INFORMATION PROCESSING 86, H.-J. Kugler (ed.)
Elsevier Science Publishers B.V. (North-Holland)

THE ROLE OF SYNTAX DIRECTED PROGRAMMING ENVIRONMENTS IN HANDLING SPECIFICATIONS

Bernard LANG
INRIA
B.P. 105, 78153 Le Chesnay, France

In this position statement, we outline the potential use of syntax directed environments for the specification phase of the software life-cycle. Despite differences in their roles and meanings, similar tools should be applicable to both specifications and programs at the syntactic level. The next step is to coordinate the development and maintenance of both within the same syntactic tools.

Software engineering technology has led to the development of a large number of tools to assist the programming activity. These tools are by force often developped by the programmers themselves, too often with their own very narrow view of the activities of software production. As a result, a large body of experience and tools is devoted exclusively to the coding stage of the software life-cycle.

Is some of that technology, namely the syntax directed manipulation of programs that has recently acquired wider popularity [1, 3, 4, 6, 8, 14, 18], applicable to other parts of the software life-cycle such as the specification stage?

There is an essential difference in intent, and thus in semantic nature, between a specification and a program. A specification should define the fonctionality of a system, i.e. in simple cases its input-output relations. It is by nature extensional. A program, on the other end, is an explicit and usually detailed description of an operational process that effectively realizes such input-output relations.

Formally, following an interpretation of the Howard correspondance [9], a specification may be seen as a predicate stating the existence of some mathematical object (the result) that satisfies a property depending on other objects (the data). A program satisfying the specification is a constructive proof of the satisfiability of the predicate, i.e. of the existence of the result.

In this ideal setting, the programming activity consists in the production of such a proof. As in mathematics, this is usually achieved by a complex process of analysis and synthesis, with many intermediate steps materialized by definitions (declarations) and lemmas (subprograms).

These formal distinctions are often blurred in practice, because for historical or pragmatic reasons the specification often includes a description of part of the computing process, and because the derivation of the program often goes through a hierarchy of specifications of lower level components.

But several facts however remain:

- due to a difference in role and contents, specification languages are distinct from programming languages.

- both types of languages must have a formal definition (and thus a formal syntax) if one is to reliably (mathematically ?) establish adequation between specifications and programs, or if one is to derive programs mechanically from the specification.

- program proofs and/or derivations are complex processes that must be mechanically assisted, if only to manage correctly the large amount of data involved.

- these processes involve simultaneous consideration and handling of both programs and specifications, i.e. simultaneous use of the corresponding languages. The mixing of these languages is frequently materialized by the inclusion of specification comments within the program code.

The above remarks remain true even when one uses operational specifications expressed in a higher level language than the one actually used to produce compilable code.

Syntax directed editing technology has now moved from systems dedicated to a unique language to language independent *meta-systems* driven by tables describing the languages to be manipulated [1, 6, 10, 16]. These tables are themselves produced by compilation of high-level descriptions of the pertinent features of the corresponding languages.

Such systems can therefore be applied to specification languages (and for that matter to other kinds of languages) as well as to programming ones, inasmuch as they can be formally described and at least on a purely syntactical basis. But this is only for the creation and maintenance of specifications (which are non negligeable tasks), independently of the corresponding program.

Let us note at this point that, as is well known, the evolution and improvement of software products

implies evolution and maintenance of their specification as well as of the software products themselves.

Two additional properties (partly implemented in some systems [5, 12]) are necessary to assist the coordinated development of specifications and programs:

- the ability to manipulate both at the same time, if use of their relationship is to be mechanically assisted;
- the ability to merge both harmoniously in the same documents and/or data base, without loosing their identities.

One should be able to handle documents combining precisely and accurately programs with their specifications and emphasizing their relationship, while keeping the formal structure of both of them (e.g. by means of structured comments precisely attached to structured representations of programs and/or specifications).

The maturity of current technology has hardly reached that point where such purely syntactic assistance is provided to users. A considerable ground is still to be covered (especially in semantics) before we can produce tools that assist real documents manipulations on the basis of their meaning [2, 7, 11, 15]. The substancial advances in recent research in mathematical logic, formal systems and symbolic computation still have to be clarified and tranferred to the industrial application level.

Graphical representations traditionally play an important role in specifications (e.g. [13, 17]). Thus the developement of high quality, graphics based, man-machine interfaces will be essential if syntax directed environments are to be used in this area. Work in this direction is already under way for programming uses of syntax directed environnements [14].

References

[1] R. Bahlke and G. Snelting, The PSG - Programming System Generator, *Proc. ACM SIGPLAN 85 Symp. on Language Issues in Programming Environments, SIGPLAN Notices,* vol. 20, no. 7, 1985, 28-33.

[2] T. Despeyroux, Executable Specification of Static Semantics, *Springer Lecture Notes in Computer Science,* Vol. 185, June 1984.

[3] V. Donzeau-Gouge, G. Kahn, G. Huet, B. Lang, and J.J. Levy, A structure oriented program editor:a first step toward computer assisted programming, *Proc. International Computing Symposium,* North_Holland, 1975.

[4] V. Donzeau-Gouge, G. Kahn, B. Lang, B. Mélèse, and E. Morcos, Outline of a tool for document manipulation, *Proc. of IFIP Congress '83,* Paris, R.E.A. Mason (ed.), North Holland, September 1983.

[5] V. Donzeau-Gouge, G. Kahn, B. Lang, and B. Mélèse, Document Structure and Modularity in Mentor, *Proc. ACM SIGSOFT/SIGPLAN Soft. Eng. Symp. on Practical Software Development Environments,* Pittsburgh, April 1984.

[6] Gandalf, *The Journal of Systems and Software,* Special issue on the Gandalf system, 5 (2), May 1985.

[7] J. Heering, G. Kahn, P. Klint and B. Lang, Generation of Interactive Programming Environments. Esprit Project N.348, *Proc. of Esprit Technical Week,* July 1984.

[8] Henderson, P.B., Edit., *Proc. ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments,* Pittsburgh, April 1984.

[9] W. Howard, The formulae-as-types notion of construction, in: *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism,* Hindley and Seldin ed., 1980, 479-490.

[10] G. Kahn, B. Lang, B. Mélèse, and E. Morcos, Metal: a formalism to specify formalisms, *Science of Computer Programming,* 3, North Holland, 1983, 151-188.

[11] G.E. Kaiser, Semantics for Structure Editing Environments, PhD Thesis, Tech. Report CMU-CS-85-131, Carnegie-Mellon University, Pittsburgh, PA, May 1985.

[12] B. Mélèse, Structured editing - Unstructured editing, Cooperation and complementarity, *Actes du 2e Colloque de Génie Logiciel,* Nice, June 1984.

[13] C.A. Petri, General Net Theory, *Proc. of the Joint IBM and Univ. of Newcastle upon Tyne Seminar,* September 1976.

[14] S.P. Reiss, Graphical Program Development with PECAN Program Development Systems, *Proc. ACM SIGSOFT/SIGPLAN Soft. Eng. Symp. on Practical Software Development Environments,* Pittsburgh, April 1984.

[15] T. Reps, Optimal-time Incremental Semantic Analisys for Syntax Directed Editors, *Proc. 9th Annual Symp. on Principles of Programming Languages,* January 1982.

[16] T. Reps, Generating Language Based Environments, PhD Thesis, Tech. Report 82-514, Cornell University, Ithaca, NY, August 1982.

[17] D.T. Ross, Structured Analysis (SA): A Language for Communicating Ideas, *IEEE Transactions on Software Engineering,* vol. SE-3, no. 1, January 1977.

[18] T. Teitelbaum and T. Reps, The Cornell Program Synthesizer: A Syntax-Directed Programming Environment, *Communications of the ACM,* vol. 24, no. 9, September 1981.