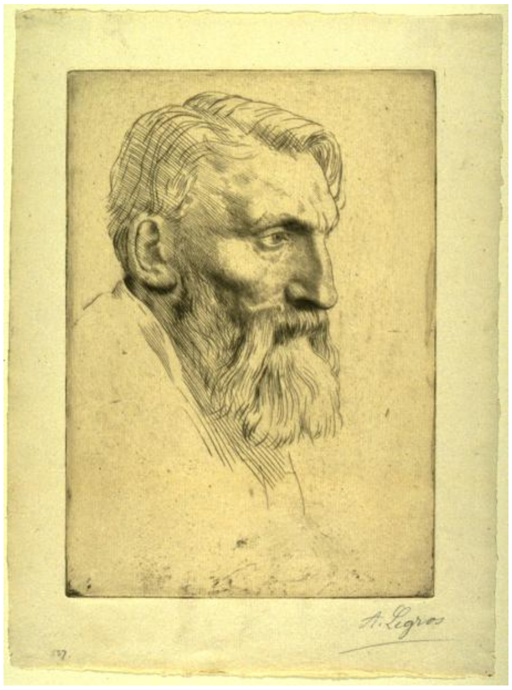
Project IST-511599

RODIN

“Rigorous Open Development Environment for Complex Systems”



RODIN Deliverable D23

Description of the Internal Version of the Rodin Platform

Editor: *Laurent Voisin (ETH Zurich)*

Public Document

February 28, 2007

http://rodin.cs.ncl.ac.uk

Contributors

Jean-Raymond Abrial	ETH Zurich
Stefan Hallerstede	ETH Zurich
Thai Son Hoang	ETH Zurich
Farhad Mehta	ETH Zurich
Christophe Métayer	ClearSy
Thierry Lecomte	ClearSy
Alexander Romanovsky	U. Newcastle
François Terrier	ETH Zurich
Laurent Voisin	ETH Zurich

Contents

1 Introduction

This document describes the contents of Rodin Deliverable D23 *Internal version of basic tools and platform.*

The Rodin platform is an extensible application for developing event-B models and proving them correct. The version presented in this document improves on the Rodin prototype [D15], providing a more mature platform which is used in the Rodin case studies. It is not yet the final platform, which will be delivered at the end of the project.

The implementation of this internal version of the platform corresponds to the advancement of tasks 3.2 to 3.7 of work package 3 at the internal version level (these tasks are carried out in parallel).

After presenting the main achievements of the internal version of the platform, we describe its coverage of the event-B notations. Finally, we give some figures about the size of the platform.

2 Main Achievements

In this section, we present the main differences between the (current) internal and the (previous) prototype version of the Rodin platform (that was delivered as [D15], one year ago). We group the achievements according to the different components of the platform:

- Rodin core,
- event-B static checker,
- event-B proof obligation generator,
- event-B prover,
- event-B user interface.

2.1 Rodin Core

The Rodin Core is made of the Rodin Database (where models, proof obligations and proofs are stored) and the Rodin builder (which incrementally builds projects).

The Rodin database has been improved, since its prototype version, to provide new features that were needed by other kernel tools or plug-in developers. Most notably, attributes on database elements and file element snapshots have been implemented.

The Rodin builder has also been improved to provide even more incrementality and reactiveness to tool developers. Also, its API has been streamlined after comments from plug-in developers, who found it cumbersome and difficult to understand. The new interface is much easier to extend and asks for minimum effort from plug-in developers.

2.2 Event-B Static Checker

The Static Checker that was developed for the prototype was a mock-up developed in order to demonstrate the feasibility of our approach, and to inquire about some implementation difficulties to be encountered. Not surprisingly, it proved to be too complicated, not modular enough, and thus difficult to reuse or extend in current version. Also, its internal architecture made it too slow and memory consuming on large models. Consequently, it has been fully reimplemented from scratch. However, the lessons learned from the prototype were not lost, and the new implementation fixes all issues of the prototype.

The current Static Checker covers all of the core event-B notation (see 3 on the following page). It is decomposed in modules and is extensible: plug-in developers can easily modify the behaviour of the static checker by adding new modules. Consequently, new checks can be added.

Also, the Static Checker computes automatically the tree structure and the order in which modules have to be executed, based on the specification of module dependencies given to its extension point. This allows to extend it without prior extensive knowledge of its internals.

Finally, the Static Checker is configurable: plugin developers can contribute *configurations*, that is lists of modules that are to be executed in one run of the Static Checker. The provided mechanism ensures that configurations remain stable also when the Static Checker is extended.

2.3 Event-B Proof Obligation Generator

Like the Static Checker, the Proof Obligation Generator has been fully reimplemented, as the prototype version was not satisfactory.

The current Proof Obligation Generator covers all of the core event-B notation (see 3 on the next page) and is configurable: plug-in developers can easily add new proof obligations or remove some. Its architecture is very similar to that of the Static Checker (plug-in developers contribute modules which form a tree structure, and can be activated or disabled).

The overall coherence of static checking and proof obligation generation is ensured by means of their configurations: if a static checker configuration bears the same name as a proof obligation one, then they are deemed compatible.

2.4 Event-B Prover

Most of the work around the event-B Prover concerned interactive proof. The initial implementation of the prototype was good for automated proofs, but lacked a lot of necessary features for manual proofs. Also, its persistence framework was too coarse and required unexpectedly high disk space.

The most notable improvements of the prover are the following:

- We have added many new proof commands in interactive mode, that allow to rewrite and simplify predicates and expressions, or capture very common ways of interacting with a prover (modus ponens for instance). The number and level of proof commands is now comparable to the most efficient classical B tools (e.g., Click'n'Prove).

- We have also added an automated tactic to the interactive prover that is run after every user interaction and does many simplifications and cleanup (e.g., removing redundant hypotheses), most of the time by applying classical forward inferences. Thanks to this automated tactic, the user doesn't have to do much clerical work, and can thus focus more easily on the essence of his interactive proof.

- The persistence framework has been drastically improved, so that proofs saved to disk take reasonable space. The improvement is more than tenfold.

- Finally, a plug-in has been developed to connect existing provers to our framework. In the current version, this already allows to connect all provers that take input in the TPTP format. This plug-in indeed demonstrate that the event-B prover is truly extensible.

2.5 Event-B User Interface

The prototype User Interface has been improved to make it more user friendly. As it is the first thing that users see when they try the Rodin platform, a lot of comments were made on the interface. Most of them have been taken into account, so that using the Rodin platform is now much more smooth than with the prototype.

Also, a number of features have been added to the user interface. Firstly, as the prototype was covering only a small subset of the event-B notation, the interface has been extended to provide full coverage (adding most notably refinement).

Secondly, a new view has been developed, that displays an event-B model pretty-printed in textual form, as if it was a source file. This corresponds to a strong requirement of some users, that sometimes got lost with the database oriented interface.

Finally, alongside the prover improvements, the interface has been extended to allow easy use of the new commands. Now, the formulas displayed during a proof session are more than mere pieces of text, they also provide means for manipulating them easily using the mouse, so that most of a proof can now be entered without using the keyboard.

3 Coverage

This internal version of the platform covers all of the core event-B notation:

- All the mathematical language, as defined in [D7] is fully supported.

- Contexts are fully supported (carrier sets, constants, axioms and theorems). Also, one can extend a context by another context.

- Machines are fully supported (variables, invariants, variant and events) and all event elements are supported (local variables, guards, actions and witnesses). Also, a machine can see several contexts and refine another machine. The only feature of [D7] that was not implemented is the concept of event arrays. The reason for this is that this concept was quite new when added to [D7], and after review of the language, it was found not useful: it was just a variation on the concept of local variable that is already covered by the core language.

- All context and machine proof obligations are generated, including refinement, as well as event convergence, proof obligations.

4 Development

We first present the coding conventions and guidelines used when developing the Rodin platform, then we give some interesting figures about the code size of the Rodin platform.

4.1 Coding Guidelines

The development of the platform has been performed according to the regular Eclipse conventions and guidelines that follows:

- Eclipse's Naming Conventions (`http://wiki.eclipse.org/index.php/Naming_Conventions`),
- Sun's Code Conventions for the Java Programming Language (`http://java.sun.com/docs/codeconv/index.html`),
- Sun's Requirements for Writing Java API Specifications (`http://java.sun.com/products/jdk/javadoc/writingapispecs/index.html`),
- Sun's How to Write Doc Comments for Javadoc (`http://java.sun.com/products/jdk/javadoc/writingdoccomments/index.html`),
- Eclipse's User Interface Guidelines (`http://wiki.eclipse.org/index.php/User_Interface_Guidelines`),

4.2 Some Figures

The internal version of the platform is made of 90 000 lines of source code (160 000 lines with comments and white lines).

The validation test suite is made of 35 000 lines of source code (50 000 lines with comments and white lines).

References

[D5] C. Métayer et al. *Final Decisions.* Rodin Deliverable D3.1 (D5). February 28, 2005.

[D7] C. Métayer et al. *Event-B Language.* Rodin Deliverable D3.2 (D7). May 31, 2005.

[D10] L. Voisin (Ed) *Specification of Basic Tools and Platform.* Rodin Deliverable D3.3 (D10). August 31, 2005.

[D15] L. Voisin (Ed) *Description of the Rodin Prototype.* Rodin Deliverable D3.4 (D15). February 28, 2006.