The ASF+SDF Meta-Environment: a Component-Based Language Development Environment

M.G.J. van den Brand[1], A. van Deursen[1], J. Heering[1], H.A. de Jong[1], M. de Jonge[1], T. Kuipers[1], P. Klint[1], L. Moonen[1], P.A. Olivier[1], Scheerder[2], J.J. Vinju[1], E. Visser[3], and J. Visser[1]

[1] *Centrum voor Wiskunde en Informatica (CWI), Kruislaan 413, 1098 SJ Amsterdam, The Netherlands*
[2] *Faculty of Philosophy, Utrecht University, Heidelberglaan 8, 3584 CS Utrecht, The Netherlands*
[3] *Faculty of Mathematics and Computer Science, Utrecht University, Padualaan 14, 2584 CH Utrecht, The Netherlands*

Abstract. The ASF+SDF Meta-Environment is an interactive development environment for the automatic generation of interactive systems for constructing language definitions and generating tools for them. Over the years, this system has been used in a variety of academic and commercial projects ranging from formal program manipulation to conversion of COBOL systems. Since the existing implementation of the Meta-Environment started exhibiting more and more characteristics of a legacy system, we decided to build a completely new, component-based, version. We demonstrate this new system and stress its open architecture.

1 Introduction

The ASF+SDF Meta-Environment [12] is an interactive development environment for the automatic generation of interactive systems for constructing language definitions and generating tools for them. A language definition typically includes such features as syntax, prettyprinting, typechecking, and execution of programs in the target language. The ASF+SDF Meta-Environment can help in the following cases:

– You have to write a formal specification for some problem and you need interactive support for this.
– You are developing your own (application) language and want to create an interactive environment for it.
– You have programs in some existing programming language and you want to analyze or transform them.

The ASF+SDF formalism [1] [10] allows the definition of syntactic as well as semantic aspects. It can be used for the definition of languages (for programming, writing specifications, querying databases, text processing, or other applications). In addition it can be used for the formal specification of a wide variety of problems. ASF+SDF provides:

– A general-purpose algebraic specification formalism based on (conditional) term rewriting.
– Modular structuring of specifications.
– Integrated definition of lexical, context-free, and abstract syntax.
– User-defined syntax, allowing you to write specifications using your own notation.
– Complete integration of the definition of syntax, and semantics.
– Traversal functions (for writing very concise program transformations), memo functions (for caching repeated computations), and more.

The ASF+SDF Meta-Environment offers:

– Syntax-directed editing of ASF+SDF specifications.
– Incremental compilation and testing of specifications.
– Compilation of ASF+SDF specifications into dedicated interactive stand-alone environments containing various tools such as a parser, prettyprinter, syntax-directed editor, debugger, and interpreter or compiler.
– User-defined extensions of the default user-interface.

The design goals of the new implementation to be demonstrated include: openness, reuse, extensibility, and in particular the possibility to generate complete stand-alone environments for user-defined languages.

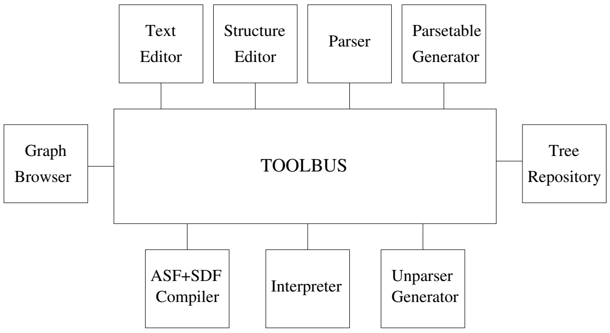

Fig. 1. Architecture of the ASF+SDF Meta-Environment

2 Technological background

ToolBus A hallmark of legacy systems in general and the old ASF+SDF Meta-Environment in particular is the entangling of control flow and actual computation. To separate coordination from computation we use the ToolBus coordination architecture [2], a programmable software bus based on process algebra. Coordination is expressed by a formal description of the cooperation protocol between components while computation is expressed in components that may be written in any language. We thus obtain interoperability of heterogeneous components in a (possibly) distributed system.

ATerms Coordination protocol and components have to share data. We use ATerms [4] for this purpose. These are trees with optional annotations on each node. The annotations are used to store tool-specific information like text coordinates or color attributes. The implementation of ATerms has two essential properties: terms are stored using maximal subterm sharing (reducing memory requirements and making deep equality tests very efficient) and they can be exchanged using a very dense binary encoding that preserves sharing. As a result very large terms (with over $1,000,000$ nodes) can be processed.

SGLR In our language-centric approach the parser is an essential tool. We use scannerless, generalized-LR parsing [13]. In this way we can parse arbitrary context-free grammars, an essential property when combining and parsing large grammars for (dialects of) real-world languages.

Term rewriting ASF+SDF specifications are executed as (conditional) rewrite rules. Both interpretation and compilation (using the ASF2C compiler [5]) of these rewrite rules are supported. The compiler generates very efficient C code that implements pattern matching and term traversal. The generated code uses ATerms as its main data representation, and ensures a minimal use of memory during normalization of terms.

3 Architecture

The architecture of the ASF+SDF Meta-Environment is shown in Figure 1. It consists of a ToolBus that interconnects the following components:

- **User interface**: the top level user-interface of the system. It consists primarily of a graph browser for the import graph of the current specification.
- **Text Editor**: a customized version of `XEmacs` for text editing.
- **Structure Editor**: a syntax-directed editor that closely cooperates with the Text Editor.
- **Parser**: scannerless, generalized-LR parser (SGLR) that is parametrized with a parse table.
- **Parsetable generator**: takes an SDF syntax definition as input and generates a parse table for SGLR.
- **Tree Repository**: stores all terms corresponding to specification modules, parse tables, user-defined terms, etc.
- **Compiler**: the ASF2C compiler.
- **Interpreter**: executes specifiations by direct interpretation.
- **Unparser generator**: generates prettyprinters.

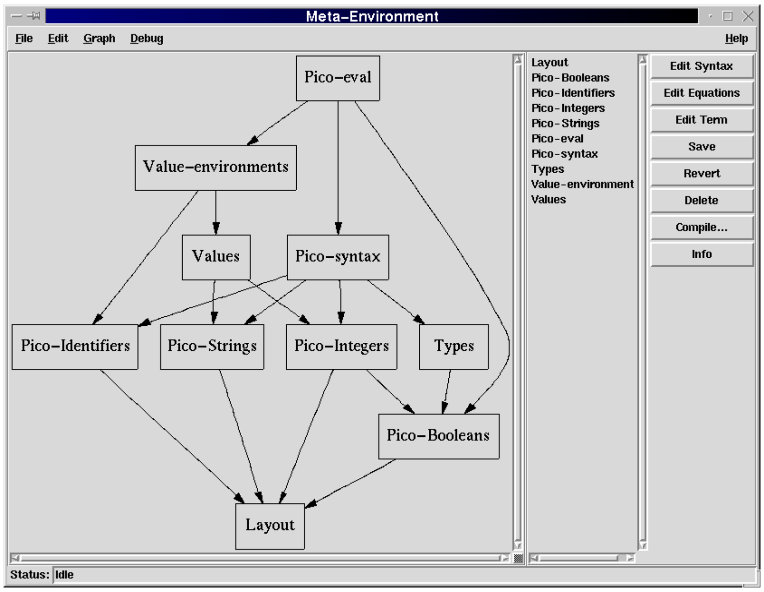

Fig. 2. The main user-interface of the Meta-Environment is a module browser that provides a graphical and a textual view of the modules in a specification. A number of operations can be initiated for each module. Here it is shown with the modules from a small specification of a typechecker for the toy language Pico.

4 Applications of ASF+SDF and the Meta-Environment

There are a number of academic and industrial projects that use either ASF+SDF directly or components of the Meta-Environment in one way or another. c The applications of ASF+SDF can be split into three groups:

1. In the field of language prototyping ASF+SDF has been used to describe the syntax and semantics of *domain specific languages*, e.g., the language Risla for describing financial products [3]. As another example, the syntax of the algebraic specification language CASL has been prototyped using ASF+SDF [7]. BOX [9] [11] is a small domain specific language developed for prettyprinting within the Meta-Environment.
2. In the field of *reverse engineering and system renovation*, ASF+SDF is used to analyze and transform COBOL legacy code [8].
3. As an algebraic specification formalism for specifying *language processing tools*. In fact, a number of components of the Meta-Environment itself have been specified using ASF+SDF:
   - the ASF2C compiler,
   - the unparser generator, and
   - parts of the parsetable generator.

   For other components, such as the ToolBus and the syntax-directed editor, an ASF+SDF specification was made for prototyping use only. That specification formed the basis for an optimized, handcrafted implementation.

Components of the Meta-Environment are used as stand-alone tools in a variety of applications. Examples are the Stratego compiler [14], the Risla compiler, the Elan environment [6], and a commercial tool for designing and implementing information systems.

5 Demonstration

We will show a number of applications of the Meta-Environment ranging from a simple typechecking problem (Figure 2) to syntax-directed editing and transformation of COBOL systems.

6 Obtaining the ASF+SDF Meta-Environment

The ASF+SDF Meta-Environment can be downloaded from:

`http://www.cwi.nl/projects/MetaEnv/completa/`

Individual components, such as the ATerm library, ToolBus, parser generator, and parser (SGLR) can be obtained from: `http://www.cwi.nl/projects/MetaEnv/`.

All components of the ASF+SDF Meta-Environment are available as open source software.

References

1. J.A. Bergstra, J. Heering, and P. Klint, editors. *Algebraic Specification*. ACM Press/Addison-Wesley, 1989.
2. J.A. Bergstra and P. Klint. The discrete time ToolBus – a software coordination architecture. *Science of Computer Programming*, 31(2-3):205–229, July 1998.
3. M.G.J. van den Brand, A. van Deursen, P. Klint, S. Klusener, and E.A, van den Meulen. Industrial applications of ASF+SDF. In M. Wirsing and M. Nivat, editors, *Algebraic Methodology and Software Technology (AMAST '96)*, volume 1101 of *LNCS*. Springer-Verlag, 1996.
4. M.G.J. van den Brand, H.A. de Jong, P. Klint, and P. Olivier. Efficient Annotated Terms. *Software, Practice & Experience*, 30:259–291, 2000.
5. M.G.J. van den Brand, P. Klint, and P. A. Olivier. Compilation and memory management for ASF+SDF. In S. Jähnichen, editor, *Compiler Construction (CC '99)*, volume 1575 of *Lecture Notes in Computer Science*, pages 198–213. Springer-Verlag, 1999.
6. M.G.J. van den Brand and C. Ringeissen. ASF+SDF parsing tools applied to ELAN. In *Third International Workshop on Rewriting Logic and Applications*, ENTCS, 2000.
7. M.G.J. van den Brand and J. Scheerder. Development of Parsing Tools for CASL using Generic Language Technology. In D. Bert, C. Choppy, and P. Mosses, editors, *Workshop on Algebraic Development Techniques (WADT'99)*, volume 1827 of *LNCS*. Springer-Verlag, 2000.
8. M.G.J. van den Brand, M.P.A. Sellink, and C. Verhoef. Generation of components for software renovation factories from context-free grammars. *Science of Computer Programming*, 36:209–266, 2000.
9. M.G.J. van den Brand and E. Visser. Generation of formatters for context-free languages. *ACM Transactions on Software Engineering and Methodology*, 5:1–41, 1996.
10. A. van Deursen, J. Heering, and P. Klint, editors. *Language Prototyping: An Algebraic Specification Approach*, volume 5 of *AMAST Series in Computing*. World Scientific, 1996.
11. M. de Jonge. A pretty-printer for every occasion. In I. Ferguson, J. Gray, and L. Scott, editors, *Proceedings of the 2nd International Symposium on Constructing Software Engineering Tools (CoSET2000)*. University of Wollongong, Australia, 2000.
12. P. Klint. A meta-environment for generating programming environments. *ACM Transactions on Software Engineering and Methodology*, 2:176–201, 1993.
13. E. Visser. *Syntax Definition for Language Prototyping*. PhD thesis, University of Amsterdam, 1997.
14. E. Visser, Z. Benaissa, and A. Tolmach. Building Program Optimizers with Rewriting Strategies. In *International Conference on Functional Programming (ICFP'98)*, pages 13–26, 1998.