal 2, 2, 46-52.

Rueher M 1987 "From specification to design: an approach based on rapid prototyping" IEEE p126-132.

Singleton P & Farris C D 1988 "Software configuration management using Prolog" pp 340-356 IN Gray P M D & Lucas R J "Prolog & databases: implementations and new directions" Ellis Horwood, 0-7458-0371-7.

Soper J B & Lee M P 1990 "Statistics with Lotus 1-2-3" Chartwell-Bratt, 0-14-145870-3.

Valdes R 1991"Little languages - big questions" Dr Dobb's Journal 16,9,16-25.