An Eclipse-based Integrated Environment for Developing Executable Structural Operational Semantics Specifications

Adrian Pop[1,2] Peter Fritzson[3]

Programming Environments Laboratory
Department of Computer and Information Science
Linköping University
Linköping, Sweden

Abstract

The Structural Operational Semantics Development Tooling (SOSDT) Eclipse Plugin integrates the Relational Meta-Language (RML) compiler and debugger with the Eclipse Integrated Development Environment Framework. SOSDT, together with the RML compiler and debugger, provides an environment for developing and maintaining executable Structural Operational Semantics specifications, including the Natural Semantics big step variant of SOS specifications. The RML language is successfully used at our department for writing large specifications for a range of languages like Java, Modelica, Pascal, MiniML etc. The SOSDT environment includes support for browsing, code completion through menus or popups, code checking, automatic indentation, and debugging of specifications.

Keywords: SOS, Natural Semantics, executable specification, Eclipse, RML, debugging.

1 Introduction

No programming language environment can be considered mature if is not supported by a strong set of tools which include execution, debugging, and profiling.

In this paper we present an integrated development environment called Structural Operational Semantics Development Tooling (SOSDT) [4] for browsing, checking, and debugging semantic specifications. The SOSDT environment is based on the existing Relational Meta-Language (RML) system and its debugger and provides an easy to use graphical interface for these systems.

[1] This research was partially supported by the National Graduate School in Computer Science (CUGS) and the SSF RISE project.

[2] Email: adrpo@ida.liu.se

[3] Email: petfr@ida.liu.se

This paper is electronically published in
Electronic Notes in Theoretical Computer Science

2 SOS/Natural Semantics and the Relational Meta-Language (RML)

Natural Semantics [2] is formalism for specifying many aspects of programming languages, e.g. type systems, dynamic semantics, translational semantics, static semantics, etc. Natural Semantics is an operational semantics derived from the Plotkin [6] structural operational semantics combined with the sequent calculus for natural deduction.

The Relational Meta-Language (RML) [5], is a practical language for writing executable SOS/Natural Semantics Specifications. The RML language is extensively used at out department for teaching and writing large specifications for different languages like Java, Modelica, MiniML, Pascal, etc. The RML language is compiled to highly efficient C code by the rml2c compiler. In this way, large parts of a compiler can be automatically generated from their Natural Semantics specifications. From the features of the RML language we can mention: strong static typing, simple module system, type inference, pattern matching and recursion are used for control flow, types can be polymorphic.

As pointed out in [3], the computer science community is constantly ignoring the debugging problem even though the debugging phase of software development takes more than the overall development time. Even if the RML language has a very short learning curve, the absence of debugging facilities previously created problems of understanding, debugging and verification of large specifications. We have addressed the debugging issue by providing a debugging framework for RML [7]. The debugger is based on abstract syntax tree instrumentation (program transformation) in the RML compiler and some runtime support. Type reconstruction is performed at runtime in order to present values of the user defined types.

3 The RML Integrated Environment (SOSDT) as an Eclipse Plugin

The SOSDT (previously named RML Development Tooling (RDT)) environment provides an integrated environment for our tools. The integrated environment with debugging and the various interactions between the components is presented in Figure 1 and 2.

The SOSDT environment has three major components, the RML Editor, the RML Browser and the RML Debugging components. All the components are active when the SOSDT perspective is selected within the Eclipse environment. Perspectives within Eclipse are used for configuration of views in connection with specific projects. Within the SOSDT environment the user creates and manages RML projects and RML files via wizards.

The RML Editor component provides syntax highlighting, auto indentation, code completion, type information and error highlighting. This component obtains the needed information from the RML parser and the RML Compiler. From the RML Compiler the errors and the type inference information is gathered. The type information is displayed when hovering over a variable, relation or pattern. Code completion is provided when the user writes relation calls or patterns.

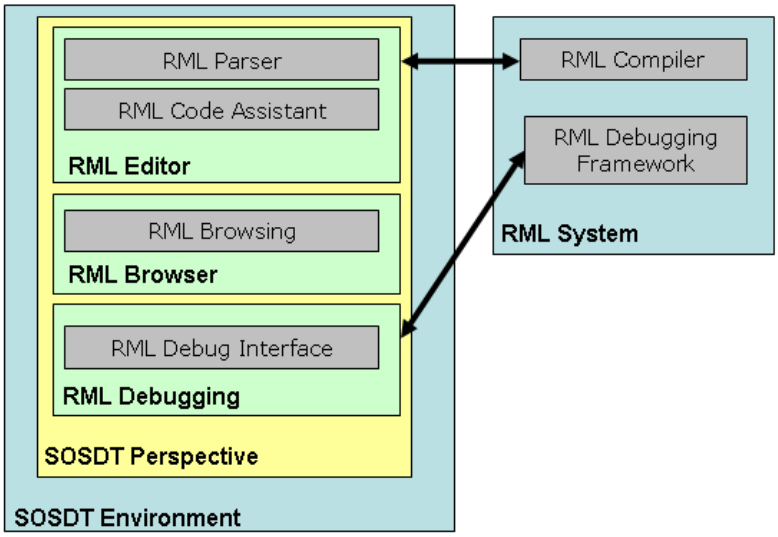

Fig. 1. Architecture of the RML system and SOSDT environment.

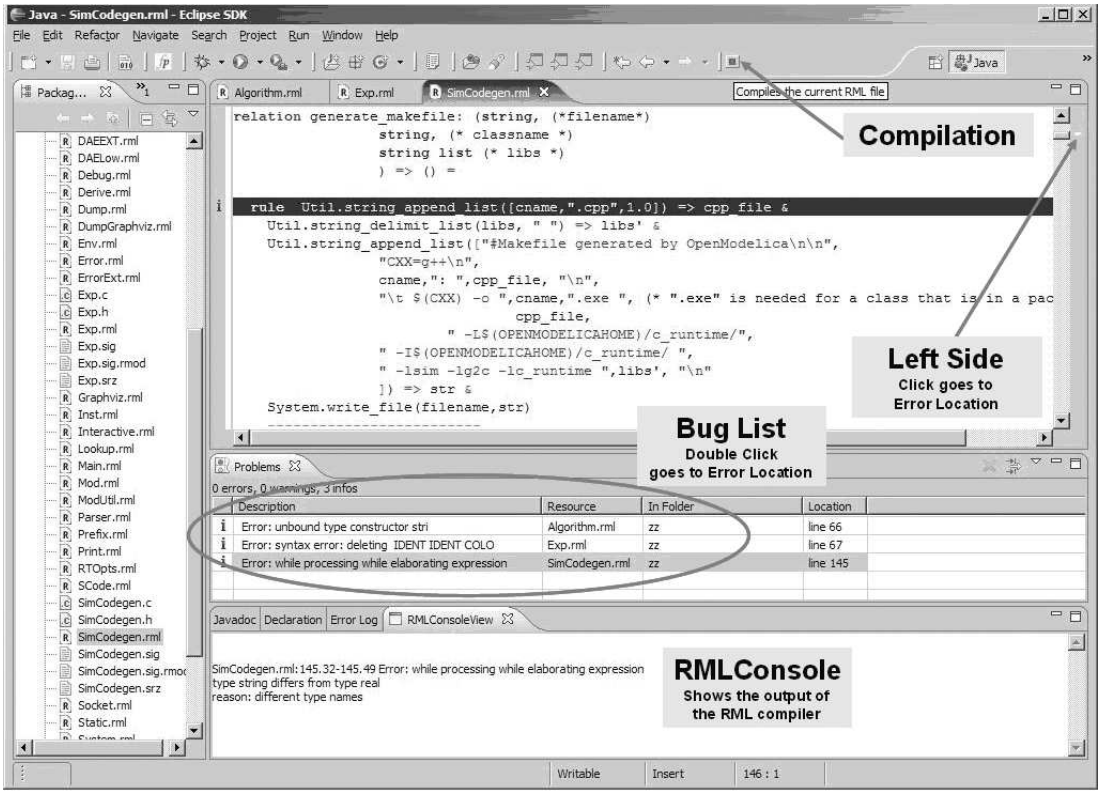

Fig. 2. SOSDT Eclipse Plugin for RML Development.

The RML Browser component provides easy navigation within an file. The RML parser is used to gather the information needed for browsing. The types, values, relations and rules are displayed within a tree for each RML file.

The RML Debugging component communicates via sockets with the RML Debugging Framework to provide debugging facilities like breakpoints, running and

stepping, variable value inspection, etc.

All the SOSDT components are using the components of the Eclipse framework which are populated with information from the RML Parser and the RML Compiler. When a file is saved the RML Parser reads the file and updates the internal RML model information which triggers the update of the RML Browser. Also, on save the RML file is sent to the RML Compiler which dumps error information to be displayed in the Problems View and type information used to update the internal RML model.

4 Performance Evaluation

The test case used for the table below is based on an executable specification (SOS/Natural Semantics in RML) of the MiniFreja language [5] running a test program based on the sieve of Eratosthenes. All the needed information for reproducing the tests are available at http://www.ida.liu.se/~adrpo/sosdt/tests.

Mini-Freja is a call-by-name pure functional language. The test program calculates prime numbers. The Prolog translation (mf.pl) was originally implemented by Mikael Pettersson. The comparison was performed on a Fedora Core4 Linux machine with two AMD Athlon(TM) XP 1800+ processors at 1500 MHz and 1.5GB of memory. The measurements were done during April 2006.

Prime#	RML	SICStus	SWI	Maude-MSOS-Tool
8	0.00	0.05	0.00	2.92
30	0.02	1.42	1.79	226.77
40	0.06	3.48	3.879	-
50	0.13	-	11.339	-
100	1.25	-	-	-
200	16.32	-	-	-

Execution time is in seconds. The sign represents out of memory. The memory consumption was at peak 9Mb for RML. The other systems consumed the entire 1.5Gb of memory and aborted at around 40 prime numbers. The largest executable specification developed so far using RML is the Modelica Language specification (an equation-based language), which is approximately 80 000 lines. We have improved compilation speed more than a factor of 10 since a year ago compiling 80 000 lines of RML now takes less than minute on a 1.5 GHz laptop.

5 Conclusions and Future Work

Our experience of writing large executable specifications in SOS/Natural Semantics style using RML for several different programming languages shows that a supportive development environment is essential also for developing specifications.

Therefore we have designed and implemented a prototype of an integrated environment for supporting such development, first as a version partly based on Emacs, and currently integrated in Eclipse [1], as an SOSDT Eclipse plugin. Some of our RML users who have debugged their specifications using a prototype of this environment have given us positive feedback and also various suggestions for improvement.

While this is a good start, many improvements can be made to this environment. In the future we plan to improve the debugger execution speed, and implement additional features. Our goal is to provide a very well integrated and supportive development environment (IDE) for RML based on the Eclipse platform.

References

[1] EclipseFoundation, *Eclipse Development Platform*, http://www.eclipse.org.

[2] Kahn, G., *Natural Semantics*, in: *Programming of Future Generation Computers. ed Niva M., p. 237.258*, 1998.

[3] Libermann, H., *The debugging scandal and what to do about it*, in: *Communication of the ACM. vol:40(4), p:27-29*, 1997.

[4] PELAB, *Structural Operational Semantics Development Tooling (SOSDT) Eclipse Plugin*, http://www.ida.liu.se/~adrpo/sosdt.

[5] Petterson, M., "Compiling Natural Semantics," Ph.D. thesis, Linköping University (1995), dissertation No. 413,
also as Lecture Notes in Computer Science (LNCS) 1549, Springer-Verlag, 1999,
RML Site: http://www.ida.liu.se/labs/pelab/rml.

[6] Plotkin, G. D., *A Structural Approach to Operational Semantics*, in: *The Journal of Logic and Algebraic Programming 60-61 , 17-139.*, 2004.

[7] Pop, A. and P. Fritzson, *Debugging Natural Semantics Specifications*, in: *Sixth International Symposium on Automated and Analysis-Driven Debugging (AADEBUG2005)*, 2005, Monterey, California.