Available online at www.sciencedirect.com

SCIENCE DIRECT®

The Journal of Logic and Algebraic Programming 59 (2004) 1–4

THE JOURNAL OF LOGIC AND ALGEBRAIC PROGRAMMING

www.elsevier.com/locate/jlap

Guest editor's introduction: Special issue on Annotated Terms (ATerms)

M.G.J. van den Brand[a,b]

[a] *Department of Software Engineering, CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands*
[b] *Hogeschool van Amsterdam, Institute for Information Technology and Electrical Engineering, Weesperzijde 190, 1097 DZ Amsterdam, The Netherlands*

ATerms (short for Annotated Terms) are an abstract data type designed for the exchange of tree-like data structures between distributed applications. ATerms and the ATerm library were designed and implemented in the beginning of 1998, independent of the Extensible Markup Language[1] and OMGs Interface Definition Language (IDL). The reason for developing ATerms was the need for having an open, simple, efficient, concise, and language independent solution for the exchange of tree-like data structures between distributed applications. The characteristics of the ATerm library are maximal subterm sharing and automatic garbage collection. These characteristics prove to be very beneficial when developing, for instance, rewrite engines, and model checkers.

This special issue of the Journal of Logic and Algebraic Programming is devoted to ATerms. Preparing this special issue raises several questions. First of all, what makes ATerms such a useful exchange format? Second, why were ATerms invented, designed and implemented in the first place? Finally, what is the relation between logic and algebraic programming and ATerms? Before answering these questions I want to introduce the three papers in this special issue. The selected papers cover a broad range of topics related to ATerms. The paper by Pierre-Etienne Moreau and Olivier Zendra addresses the very technical issue of garbage collection inside the C implementation of the ATerm library. The second paper by Hayco de Jong and Pieter Olivier describes an extension of the ATerm library to improve programming against the ATerm library. The third paper by Anamaria Martins Moriera, Christophe Ringeissen, David Déharbe and Gleydson Lima discusses the FERUS tool set in which ATerms are used to connect with external tools, such as parsers, rewrite engines, etc.

The paper by Pierre-Etienne Moreau and Olivier Zendra presents an improvement of the garbage collector of the C implementation of the ATerm library. Garbage collection is crucial to the C version of the ATerm library because it frees the developer of the annoying administration task of allocating and freeing terms. The original garbage collector used in the ATerm library was a non-generational mark-and-sweep garbage collector based on the work described in [6]. Moreau and Zendra present in their paper a generational garbage collector which exploits the functional characteristics of the ATerm library. An ATerm node

E-mail address: mark.van.den.brand@cwi.nl (M.G.J. van den Brand).

[1] http://www.w3c.org/XML/hist2002.

doi:10.1016/j.jlap.2003.12.004

can never have a pointer to a node which is younger. A second improvement presented in this paper is the possibility to return allocated blocks to the operating system. This is achieved by shrinking the block sizes in combination with the generational garbage collector approach.

The paper by Hayco de Jong and Pieter Olivier[2] describes ApiGen: an "extension" to improve programming against the ATerm library. The motivation for their work was the observation that programming against the ATerm library could be considered harmful. The combination of the fact that every thing is an ATerm and the simple "match-and-make" paradigm resulted in code that is hard to maintain. ApiGen is a methodology, technique, and implementation vehicle to address this shortcoming of the ATerm library. Based on an abstract data type definition a collection of functions is generated which allows the manipulation of ATerms in a type-safe way. They show by means of figures derived from the implementation of the ASF+SDF Meta-Environment [1], the reduction in code size and indirectly the improvement in maintainability.

The paper by Anamaria Martins Moreira, Christophe Ringeissen, David Déharbe and Gleydson Lima presents the application of the ATerm library in the FERUS tool. The FERUS tool enables software component reuse by means of evolution of formal specification components. FERUS manipulates algebraic specification written in CASL [11]. ATerms are used as the internal data structure. The paper discusses the architecture of the FERUS tool set as well as the use of an alternative intermediate format, a graph format. This format is more concise when dealing with the transformations on the CASL specifications. ATerms are mainly used to communicate with other CASL related tools, such as the CATS tool set [12]. A comparison between the two formats is presented.

Before addressing the three questions raised in the beginning of this introduction I would like to thank a number of people, first of all Jan Bergstra, the chief editor of this journal, Inge Bethke, who provided the administrative support and kept me going, the referees who took the effort of reading and criticising the submitted papers and finally the authors who had the courage to submit papers. Without them this special issue would never have been created.

0.1. What makes ATerms such a useful exchange format?

Annotated Terms (ATerms) are a data type to represent tree structured data. The philosophy and implementation of ATerms is described in the original paper published in Software, Practice and Experience [3]. ATerms and the ATerm-library can be characterised as an *open*, *simple*, *efficient*, *concise*, and *language independent* representation for exchanging tree-like data structures between heterogeneous distributed applications. Furthermore the trees may be decorated with *annotations* which are transparent to the various applications processing ATerms.

On the implementation level two important characteristics make the ATerm-library unique: *maximal subterm sharing* and *fully automatic garbage collection*. Furthermore, ATerms and the ATerm-library provide a very simple *match-and-make* paradigm, that enables programmers to compose and decompose ATerms in a simple and efficient way.

[2] They actually designed and implemented the ATerm library, both the C and JAVA version.

0.2. Why were ATerms invented, designed, and implemented?

ATerms are the intermediate exchange format in the ASF+SDF Meta-Environment. The first generation of the ASF+SDF Meta-Environment [10] was completely implemented in LeLisp [7] and could be characterised as a monolithic system. The next generation of the Meta-Environment has a component-based architecture. The various components of the Meta-Environment: the ToolBus [4] (a software coordination architecture), the compiler, the parser, etc., all used their own format. This led to a lot of redundant conversions between these various formats. Looking in more detail at the various formats revealed that they had a strong syntactic resemblance and that all were used to represent syntax trees, either parse trees or abstract syntax trees. This observation lead to the creation of ATerms and the ATerm library. Of course, this also involved a conversion of all components of the Meta-Environment to this new format.

0.3. What is the relation between ATerms and logic and algebraic programming?

ATerms are a solution for a technical problem, how to exchange information between components in a component-based architecture such as the ASF+SDF Meta-Environment. The Meta-Environment can be characterised as an interactive programming environment for the algebraic specification formalism ASF+SDF. ATerms are very popular in systems which are similar to the Meta-Environment. In this special issue the FERUS tool set will be discussed. The tool set for CASL uses ATerms as the exchange format [12]. The programming environment, XT [8], of the strategy-based specification language Stratego [13], reuses a number of components of the Meta-Environment and thus uses ATerms as its exchange format. The new programming environment for the algebraic specification language ELAN [5] reuses the same architecture as the Meta-Environment and also uses ATerms as its exchange format. The μCRL tool set [2], a collection of tools for the manipulation of process and data descriptions in μCRL (micro Common Representation Language) [9] is also based on the ATerm-library.

References

[1] M.G.J. van den Brand, A. van Deursen, J. Heering, H.A. de Jong, M. de Jonge, T. Kuipers, P. Klint, P.A. Olivier, J. Scheerder, J.J. Vinju, E. Visser, J. Visser, The ASF+SDF Meta-Environment: a component-based language development environment, in: R. Wilhelm (Ed.), Compiler Construction (CC'01), Lecture Notes in Computer Science, vol. 2027, Springer-Verlag, 2001, pp. 365–370.

[2] S.C.C. Blom, W.J. Fokkink, J.F. Groote, I. van Langevelde, B. Lisser, J.C. van de Pol, μCRL: A toolset for analysing algebraic specifications, in: G. Berry, H. Comon, A. Finkel (Eds.), Proceedings of the CAV 2001, LNCS, vol. 2102, Springer-Verlag, 2001, pp. 250–254.

[3] M.G.J. van den Brand, H.A. de Jong, P. Klint, P. Olivier, Efficient Annotated Terms, Software, Practice and Experience 30 (2000) 259–291.

[4] J.A. Bergstra, P. Klint, The discrete time ToolBus—a software coordination architecture, Science of Computer Programming 31 (2–3) (1998) 205–229.

[5] P. Borovansky, C. Kirchner, H. Kirchner, P.-E. Moreau, ELAN from a rewriting logic point of view, Theoretical Computer Science (285) (2002) 155–185.

[6] H. Boehm, M. Weiser, Garbage collection in an uncooperative environment, Software, Practice and Experience (SPE) 18 (9) (1988) 807–820.

[7] J. Chailloux, M. Devin, J.M. Hullot, Lelisp a portable and efficient lisp system, in: Conference Record of the 1984 ACM Symposium on LISP and Functional Programming, ACM, 1984, pp. 113–123.

[8] M. de Jonge, E. Visser, J. Visser, Xt: A bundle of program transformation tools, in: M. van den Brand, D. Parigot (Eds.), Electronic Notes in Theoretical Computer Science, vol. 44, Elsevier, 2001.
[9] J.F. Groote, A. Ponse, The syntax and semantics of μCRL, in: Algebra of Communicating Processes '94, Workshops in Computing, Springer-Verlag, 1995, pp. 26–62.
[10] P. Klint, A meta-environment for generating programming environments, ACM Transactions on Software Engineering and Methodology 2 (1993) 176–201.
[11] P.D. Mosses, CASL: A guided tour of its design, in: Recent Trends in Algebraic Development Techniques, 13th International Workshop, WADT'98, Lisbon, Portugal, 1998, Selected Papers, vol. 1589, 1999, pp. 216–240.
[12] T. Mossakowski, CASL: From semantics to tools, in: S. Graf, M. Schwartzbach (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, 6th International Conference, TACAS 2000, Berlin, Germany, Proceedings, Lecture Notes in Computer Science, vol. 1785, Springer-Verlag, 2000, pp. 93–108.
[13] E. Visser, Stratego: A language for program transformation based on rewriting strategies, System description of Stratego 0.5, in: A. Middeldorp (Ed.), Rewriting Techniques and Applications (RTA'Ol), Lecture Notes in Computer Science, vol. 2051, Springer-Verlag, 2001, pp. 357–361.