APPLAB — A Laboratory for Application Languages

Elizabeth Bjarnason

LU-CS-TR:96-177
LUTEDX/(TECS-3073)/1-5/(1996)

Also published in: Proceedings of NWPER’96, Nordic Workshop on
Programming Environment Research, L. Bendix, K. Nørmark,
and K. Østerbye (eds), Aalborg, Demark, May 1996.
Aalborg University, Technical Report R-96-2019.

Department of Computer Science

Lund Institute of Technology
Lund University

Box 118, S-221 00 Lund, Sweden

APPLAB — A Laboratory For Application Languages

Elizabeth Bjarnason

Dept of Computer Science, Lund University
Box 118, S-221 00 Lund, Sweden
e-mail: Elizabeth.Bjarnason@dna.lth.se

APPLAB (*APPlication language LABoratory*) is an environment that supports the interactive development of application languages. The system allows the language designer to edit a language description and simultaneously edit a program in the new (changing) language. The system also supports the design of static semantics for the language. APPLAB is used in a case study looking at the different levels of programming involved in programming industrial robots.

1 Introduction

Designing a new language is an iterative process where language constructs are designed and tested in a prototypical fashion. Application-specific languages, i.e. languages designed to be used within a specific application domain, are often revised and extended as new needs arise, as opposed to general-purpose languages which are rarely changed. When working with application domain languages it is desirable to have an environment that supports the process of designing and extending languages in an iterative and integrated fashion. We call such an environment a *language laboratory*. The environment should allow the language designer to switch freely between editing the language description and editing a program in the (changing) new language. Examples of such systems include GIPE [Kli91], DOSE [FJS86], and Orm [MHM+90]. An example of an environment that is designed to support the language design process, but does not offer immediate feedback on editing operations performed on the language description is TaLE [JKP91, JK93]. Our language laboratory, APPLAB [Bja95] (*APPlication language LABoratory*) is an extension of the grammar environment part of Orm. Like Orm, it is based on *grammar interpretation* [MH93], i.e. the grammar of a language is interpreted to supply language-specific behaviour to the system, thus avoiding a compilation and linking phase after each grammar change to keep the language-specific features of the system consistent with the current grammar description.

2 Editing in APPLAB

The SbyS editor [Min90] originally developed within the Mjølner/Orm project [KLMM93] is used in APPLAB. When editing a program the SbyS editor supports the user in editing language constructs defined by the grammar given for the language. The grammar itself can also be edited in APPLAB using an instance of the same structure editor, since it is expressed in a language described by a (meta-)grammar. When performing structure-oriented editing, constructs are inserted into a program by selecting them from a menu. The menu always contains exactly all the available constructs at the current position in the program as described by the grammar. Any construct of the language can also be edited as text using an instance of the *Grammar Interpretive Parser*, GRIP [BH96]. The edited text is then mapped to the corresponding language construct and inserted into the program.

The editing style provided by a grammar-interpreting structure editor is ideally suited for application languages. The grammar-interpreting features allow an instant feedback on new grammar rules, thus allowing the language designer to work in an experimental fashion, trying out new language constructs interactively, as they are defined. Programming applications using an application language is, or could be, done by someone who is more familiar with the application area than with actual programming. Also, since application languages often are changed and expanded, a structure-oriented editor is of great use to an application programmer. It relieves him of having to remember the correct syntax and constructs of the current version of the application language.

3 Static Semantics in APPLAB

One of the major advantages of using an application language when programming in a specific application domain is that the conventions for using lower level library routines can be expressed as static semantic rules for the application language instead of having to rely on the programmer to correctly use the library routines [Hed96]. APPLAB lets the language designer add static semantics to a language by means of an attribute grammar notation. Some initial support for this is already implemented. Our goal is to give full support for specifying Door AG:s [Hed92], an extended AG formalism suitable for efficient incremental evaluation. The Door AG will be interpreted and the defined semantic checks performed as the application programmer edits a program written in the application language.

4 Case Study: Robot Programming

We are currently involved in a case study in cooperation with the Department of Automatic Control at Lund University, looking at the problems involved in programming industrial robots. APPLAB is used both to design an application language suitable for robot programming, and as a programming interface using the designed language.

When programming industrial robots there are several different levels of programming involved. The *manufacturer* of the industrial robot delivers the product together with a programming interface for the basic motion control of the robot. The *application developer* then uses this interface to specialize the robot for a specific application, e.g. welding or gluing. At the moment this phase is most often done by the manufacturers themselves, since there are so many conventions to follow when using the motion-control interface that it is not considered safe to release the code of the interface for general use. The resulting, specialized robot is delivered to a customer who wishes to use the robot to manufacture some product. The customer then programs the robot to perform the required task. E.g. to weld a hull of a ship according to data from a CAD/CAM system. See figure 1, and [Nil92] .

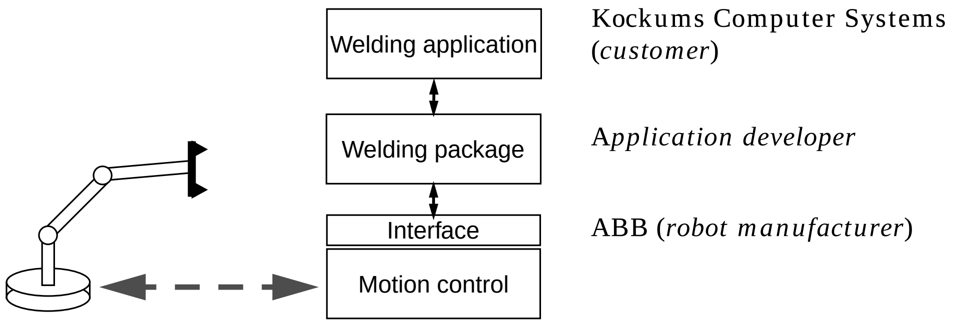

Figure 1 *The Different Levels Of Robot Programming*

Our idea is to introduce an application language for each level of programming. The application language for each layer should contain the constructs needed for that layer, and handle the conventions attached to the underlying interface and support the programmer in using them. This results in a hierarchy of languages which are implemented on top of each other.

5 Current Status

Currently, new languages can be defined and edited in APPLAB, as well as programs expressed in those languages. The structure-oriented editor has been augmented to also allow textual editing. This was done by adding *a Grammar Interpreting Parser* component. which also makes it possible to import programs in textual form from other systems. The support for static semantics is, at present, supplied by a *demand-attribute evaluator* for a subset of Door AG:s. The demand-attribute evaluator is not grammar interpretive and, thus, requires some preprocessing to work according to the current static semantic grammar. In the future it is desirable to have a grammar-interpretive semantic component that supports (full) Door AG:s and incremental attribute evaluation.

APPLAB has been integrated into the Robotics lab at the Department of Automatic Control, Lund University, and is used as a programming interface to an ABB Irb-6 robot. An extended version of ABB's robot programming language ARL[ABB94] has been implemented in APPLAB, and simple programs have been entered into APPLAB and executed on the robot. The demand-attribute evaluator has been used to generate C code from ARL which is dynamically linked into the motion control of the robot

APPLAB is also used in a master thesis project in cooperation with SAAB Military Aircrafts, Linköping. In this project, APPLAB has been integrated with the Verilog CASE environment which supports graphical development of real-time systems. APPLAB is used to view and edit the finite state machine specifications, thus integrating structure-based editing into a graphical environment.

6 Future Work

In the case study on robot programming we intend to look into the language design level in more detail, and try to identify the set of domain-specific properties that need to be included in an application language for robot programming. It would also be interesting to consider if there are any general techniques useful for identifying such properties and including them in the design of an application language.

Another interesting area of research is *modular grammars*. When defining a new language, in APPLAB, or for a compiler-compiler tool, the language description is written as one document. If the language designer wishes to reuse parts of an old description these have to be copied into the new language description. It is desirable to be able construct a language by combining a number of different basic language blocks, e.g. expressions, statements, declarations. Also, e.g. in robot programming, it is of interest to support the design of a new language *on top of* an old one. I.e. to have several layers of grammars define a language. Tools and techniques for supporting the construction and use of such modular grammars are interesting research topics. Some related work has been done in this direction, e.g. [GCN92], [Bos95].

7 References

[ABB94] ABB Robotics. *ARL Reference Manual*, 1st edition, February 1994.

[BH96] Elizabeth Bjarnason and Görel Hedin. A Grammar-Interpreting Parser in a Language Design Laboratory. In *Proceedings of the Poster Session of CC'96 (International Conference on Compiler Construction)*. P. Fritzson (ed.), pp 15-24, LiTH-IDA-R-96-12, Dept. of Computer Science, Linköping University, Sweden, April 1996.

[Bja95] Elizabeth Bjarnason. *APPLAB: User's Guide* (version 1.0). Technical Report LU-CS-IR:95-2, Department of Computer Science, Lund University, September 1995.

[Bos95] Jan Bosch. *Layered Object Model—Investigating Paradigm Extensibility*. PhD thesis, Department of Computer Science, Lund University, October 1995.

[FJS86] Peter H Feiler, Fahimeh Jalili, and Johann H Schlichter. An Interactive Prototyping Environment for Language Design. In *Proceedings of the Nineteenth Annual Hawaii International Conference on System Sciences*, volume II, pages 106–116. IEEE, 1986.

[GCN92] D. Garlan, L. Cai, and R. L. Nord. A Transformational Approach to Generating Application-Specific Environments. In H. Weber, editor, *Proceedings of the 5th ACM SIGSOFT Symposium on Software Development Environments*, pages 68 – 77, Tyson's Corner, Va., December 1992. ACM. SIGSOFT Software Engineering Notes, 17(5).

[Hed92] Görel Hedin. *Incremental Semantic Analysis*. PhD thesis, Department of Computer Science, Lund University, Sweden, March 1992.

[Hed96] G. Hedin. Enforcing programming conventions by attribute extension in an open compiler. In *Proceedings of the Nordic Workshop on Programming Environment Research (NWPER'96)*, Aalborg, Denmark, May 1996.

[JK93] Esa Järnvall and Kai Koskimies. *Computer-Aided Language Implementation with TaLE*. Technical Report A-1993-4, Department of Computer Science, University of Tampere, Finland, July 1993.

[JKP91] Esa Järnvall, Kai Koskimies, and Jukka Paakki. *The Design Of The Tampere Language Editor (TaLE)*. Technical Report A-1991-10, Department of Computer Science, University Of Tampere, December 1991.

[Kli91] P. Klint. A Meta-Environment For Generating Programming Environments, pages 105–124. In Bergstra and Feijs, editors. *Algebraic Methods II: Theory, Tools and Applications*. Springer-Verlag New York Inc, 1991.

[KLMM93] J. L. Knudsen, M. Löfgren, O. L. Madsen, and B. Magnusson, editors. *Object-Oriented Environments. The Mjølner Approach*. The Object-Oriented Series. Prentice Hall, 1993.

[MHM+90] B. Magnusson, G. Hedin, S. Minör, et al. An overview of the Mjölner/Orm environment. In J. Bezivin et al., editors, *Proceedings of the 2nd International Conference TOOLS (Technology of Object-Oriented Languages and Systems)*, pages 635–646, Paris, June 1990. Angkor.

[MH93] Sten Minör and Görel Hedin. *Grammar Interpretation in Orm*, In [KLMM93], chapter 20, pages 297–306. Prentice Hall, 1993.

[Min90] Sten Minör. *On Structure-Oriented Editing*. PhD thesis, Department Of Computer Science, Lund University, 1990.

[Nil92] Klas Nilsson. *Application Oriented Programming and Control of Industrial Robots*. Lic. thesis, Department of Automatic Control, Lund Institute of Technology, July 1992.