Prehistory of the ASF+SDF System (1980–1984)

Jan Heering[1] Paul Klint[1,2]

[1]CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
[2]Faculty of Mathematics and Computer Science, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
e-mail: jan@cwi.nl and paulk@cwi.nl

1 Monolingual beginning

Our work on programming environments started in 1980 with the design of a dedicated environment for the Summer programming language [1], an object-oriented language with class definitions. Rather than a dedicated Summer environment, the general concept of a *monolingual environment* emerged [2]. In such an environment, a single language is used in different modes. More specifically, we investigated the requirements an integrated command/programming/debugging language would have to satisfy. Since Summer had not been designed with this particular purpose in mind, it is not surprising that a monolingual environment for Summer would have involved a revision of the language. This may have been one of the reasons we never "instantiated" the monolingual concept for Summer, but there were other, more important, ones:

- At that time Leo Geurts, Lambert Meertens, and other members of the Afdeling Informatica were developing the B language system (later renamed to ABC), which had a monolingual character in the sense that the command and programming modes of the system were integrated. The development of a monolingual environment for a suitably revised version of Summer would have been a major effort without obvious additional benefits.
- We started to realize that a monolingual environment would be a closed world whose facilities could not be easily borrowed or reused by other languages. Since every application has its own language (however small), we decided it would be much more efficient to develop a generic multilingual environment. Its design was started in 1982.

2 A programming environment based on language definitions

The idea was to base the generic environment on *language definitions*. These would consist of a combined syntax/prettyprinting section and two additional sections for static and dynamic semantics. The generic environment would support the interactive development of language definitions and their compilation to language specific subenvironments. It would view language definitions as libraries of language constructs from which individual constructs could be borrowed or reused to facilitate the construction of new definitions. A language needing an **if**-statement, for instance, would probably be able to borrow a suitable one from another language for which a definition already existed in the system.

We had some experience with language definitions. Part of the semantics of Summer had been described in a formalism consisting of BNF-like rules with embedded variables to which semantic actions written in Summer itself were attached [3]. Furthermore, Gert Florijn and Geert Rolf had written PGEN, an LL(1) parser generator [4]. One of the things PGEN taught us was that molding grammars

Proceedings of ASF+SDF95. A workshop on Generating Tools from Algebraic Specifications. May 11 & 12, 1995, CWI, Amsterdam, M.G.J. van den Brand, A. van Deursen, T.B. Dinesh, J.F.Th. Kamperman & E. Visser (eds.) Technical Report P9504, Programming Research Group, University of Amsterdam

to fit the LL(1) restriction was no fun. This influenced our early decision to allow general context-free syntax in language definitions. Aloysius Tan designed a VLSI-algorithm to reduce the parsing time in the general context-free case to an acceptable value [5]. This was long before the syntax definition formalism SDF and lazy/incremental parser generation.

In the meantime, Henk Kroeze had experimented with a combined syntax/prettyprinting language for use in the first section of language definitions [6]. It turned out, however, that BNF rules with integrated prettyprint instructions were unreadable, and this remained a problem.

Although the generic environment we had in mind obviously needed a built-in semantics definition formalism (we did not yet know which one), it would be possible to use any language for which a definition had been constructed as a semantics definition formalism in the system. The corresponding towers of language interpreters would be very inefficient, so they would have to be flattened by the removal of intermediate layers. This we planned to do by *partial evaluation*.

This system concept was discussed with Wim Böhm, Marleen Sint, and Arthur Veen at several Data Flow Club meetings in 1982. It was subsequently presented at the Colloquium Programmeeromgevingen in the fall of that year [7, 8] and at the NGI-SION Symposium in Amsterdam in March 1983 (no proceedings).

3 Algebraic specification

The main decision facing us was what semantics definition method to use. The importance of partial evaluation in the system suggested a functional method without side-effects. Although denotational semantics would have been a natural choice, the closest we came to it was when we considered a statically scoped version of Lisp as a semantics definition formalism.

Among the papers on partial evaluation we studied were several by Valentin Turchin, which used the (string) rewrite rule language Refal, and we started discussing rewrite rules with Jan Bergstra. He taught us the relation between (term) rewrite rules and algebraic specifications. The fact that modularization was an important topic in the algebraic specification community was attractive to us in view of the modular construction of language definitions the generic environment had to support.

Although the algebraic semantics of programming languages was not a well developed subject, Jan Bergstra and Jan Willem Klop were working on *process algebra* (the algebraic semantics of processes) and we somehow suspected that algebraic specifications would be suitable for describing the static and dynamic semantics of languages in the generic environment. We never considered using different formalisms for static and dynamic semantics since we did not see a clear distinction between them. In this we were perhaps influenced by the monolingual concept discussed in Section 1. At a later stage, we started by not making a distinction between lexical and context-free syntax description in the syntax definition formalism SDF, but this proved untenable.

After a joint excursion into object-oriented algebraic specification [9], we set out to give an algebraic definition of the toy language PICO. Since we did not yet have a well-developed algebraic specification formalism, it was designed simultaneously. This became ASF. The syntax definition formalism SDF did not yet exist either, so the PICO definition included an algebraically specified syntax of PICO and a parser.

The proper modularization of the PICO definition turned out to be a major problem whose solution involved the repeated redesign of the module construction operators of ASF. The modularization finally adopted was very reasonable, but it

did not permit the reuse of individual PICO constructs in other language definitions. In this respect we did not achieve one of our original goals and this is still an open problem.

In the meantime, partial evaluation had not been forgotten. Although its algebraic semantics had not been studied in detail, it had been clear from the outset that algebraic specification and term rewriting were excellent frameworks for partial evaluation. As it turned out, partial evaluation involves the notion of ω-completeness of algebraic specifications. Somewhat ironically, the idea to allow any language for which a definition had been constructed as a semantics definition formalism in the system, which had been the main reason for studying partial evaluation, was gradually abandoned with the advent of algebraic specifications. Anyway, we finished both the PICO definition [10] and the partial evaluation paper [11] virtually at the time the GIPE project started in January 1985. At that time the implementation of ASF consisted of a parser, a type checker, and a Structure Diagram generator, all of them written in Summer using the PGEN parser generator mentioned before. Term rewriting had not yet been implemented.

4 Towards the ESPRIT/GIPE project

In July 1983 Paul Klint had visited INRIA Rocquencourt where he had familiarized himself with several generic environments [12]. One of them was the Mentor system which had been developed in the seventies by Véronique Donzeau-Gouge, Gérard Huet, Gilles Kahn, Bernard Lang, and others at INRIA [13]. In fact, Mentor was rather similar to what we had in mind for the syntactic part of the generic environment. Furthermore, its extension towards semantics had just begun with the development of the Typol language [14, 15], bringing INRIA's work even closer to ours.

Typol was based on Plotkin's Structural Operational Semantics, but it may be interesting to note that earlier experiments had been done with Formol, an Ada-like specification language specially designed for writing denotational semantics definitions of programming languages. Formol specifications were considered too low-level, however, and denotational semantics was abandoned.

Paul's visit did not immediately lead to further co-operation with INRIA, but in the spring of 1984 Gilles Kahn proposed to submit a joint ESPRIT proposal on the Generation of Interactive Programming Environments. For INRIA, it would be basically an extension of Mentor with semantics facilities. For us, it would be a continuation of our work on a generic environment based on algebraic language definitions. The ensuing proposal (part of which was later published [16]) was accepted by the European Communities and the GIPE project started in January 1985 with the software companies BSO (The Netherlands) and SEMA-METRA (France) as industrial partners. When it ended 5 years later, GIPE II took over for another 4 years [17].

References

[1] P. Klint, *From Spring to Summer*, Ph.D. Thesis, TH Eindhoven, 1982. Published as LNCS, Vol. 205, Springer-Verlag, 1985.

[2] J. Heering and P. Klint, Towards monolingual programming environments, Report IW 185/81, Mathematisch Centrum, Amsterdam, December 1981. Published in *ACM Transactions on Programming Languages and Systems*, **7** (1985), pp. 183–213.

[3] P. Klint, Formal language definitions can be made practical, Report IW 159/81, Mathematisch Centrum, Amsterdam, 1981. Published in J.W. de Bakker and J.C. van Vliet (Eds.), *Algorithmic Languages*, North-Holland, 1981, pp. 115–132, and in [1, Chapter 4].

[4] G. Florijn and G. Rolf, PGEN—A general purpose parser generator, Report IW 157/81, Mathematisch Centrum, Amsterdam, 1981.

[5] H.D.A. Tan, VLSI-algoritmen voor herkenning van context-vrije talen in lineaire tijd, Report IN 24/83, Mathematisch Centrum, Amsterdam, June 1983 (VLSI algorithms for the recognition of context-free languages in linear time—in Dutch). See also: A. Nijholt, Overview of parallel parsing strategies, in M. Tomita (Ed.), *Current Issues in Parsing Technology*, Kluwer Academic, 1991, Section 14.4.2.

[6] H. Kroeze, Een taalonafhankelijke benadering van prettyprinten, Report IN 21/82, Mathematisch Centrum, Amsterdam, December 1982 (A language independent approach to prettyprinting—in Dutch).

[7] J. Heering, Taaldefinities als kern voor een programmeeromgeving, in *Colloquium Programmeeromgevingen*, MC Syllabus 30, Mathematisch Centrum, Amsterdam, 1983, pp. 69–81 (A programming environment based on language definitions—in Dutch).

[8] P. Klint, Partiële evaluatie als implementatiemethode voor een programmeeromgeving, in *Colloquium Programmeeromgevingen*, MC Syllabus 30, Mathematisch Centrum, Amsterdam, 1983, pp. 83–100 (Partial evaluation as an implementation method for a programming environment—in Dutch).

[9] J.A. Bergstra, J. Heering, and J.W. Klop, Object-oriented algebraic specification: proposal for a notation and 12 examples, Report CS-R8411, CWI, Amsterdam, June 1984.

[10] J.A. Bergstra, J. Heering, and P. Klint, Algebraic definition of a simple programming language, Report CS-R8504, CWI, Amsterdam, February 1985. Published in J.A. Bergstra, J. Heering, and P. Klint (Eds.), *Algebraic Specification*, ACM Press Frontier Series, 1989, Chapter 2.

[11] J. Heering, Partial evaluation and ω-completeness of algebraic specifications, Report CS-R8501, CWI, Amsterdam, January 1985. Published in *Theoretical Computer Science*, **43** (1986), 149–167.

[12] P. Klint, A survey of three language-independent programming environments, Report IW 240/83, Mathematisch Centrum, Amsterdam, 1983.

[13] V. Donzeau-Gouge, G. Huet, G. Kahn, and B. Lang, Programming environments based on structured editors: the Mentor experience, INRIA Research Report No. 26, 1980. Published in D.R. Barstow, H.E. Shrobe, and E. Sandewall (Eds.), *Interactive Programming Environments*, McGraw-Hill, 1984, pp. 128–140.

[14] Th. Despeyroux and V. Donzeau-Gouge, Typol: Introduction de spécifications sémantiques dans Mentor, INRIA Research Report, 1983 (Typol: Introduction of semantics specifications in Mentor—in French).

[15] Th. Despeyroux, Executable specification of static semantics, INRIA Research Report No. 295, 1984. Published in G. Kahn, D.B. MacQueen, and G. Plotkin (Eds.), *Semantics of Data Types*, LNCS Vol. 173, Springer, 1984, pp. 215–233.

[16] J. Heering, G. Kahn, P. Klint, and B. Lang, Generation of interactive programming environments, in The Commission of the European Communities (Eds.), *ESPRIT '85: Status Report of Continuing Work*, Part I, Elsevier Science Publishers, 1986, pp. 467–477.

[17] J. Heering and P. Klint, Work done at CWI/UvA—Final report, in: *Sixth Review Report ESPRIT Project 2177 (GIPE II)*, January 1994.