# BE[4]: The B Extensible Eclipse Editing Environment *

Jens Bendisposto and Michael Leuschel

Heinrich-Heine Universität Düsseldorf
{bendisposto,leuschel}@cs.uni-duesseldorf.de

## 1 Introduction

The open-source Eclipse platform[1] has become hugely popular as an integrated development environment for Java, and a considerable number of plug-ins have been developed for other programming languages (e.g., C++,PHP, Eiffel, Python, Fortran, etc.). In this paper we present a new plug-in for Eclipse, supporting the B-method and B's abstract machine notation (AMN) [**?**]. In addition to providing editing and syntax highlighting, the plug-in displays syntax and structural errors in the B source code, as well as suggesting fixes for those errors.

## 2 Building a document object model from B

The centerpiece of a semantic-aware editor for programming languages is a parser that generates a model from source-code. In Eclipse, a parser can be integrated by creating a plug-in that extends *org.eclipse.core.resources.builders*. Because we want to allow later contributions to the parser from other plug-ins, we decided to build a multi-phase parsing framework for B projects (Fig. 1).

| Phase | Objective |
|---|---|
| I | Create and modify the syntax tree |
| II | Run file based build tools |
| III | Analyze all resources |
| IV | Run tools to decorate the models |

**Table 1.** Parser phases

For each phase it is guaranteed, that all tools from a previous phase have finished their work. Since Phases I and II work on a file basis, it is possible that the builder[2] for file1 is in Phase II and for file2 in Phase I.

**Phase I** generates an abstract syntax tree (AST) from a B file. This is done by applying a modified version of Tatibouet's jbtools [**?**] Parser. If the

* This research is being carried out as part of the EU funded research projects: IST 511599 RODIN (Rigorous Open Development Environment for Complex Systems).

[1] http://www.eclipse.org/

[2] A builder is a tool that runs every time a project is being rebuilt.

AST-Generator completes its work without error, then other plug-ins can be called. For example, one of our extensions to Phase I checks if the name of the component matches the filename and if the type of the component matches the file extension. Any plug-in that modifies the syntax tree should be run in this phase, that means that in the second phase the syntax tree is stable.

**Phase II** contains file based builders that must not modify the syntax tree, but can create abstractions of the syntax tree or other artifacts. For instance, our standard builder creates a simplified syntax tree that is easier to handle for some of the editor views (like the outline view). Tools in Phase II run if and only if the AST-Generator completed its work without an error.

**Phase III** runs on all resources even when some AST generation failed. This phase can be used to perform a "global analysis". Currently, our standard builder uses this phase to check if all dependencies (SEES, INCLUDES, etc.) are being satisfied. In future, we also plan to check the structuring guidelines from [**?**].

**Phase IV** must not modify the model in any way, it contains plug-ins that only read from the model and update other parts of the plug-ins. For example, our builder uses the final phase to update some properties of the so called markers[3].

In Phase I - III it is also possible to give dependencies for a tool. For example, if a tool C relies on the output of tool A and B it is possible to specify this dependency in the extension configuration. If the dependencies do not contain cycles, the building framework will automatically generate an order via topological sorting.

Our architecture was designed for extensibility by new, as of yet unknown, plug-ins (without this requirement, a simple dependency graph of the various tools would have been sufficient).

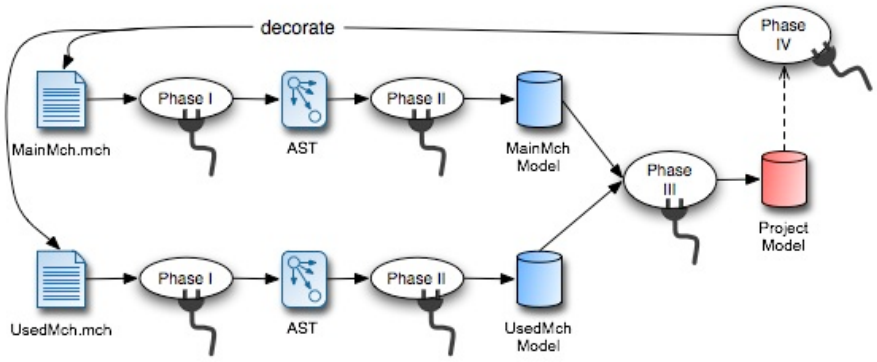


**Fig. 1.** Phases of the building framework

[3] http://www.eclipse.org/articles/Article-Mark My Words/mark-my-words.html

## 3 Using the model for editing B

The model created by the parser can be used for several tasks; so far we have implemented the following features:

- **Context based completion:** Since the editor knows if the user types within an operation or the machine's head, it can choose different proposals to complete the word the user types[4]. In addition, we support templates that contain parts to be filled by the user.
- **Hover information:** The Editor is aware of the token the mouse points to and can display information about that token. For instance, if the mouse points to the token $\setminus/$, the editor displays the information "$S \setminus/ T$: *union of sets* $S$ *and* $T$" as a hover text.
- **Error Displaying and Correction:** (Syntax) Errors are caught and displayed directly in the source code window (line 1 and 18 in Fig. 2) and additionally in a special "Problems view". As shown in the screenshot, the editor supports auto-correction.[5] Based on the error it determines a set of actions called quick-fixes that might be applied to correct the error.
- **Outline view:** The editor produces an outline view of the machine, e.g., the variables used, the operations defined, etc. If the user clicks on any item in the outline, the editor jumps to the line, where the item is being defined.

## 4 Related and Future work

The BZ testing tool (BZTT) [**?**] as well as ProB [**?**] provide simple editing and highlighting, but lack the features of a dedicated editing/development tool. The `EmacsPri`[6] and the more recent Click'N'Prove [**?**] by Dominique Cansell and J.-R. Abrial provide syntax highlighting and an interface to AtelierB within Emacs. Bruno Tatibouet's jbtools package [**?**] also contains a B plug-in for the Java-based Editor jEdit, with syntax-highlighting, type checking and a shortcut pane for the mathematical symbols.

Finally, the Rodin EventB (BSharp) Toolkit[7] is also developed within Eclipse, but has moved away from an ASCII AMN encoding to an internal storage in an XML database of the components of B machines, which can be manipulated directly by various graphical editors. It is actually our goal to combine these two Eclipse plug-ins, so as to also allow editing of EventB components in AMN as well as linking ProB directly to the Rodin EventB core and the associated provers. We are also working on integration of refactorings into the editor, as well as more semantic checks and quick-fixes. Our tool is available from `http://www.stups.uni-duesseldorf.de/ProB/be4`.

---

[4] As in Java mode, the autocompletion can be invoked typing **CTRL+SPACE**

[5] **CTRL+1** invokes the auto-correction

[6] `http://www.atelierb.societe.com/emacspri/emacspri_uk.html`

[7] `http://sourceforge.net/projects/rodin-b-sharp/`

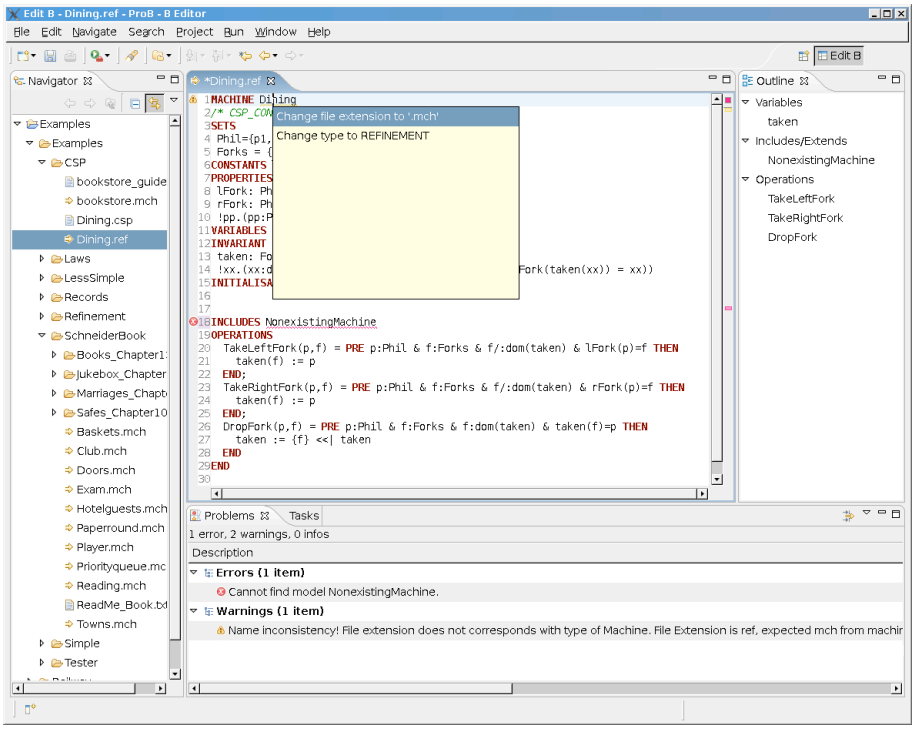


**Fig. 2.** B-Editor screenshot

# References