Enabling the usage of formal methods by creation of convenient tools.

Boris Gruschko, Friedrich H. Vogt, and Simon Zambrovski

Hamburg University of Technology

Abstract. Creation of formal specifications is being considered a relief for the difficulties of inception and construction of distributed systems. Numerous formal methods exist for the purpose of description of distributed systems and protocols. The creation of formal specifications for these systems lacks the extensive support by tools vendors. This results in lack of sophisticated tools, which help the developer to overcome the initial training investment, shallowing the learning curve. Thus, the development of formal specification for the systems under construction stays an expensive undertaking, which lacks the immediate results, important for the overall acceptance of formal methods by the industry. In this paper we describe a plugin for the Eclipse IDE, developed to simplify the task of authoring formal specifications in the TLA+ environment. This plugin provides features, expected from an IDE for a common programming language, such as syntax highlighting, autocompletion and execution assistance.

1 Introduction

The need for tooling support for the authoring of formal specification became apparent, after the authors tried to write a simple example specification in the ASCII form of TLA+[6]. The most time consuming task was not the formulation of ideas behind the specification, but the syntactical correctness of the produced ASCII form. Without the support of tools, which would check the syntax, while the specification was being written, the roundtrip consisting of creation of the ASCII form, type setting and model checking, became too long for the immediate feedback about the syntactical correctness of the specification. Tools allowing for easier development of TLA+ specification are available for the Emacs[3] environment. Nevertheless, it is important, to provide a tool set for the environment, the specifier is accustomed to.

The described plugin is an interface between two tool sets. One of them are the TLA+ Tools[1] provided by Microsoft Corporation. TLA+ Tools provide a set of tools for publishing and verification of TLA+ specifications prepared in form of ASCII files. The other tool set is the Eclipse Platform[2]. It provides a generic foundation for the development of IDEs for any programming language. Although TLA+ is not a programming language as is, there are numerous similarities between the processes of authoring a TLA+ specification and a program.

Eclipse Platform has been chosen for the project, because of the numerous prefabricated features which were considered needed for the TLA+ authoring tool. These features are exposed to plugins via clearly defined extension points. Some of the main extension points provided by Eclipse, are convenient text editors, with support for configurable syntax highlighting and autocompletion and programmable execution support. The use of Java[4] as execution platform by Eclipse was another reason for it's choice for the project. TLA+ Tools are implemented in Java too, which makes the integration with Eclipse a seamless one.

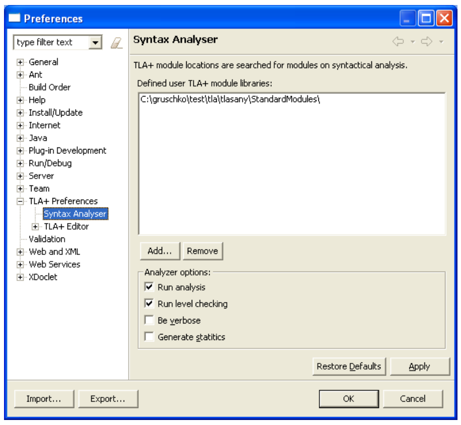
2 Plugin usage

TLA+ Tools provide utilities for typesetting and model checking of TLA+ specifications. It is a collection of command line utilities written in Java language. Although powerful, these tools do not provide a convenient interface for the specifier. These tools are helpful for the usage in conjunction with automation tools and scripts, but tedious for human usage. The plugin maps tools provided by the Eclipse Platform for the authorship of conventional programs, onto similar concepts of TLA+ Tools.

2.1 Development environment setup

To allow the usage of the plugin, Eclipse Platform version 3.0 or later is needed. Eclipse Platform itself depends on Java Runtime Environment, or Java SDK. Both products are available free of charge.

The described plugin provides the functionality needed to setup the development environment for the authoring of TLA+ specifications. The installation of the plugin is performed by the means of standard Eclipse update mechanism via an update site. After being installed the plugin has to be configured, to allow for access to standard TLA+ modules such as *Naturals*. This step is being performed via Eclipse *Preferences* window, as shown in Fig. 1.



Syntax Analyser

TLA+ module locations are searched for modules on syntactical analysis.

Defined user TLA+ module libraries:

C:\gruschko\test\tla\tlasany\StandardModules\

Analyzer options:
- ✓ Run analysis
- ✓ Run level checking
- ☐ Be verbose
- ☐ Generate statitics

Fig. 1. Preferences window

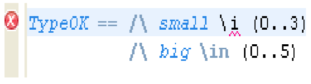
2.2 Specification authoring

Main application of the described plugin is the development of TLA+ specifications. This involves a typical development roundtrip similar to the construction of a normal program. Thus the plugin is mainly constructed to shorten this development cycle. Using the TLA+ Tools, the roundtrip would consist of the specifier creating an ASCII version of the specification, then executing the syntax checker and receiving an output analog to the one shown in Listing 1.1. After reviewing the output, the specifier has to find the erroneous piece of specification, fix it and to execute the next iteration of the roundtrip.

```
*** Errors: 1
line 8, col 21 to line 8, col 21 of module DieHard
Could not find declaration or definition of symbol 'i'.
```

Listing 1.1. Parser output

To shorten this cycle, our plugin provides eager syntax checking. The syntax check will be executed every time the user saves his documents. The same error, which produced the output in Listing 1.1, will cause the plugin to underline the erroneous document fragment, as shown inf Fig. 2. Since the checks are being run in background, the developer of the specification gets an immediate feedback about the syntactical incorrectness of his specification and can react to it.



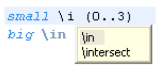
```
TypeOK == /\ small \i (0..3)
          /\ big \in (0..5)
```

Fig. 2. Highlighted syntax error

Further contraction of development roundtrip can be achieved by using the autocompletion feature of the plugin. While the first characters of a token are being typed by the user, the plugin searches it's internal memory for the constructs, which would match the presented characters. If a match is found, the user will be presented with a convenient selection panel. This feature shown in Fig. 3 provides a way, to simplify the process of learning the syntax of TLA+.

2.3 Model checking

TLA+ Tools provide model checking capability for specifications written in the ASCII form of TLA+. This feature maps onto the execution run of a conventional program. Eclipse provides a unified interface for this kind of operations. This feature of Eclipse allows a plugin to describe the needed configuration for an execution run, to start it and to display the results of an execution. All options supported by TLA+ Model Checker are integrated into Eclipse Run Configuration. (see Fig. 4).

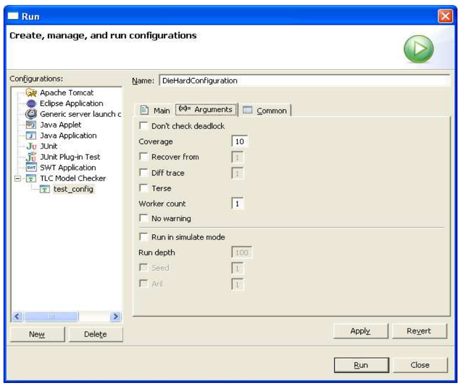


```
small \i (0..3)
big \in
```

Fig. 3. Autocompletion

3 Further Work

TLA+ Tools provide a pretty printer $TLAT_EX$- a program generating LaTeX[7] formatted output. This feature is useful for providing examples of TLA+ in publications. There are some ongoing efforts, to provide a LaTeX Environment for the Eclipse platform. Our further efforts will be directed towards the in integration of TLA+ Pretty Printer into our Eclipse plugin. In addition some effort may be spent on provision of IntelliSence[5] resembling style of autocompletion.


Name: DieHardConfiguration

Configurations:
- Apache Tomcat
- Eclipse Application
- Generic server launch c
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- SWT Application
- TLC Model Checker
  - test_config

Arguments:
- [ ] Don't check deadlock
- Coverage: 10
- [ ] Recover from: 1
- [ ] Diff trace: 1
- [ ] Terse
- Worker count: 1
- [ ] No warning
- [ ] Run in simulate mode
- Run depth: 100
- [ ] Seed: 1
- [ ] Aril: 1

Fig. 4. Model Checker Options

4 Conclusion

Eclipse is a widely used and accepted platform for IDE development. Environments for many programming languages exist on the basis of Eclipse. This leads to a wide adoption of the Eclipse Platform and a large user base experienced in the basic concepts of it. The presented plugin provides a possibility to develop and check TLA+ specifications in the Eclipse environment. The main emphasis of the development was the preservation of user interface concepts proposed by the Eclipse Platform. By this means, the specifier is relieved of the task of learning a new development environment. A specifier with experience in usage of the Eclipse IDE will be able to find the needed functions in a rapid manner. Therefore he can concentrate on the task at hand and start developing the specifications more rapidly. This leads to the possibility of a faster adoption of formal methods by the industry.

References

1. Microsoft Corporation. TLA Tools. Available at http://research.microsoft.com/research/sv/TLA_Tools/, 2005.
2. Eclipse Foundation. Eclipse platform. Available at $http://www.eclipse.org/$, 2005.
3. Free Software Foundation Inc. Emacs. Available at $http://www.gnu.org/software/emacs/emacs.html$, 2005.
4. Sun Microsystems Inc. Java. Available at http://java.sun.com/, 2005.
5. IntelliSence. IntelliSence. Available at $http://www$, 2005.
6. Leslie Lamport. *Specifying Systems*. Addison Wesley, 2003.
7. Leslie Lamport. LaTeX. Available at $http://www.latex-project.org/$, 2005.