EpiSpin: an Eclipse Plug-in for Promela/Spin using Spoofax

B. de Vos, L.C.L. Kats, and C. Pronk

Delft University of Technology, The Netherlands
b.devos-1@student.tudelft.nl, l.c.l.kats@tudelft.nl, c.pronk@tudelft.nl

Abstract. This paper presents EpiSpin: an Eclipse plug-in for editing Promela models. It provides error markers as you type, various editor services and an interface to perform verification and simulation runs using Spin. An additional tool shows the static relations between channels, processes and global variables. These tools have been built using the Spoofax language workbench.

1 Introduction

Model Checking [7] is a technique used for state space exploration of a model of a system to determine whether it meets a given specification. In the past, model checking was performed on relatively small models. Currently, increasingly large models are routinely being constructed. Examples of such large models are given in [9, 12, 13]. In the latter paper a model consisting of more than 7000 lines of Promela code was developed and model checked. Current work on model checking FreeRTOS [4, 15] will also lead to large programs. With these developments, it becomes increasingly important to provide state-of-the-art IDE support for efficient development and maintenance of Promela models.

In this paper, we introduce EpiSpin[1], an Eclipse plug-in for editing Promela programs and starting Spin verification and simulation runs. It includes several editor services like syntax highlighting, content completion, code folding and instant feedback on syntactic and semantic errors. Additionally, a tool is integrated which shows the static relations between processes, channels and variables.

Implementing state-of-the-art IDE support for a new language can be a challenge, requiring not only the implementation of a parser and semantic analysis, but also extensive knowledge of the sometimes complicated and highly interdependent APIs and extension mechanisms of an IDE framework. For EpiSpin, we use the locally developed Spoofax language workbench [8] for the development of an Eclipse plugin. As a language workbench, Spoofax abstracts over these implementation artifacts and provides a comprehensive environment that integrates syntax definition, program transformation, code generation, and declarative specification of IDE components.

We continue this paper with a brief introduction and background on Promela, the Spin Model Checker, the Spoofax language workbench and related work

[1] EpiSpin: Eclipse plug-in for Spin

inspiring this paper. The main development work on the Eclipse plug-in will be described in Section 3. Section 4 will describe the Static Communication Analyzer derived from the same Spoofax set-up. Testing the plug-in is covered in Section 5. The conclusions and some future work can be found in Section 6.

2 Background

Spin and Promela Promela is a state of the art modeling language to describe verification models. These models can be analyzed by the iSpin model checker. A grammar definition of the complete Promela language (including version 6 constructs) and a more complete description of Promela and the Spin tool for which Promela has been developed can be found in [6].

Spoofax Spoofax is a platform for the development of textual domain-specific languages with state-of-the-art IDE support. It combines the SDF syntax definition formalism [14] for the declarative specification of syntax with the Stratego [3] program transformation language for the specification of semantics.

Spoofax supports agile development of languages by allowing incremental, iterative construction of language and editor components. It shows editors for the language under development alongside its definition. As soon as a syntax definition has been written, an editor with default syntax highlighting and other syntactic editor services can be used, and customized. Stratego can be used to specify more sophisticated editor services as transformations on abstract syntax trees, to support e.g. semantic error markers or content completion.

Related Work Our project is not the first endeavor to create an Eclipse plugin for Spin and Promela. Two other editors were created before by Rothmaier et al. [12] and Kovše et al. [9]. Those works used the standard, Java-based Eclipse API and follow a traditional architecture for editors, using regular expressions for editor services such as syntax highlighting and code folding. In the case of Kovše, the editor could also process the output of the Spin syntax checker to provide in-editor error markers. Our approach is fundamentally different: we emphasize rich, as-you-type editor feedback to aid developers of Promela models. We use a language workbench for our approach instead of the standard, rather low-level Eclipse API and regular expressions. Using Spoofax, new languages can be developed using a set of grammar production and transformation rules. From this, the workbench generates an editor with a parser that executes (in a background thread) with every change in a Promela model. Based on the parser our editor provides more accurate syntax highlighting and as-you-type syntactic error markers. Internally, it also creates abstract syntax trees used for analyses of the model. With these analyses we provide editor services such as inline error markers without requiring the user to manually invoke the Spin syntax checker. We also provide more editor sophisticated services such as reference resolving and content completion.

3 EpiSpin

EpiSpin includes the following features:

– Promela editor with full syntax support according to Promela language version 6 including the new `for` and `select` keywords, and support for inline specification of LTL properties,
– Instantaneous feedback on syntactic and semantic errors,
– Syntax highlighting, outline view, code folding, code completion and reference resolving,
– Interface to start the Spin verifier and simulator,
– Static Communication Analyzer (see Section 4).

The Spin simulator and verifier can be called from within EpiSpin directly. A Spin command is executed according to the options specified and the resulting textual output is shown in the Eclipse console. The Promela grammar from [5] has been used as a basis to form the SDF rules. Since only context-free grammars can be specified in SDF, EpiSpin currently does not provide full support for macros. As a work-around it is possible to call the Spin syntax checker or to open the model in iSpin.

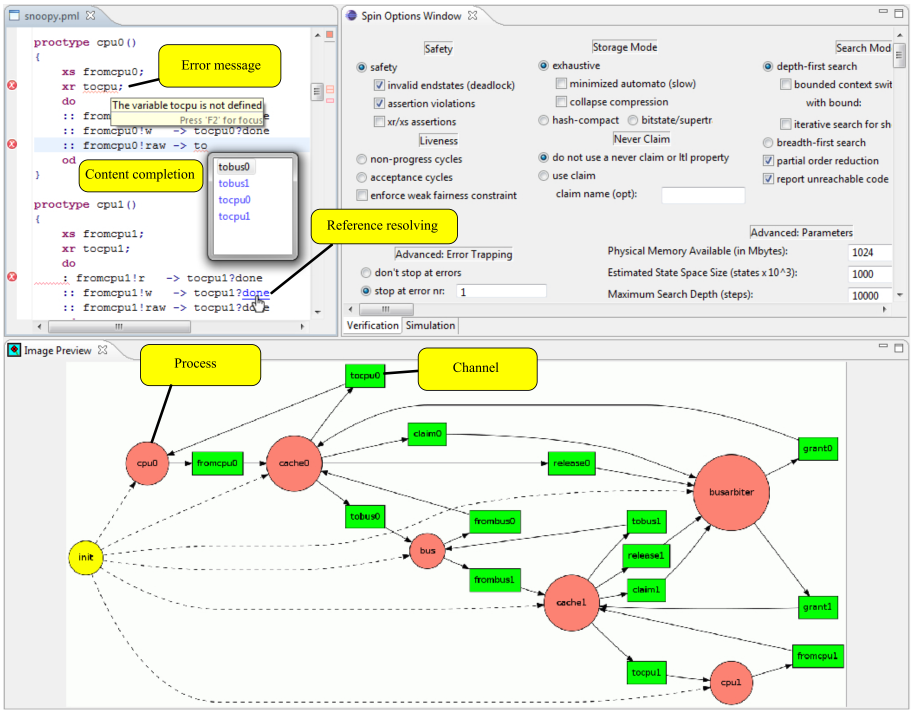
Editor services

In Spoofax, the syntactic editor services can be fully specified using declarative editor service descriptor specifications [8]. Syntax highlighting, code folding and outline view are implemented to improve the readability, especially of large Promela models. These last two editor services are implemented in Spoofax by listing the sorts and constructors for which the editor service needs to hold. Another syntactic editor service is syntactic content completion. Based on static templates, at every moment a list of completion suggestions can be requested.

There are three semantic editor services in EpiSpin, being reference resolving, semantic completion and error checking. All of those are completely integrated in Eclipse. This means that error markers and error messages are shown instantaneously when an error is made. For reference resolving and semantic completion it is necessary to know about the identifiers present in the model. Therefore the first step of the analysis is to find all identifier declarations. This is done by defining a rule that maps the identifier name to its constructor for every declaration. When an instance of an identifier is clicked, the appropiate declaration is found by passing the name of the identifier to the rule. When content completion is triggered at a position where an identifier is expected, a list of all keys of the rule is shown. In Figure 1 a partial screenshot of Eclipse with EpiSpin can be found. Explanations are given in rounded boxes.

4 Static Communication Analyzer

The third author of this paper, while marking student work in Spin, often found himself drawing the communication structure (processes, channels and global


snoopy.pml

```
proctype cpu0()
{
    xs fromcpu0;
    xr tocpu;
    do
    :: from
    :: fromcpu0!w   -> tocpu0?done
    :: fromcpu0!raw -> to
    od
}

proctype cpu1()
{
    xs fromcpu1;
    xr tocpu1;
    do
    : fromcpu1!r   -> tocpu1?done
    :: fromcpu1!w   -> tocpu1?done
    :: fromcpu1!raw -> tocpu1?done
```

The variable tocpu is not defined
Press 'F2' for focus

tobus0
tobus1
tocpu0
tocpu1

Spin Options Window

Safety
[✓ safety]
[✓] invalid endstates (deadlock)
[✓] assertion violations
[ ] xr/xs assertions

Liveness
( ) non-progress cycles
( ) acceptance cycles
[ ] enforce weak fairness constraint

Storage Mode
[✓ exhaustive]
[ ] minimized automato (slow)
[ ] collapse compression
( ) hash-compact ( ) bitstate/supertr

Never Claim
[✓ do not use a never claim or ltl property]
( ) use claim
claim name (opt):

Search Mod
[✓ depth-first search]
[ ] bounded context swit
with bound:
[ ] iterative search for sh
( ) breadth-first search
[✓] partial order reduction
[✓] report unreachable code

Advanced: Error Trapping
( ) don't stop at errors
[✓ stop at error nr:] 1

Advanced: Parameters
Physical Memory Available (in Mbytes): 1024
Estimated State Space Size (states x 10^3): 1000
Maximum Search Depth (steps): 10000

Verification Simulation

Image Preview

Fig. 1. Editor services, option window and dot graph

variables) of the delivered work by hand; a task which can be taken over by this tool. Using rewrite rules, Promela code is transformed into constructors of another language or into code directly. With the Static Communication Analyzer, all processes, global variables and channel operations are displayed in a DOT graph which can be viewed in Eclipse using the ImageViewer[2]. This DOT code is obtained by traversing the AST and creating a node or edge for every process, variable or channel operation that is in the AST. When a process executes another process, this is shown by a dashed arrow. A channel sent as a parameter to another process and therefore existing multiple times as a local channel is pictured as one channel in the graph.

5 Testing the Plug-in

Testing EpiSpin has been done by syntactic testing and semantic testing. Syntactic testing is mainly done by feeding the parser with a lot of different Promela models from [1] and [10]. Since there is no full support for macro calls, all models that include macros are first preprocessed by the `spin -I` command and then parsed by EpiSpin. The editor services are tested during implementation by making a small test program for every rule that is implemented. Mutation testing [11] is used to obtain more test cases.

[2] `http://www.eclipse.org/evangelism/samples/imageviewer/`

6 Conclusions and future work

We created a Promela editor with a Promela parser, various editor services and the possibility to call the Spin verifier and simulator. Additional tools such as the Static Communication Analyzer can easily be derived because of the use of a language workbench. The use of Spoofax makes it it easier to include future changes in the Promela grammar. We are currently looking into work-arounds for the limited support of macro calls.

EpiSpin can be downloaded from `http://epispin.ewi.tudelft.nl` (as a plug-in or as a stand-alone Eclipse distribution). It is not needed to have Spoofax installed since the required libraries are included in EpiSpin. This site will also show how the plug-in can be installed and used.

Acknowledgements This research was supported by NWO/JACQUARD project 612.063.512, *TFA: Transformations for Abstractions*

References

1. Beem: Benchmark for explicit model checkers. `http://anna.fi.muni.cz/models/`.
2. M. Bravenboer, K. T. Kalleberg, R. Vermaas, and E. Visser. Stratego/XT 0.17. A language and toolset for program transformation. *Sci. of Comp. Programming*, 72(1-2):52–70, June 2008.
3. FreeRTOS. The FreeRTOS Project. `http://www.freertos.org`.
4. G. J. Holzmann. Promela language reference. `http://www.spinroot.com/spin/Man/promela.html`.
5. G. J. Holzmann. *The SPIN Model Checker : Primer and Reference Manual.* Addison-Wesley Professional, September 2003.
6. R. Jhala and R. Majumdar. Software model checking. *ACM Comput. Surv.*, 41:21:1–21:54, October 2009.
7. L. C. L. Kats and E. Visser. The Spoofax language workbench: rules for declarative specification of languages and IDEs. In W. R. Cook and et al., editors, *Proceedings of OOPSLA 2010, Reno/Tahoe, Nevada, USA*, pages 444–463. ACM, 2010.
8. T. Kovse, B. Vlaovic, A. Vreze, and Z. Brezocnik. Eclipse plug-in for Spin and st2msc tools-tool presentation. In C. S. Pasareanu, editor, *SPIN*, volume 5578 of *LNCS*, pages 143–147. Springer, 2009.
9. Promela database. `http://www.albertolluch.com/research/promelamodels`.
10. J. Offutt, P. Ammann, and L. L. Liu. Mutation testing implements grammar-based testing. *Mutation Analysis, Workshop on*, 2006.
11. G. Rothmaier, T. Kneiphoff, and H. Krumm. Using SPIN and Eclipse for optimized high-level modeling and analysis of computer network attack models. In *Model Checking Software*, volume 3639 of *LNCS*, pages 236–250. Springer, 2005.
12. P. Taverne and C. Pronk. RAFFS: Model Checking a Robust Abstract Flash File Store. In *Formal Methods and Software Engineering; 11th Intn'l Conf. on Formal Engineering Models, ICFEM2009*, volume 5885 of *LNCS*, pages 226–245, 2009.
13. E. Visser. *Syntax Definition for Language Prototyping.* PhD thesis, University of Amsterdam, September 1997.
14. J. Woodcock. First steps in the verified software grand challenge. *Software Engineering Workshop*, pages 203–206, 2006.