Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 141 (2005) 161–165
www.elsevier.com/locate/entcs

# TIDE: A Generic Debugging Framework — Tool Demonstration —

M.G.J. van den Brand [1]

*Department of Software Engineering*
*Centrum voor Wiskunde en Informatica*
*Kruislaan 413, NL-1098 SJ Amsterdam, The Netherlands*
*and*
*Institute for Information Technology*
*Hogeschool van Amsterdam*
*Weesperzijde 190, NL-1097 DZ Amsterdam, The Netherlands*

B. Cornelissen[2] P.A. Olivier[4] J.J. Vinju[3]

*Department of Software Engineering*
*Centrum voor Wiskunde en Informatica*
*Kruislaan 413, NL-1098 SJ Amsterdam, The Netherlands*

**Abstract**

A language specific interactive debugger is one of the tools that we expect in any mature programming environment. We present applications of TIDE: a generic debugging framework that is related to the ASF+SDF Meta-Environment. TIDE can be applied to different levels of debugging that occur in language design.
Firstly, TIDE was used to obtain a full-fledged debugger for language specifications based on term rewriting. Secondly, TIDE can be instantiated for any other programming language, including but not limited to domain specific languages that are defined and implemented using ASF+SDF.
We demonstrate the common debugging interface, and indicate the amount of effort needed to instantiate new debuggers based on TIDE.

*Keywords:* Generic debugging, rewriting, language specifications

[1] Email: Mark.van.den.Brand@cwi.nl
[2] Email: sgmcorne@science.uva.nl
[3] Email: Jurgen.Vinju@cwi.nl
[4] Email: pieter@gamesquare.nl


doi:10.1016/j.entcs.2005.02.056

# 1 Introduction

The development of mature programming environments for small languages, such as domain specific languages, is in general not feasible due to the development and maintenance overhead. The construction of a parser, an interpreter or a compiler already involves a tremendous amount of effort. The ASF+SDF Meta-Environment [1] supports the prototyping of (domain specific) languages on both the syntactic and semantic level. Given the developed specification, tools such as parsers, interpreters and pretty printers are derivable.

In addition to these tools, a generic debugging framework called TIDE [4] complements the ASF+SDF Meta-Environment with the possibility for easily obtaining interactive debuggers as well. Note that TIDE is independent of the ASF+SDF Meta-Environment, and can be applied to debugging of any other (formal) language.

In this demonstration we introduce the concepts of this generic debugging framework, and demonstrate its usage on two levels of debugging:

- Debugging a specification of a domain specific language.
- Debugging programs written in this domain specific language.

## *1.1 ASF+SDF Meta-Environment*

The ASF+SDF Meta-Environment [1] is a language definition and program manipulation environment. It can be used interactively to define languages and to generate tools from these definitions. A language specification in ASF+SDF [3] typically includes a definition of the syntax, pretty-printing, type-checking, and execution of programs of the target language.

For this demonstration we will concentrate on the ASF formalism. ASF is a declarative formalism based on conditional rewrite rules. It is a first-order strongly typed language which provides functionality for list matching and a tree traversal mechanism. Furthermore, ASF is strongly connected with SDF in the sense that ASF reuses the syntax defined in the SDF part to obtain user defined (concrete) syntax.

## *1.2 TIDE*

A language specific interactive debugger is useful when developing programs for a specific language, but creating it usually involves a lot of work. TIDE [4] remedies this fact by providing a generic debugging framework based on a language independent and extensible debugging interface.

For instance, when an interpreter communicates with TIDE's interface during execution of a program, TIDE will present a full-fledged interactive

debugger to the user. Besides facilities for stepping through the program at hand TIDE also provides facilities to set breakpoints, and inspect variables and stack frames. Depending on the specific language that is debugged, more or less features of TIDE are applicable.

### *1.3 Implementing a debugger for a new language*

To extend a run-time system of a language, there are few requirements. TIDE uses a simple communication protocol, which is implemented by libraries that are available in C and Java. Using these libraries, an expert should build the interface between TIDE and the run-time system of a language. This interface we refer to as a *TIDE adapter*. The following is an abstract overview of his agenda:

- Start TIDE in a separate process, and initialize the connection to TIDE.
- Identify the *logical breakpoints*, the steps, at which execution may be interrupted during debugging. They identify all possible breakpoints during execution that are logical from the programmer's perspective. In conventional debuggers logical breakpoints are often defined just before execution of a line of code, a single statement, or evaluation of a subexpression. A call to TIDE's interface is added at each of these points.
- At these breakpoints at least the *source location* of the current point of execution must be known, and passed to the TIDE library. For more features, the nesting depth (i.e. the stack depth) and a serialized representation of a variable environment can also be passed to TIDE.
- Finally, a number of *debugging rules* should be defined, that state which debugging actions are available at which logical breakpoints. General rules are registered with TIDE at initialization time, which may be edited by either the user or the run-time system at any time during debugging.

After completing the above tasks, a basic debugging interface that offers source inspection, breakpoints, watchpoints and variable inspection facilities is ready for use. More specialized behavior, such as language specific visualizations, require extending TIDE's implementation in Java. A number of interfaces and abstract classes are available to service the integration of new components into TIDE.

The TIDE protocol ensures that debugging is not limited to single threaded applications. Even heterogeneous, parallel and distributed applications can communicate with a single TIDE server, allowing the user to simultaneously debug communicating processes. The only requirement is that all source code must be locally available.

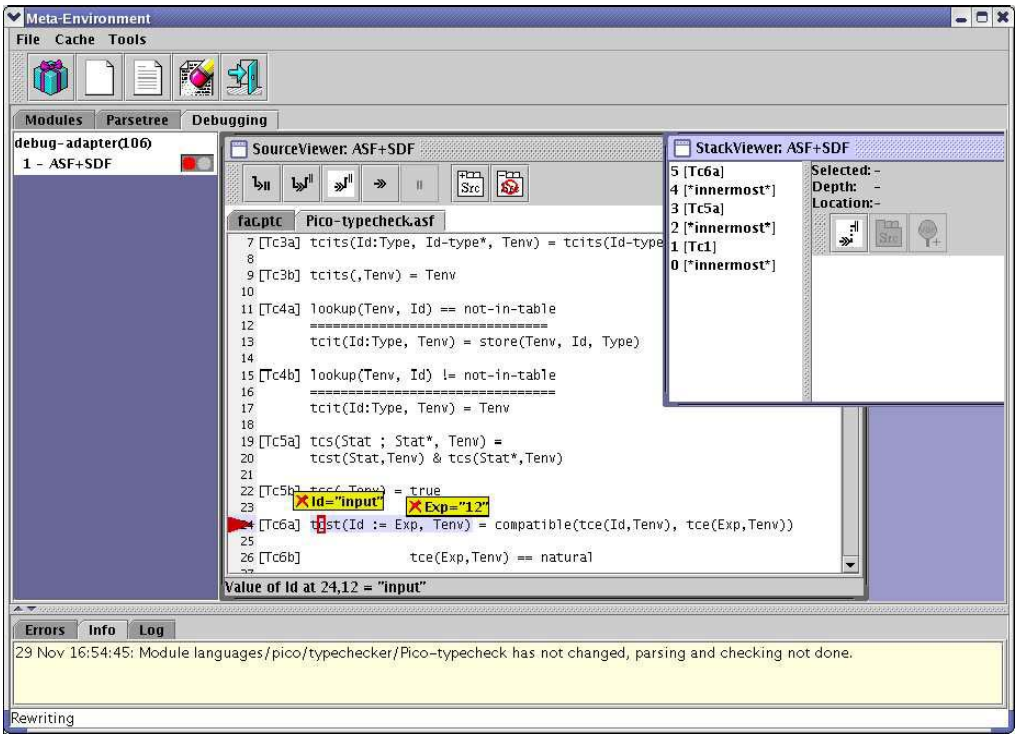


Fig. 1. The main user-interface of the ASF+SDF Meta-Environment running TIDE.

## 2 Demonstration of TIDE debuggers

### *2.1 Debugging a language specification*

ASF is used to define the semantics of (programming) languages by means of conditional equations. These equations can either be directly interpreted or compiled to efficient C code [2]. In debugging mode, the ASF interpreter communicates with TIDE using the aforementioned adapter API.

Figure 1 shows a screen dump of the user-interface of the ASF+SDF Meta-Environment with an integrated TIDE widget. The specification being debugged is a type-checker for the toy programming language Pico. The user added a breakpoint on line `24` in the module `Pico-typecheck.asf` and inspected both the variables `Id` and `Exp`.

The ASF interpreter defines 21 logical breakpoints during interpretation of a specification. For example, two breakpoints logically identify rewrite rule application: one before matching the left-hand side, and another just before constructing the right-hand side. The marshalling of the ASF value environment to TIDE is implemented in 350 lines of C code. That is enough to obtain a full-fledged ASF debugger.

### 2.2 Debugging programs written in domain specific languages

When a language is already defined in ASF+SDF, it should not be hard to obtain a TIDE-based debugger for this language. We provide a generic ASF+SDF module that encapsulates and hides the TIDE adapter API. Now we can instrument an ASF+SDF specification by adding calls to this module in any language specification. For example, the language specification of Pico was extended with calls to the `tide-step` function, obtaining a debugger that supports stepping through a Pico program and setting breakpoints.

Naturally, this route does not offer the full expressivity that can be obtained by writing a TIDE adapter manually. Indeed, the interface offered is abstract but functional. The design of the ASF+SDF debugging module is a trade-off between automation (less coding for the user) and expressivity (more coding for the user).

As an aside, we would like to point to the work of Wu et al. [5] which is highly related. It offers debuggers for domain specific languages using the Eclipse framework.

## 3 Conclusion

TIDE provides a flexible interactive debugging framework which allows a rapid development of a debugger for any language. We showed a debugger for the ASF formalism, and how ASF+SDF can be used to automatically obtain a TIDE-based debugger from a language specification.

Future work includes the development of more TIDE adapters: e.g. for grammars and the parsers generated from them, for some general purpose languages (C and Java), and for several domain specific languages.

## References

[1] M.G.J. van den Brand, A. van Deursen, J. Heering, H.A. de Jong, M. de Jonge, T. Kuipers, P. Klint, L. Moonen, P. A. Olivier, J. Scheerder, J.J. Vinju, E. Visser, and J. Visser. The ASF+SDF Meta-Environment: a Component-Based Language Development Environment. In R. Wilhelm, editor, *CC'01*, volume 2027 of *LNCS*, pages 365–370. Springer-Verlag, 2001.

[2] M.G.J. van den Brand, J. Heering, P. Klint, and P.A. Olivier. Compiling language definitions: The asf+sdf compiler. *ACM Transactions on Programming Languages and Systems*, 24(4):334–368, 2002.

[3] 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.

[4] P. A. Olivier. *A Framework for Debugging Heterogeneous Applications*. PhD thesis, Universiteit van Amsterdam, 2000.

[5] H. Wu J. Gray S. Roychoudhury and M. Mernik. Weaving a debugging aspect into domain-specific language grammars. In *ACM Symposium for Applied Computing (SAC)*, March 2005.