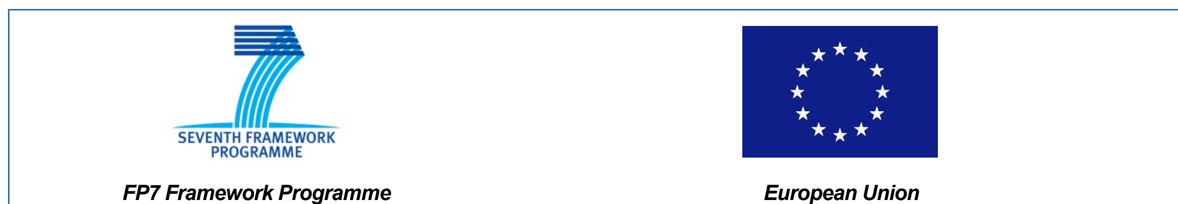
D.3.4 - Methods and Tools for Model Construction and Proof III

ADVANCE

Grant Agreement: 287563
Date: 30/11/2014
Pages: 30
Status: Final
Authors: Laurent Voisin, Nicolas Beauger Systerel
Reference: D3.4
Issue: 1

Partners / Clients:



FP7 Framework Programme European Union

Consortium Members:

University of Southampton | Critical Software Technologies | Alstom Transport | Systerel | Heinrich Heine Universität | Selex ES

Project ADVANCE
Grant Agreement 287563
"Advanced Design and Verification Environment for Cyber-physical System Engineering"

ADVANCE Deliverable D3.4

Methods and tools for model construction and proof III

Public Document

November 25, 2014

http://www.advance-ict.eu

Contributors:

Asieh Salehi	University of Southampton
Colin Snook	University of Southampton
Andy Edmunds	University of Southampton
Lukas Ladenberger	University of Duesseldorf
Sebastian Krings	University of Duesseldorf
Michael Leuschel	University of Duesseldorf
Laurent Voisin	Systerel
Nicolas Beauger	Systerel

Reviewers:

Michael Butler University of Southampton

Contents

1 Introduction 5
2 General Platform Maintenance 7
2.1 Core Rodin platform 7
2.1.1 Overview 7
2.1.2 Motivations / Decisions 7
2.1.3 Available Documentation 8
2.1.4 Conclusion 8
2.2 UML-B Improvements 8
2.2.1 Overview 8
2.2.2 Motivations / Decisions 9
2.2.3 Available Documentation 9
2.2.4 Conclusion 10
2.3 ProR/Rodin Integration Plugin 10
2.3.1 Overview 10
2.3.2 Motivations / Decisions 10
2.3.3 Available Documentation 11
2.3.4 Conclusion 11
2.4 Camille 11
2.4.1 Overview 11
2.4.2 Motivations / Decisions 12
2.4.3 Available Documentation 12
2.4.4 Conclusion 13
3 Improvement of automated proof 15
3.1 Overview 15
3.2 Motivations / Decisions 15
3.3 Available Documentation 16
3.4 Conclusion 16
4 Model Checking 17
4.1 Overview 17
4.2 Motivations / Decisions 17
4.2.1 Linear Temporal Logic 17
4.2.2 Theory Support 21
4.2.3 Physical Units 22
4.2.4 B to TLA+ 22
4.2.5 Performance Improvements 23
4.3 Available Documentation 23

4.4 Conclusion 24
5 Language extension 25
5.1 Overview 25
5.2 Motivations / Decisions 25
5.3 Available Documentation 26
5.4 Conclusion 27
6 Model Composition and Decomposition 29
6.1 Overview 29
6.2 Motivations / Decisions 29
6.3 Available Documentation 29
6.4 Conclusion 30

1 Introduction

The ADVANCE D3.4 deliverable is composed of the present document and new extensions to the Rodin toolset as of November 2014. The considered Rodin toolset consists of the Rodin core platform and the plug-ins created or maintained in the frame of the ADVANCE project: ProB, UML-B, ProR, Camille, Theory, Composition, SMT. Other plug-ins, available for the Rodin platform but not maintained within the ADVANCE project, are not taken into account within this deliverable.

The Rodin platform can be downloaded from the SourceForge site.[1]

Moreover, the platform includes a collaborative documentation that is collected from two maintained locations:

- the Event-B wiki,[2]
- the Rodin Handbook.[3]

These locations can be consulted from outside the Rodin tool.

The present document intends to give a relevant overview of the work achieved within work package 3 *Methods and Tools for Model Construction and Proof*, during the final period of the ADVANCE project (Oct 2013 - Nov 2014), and aims to let the reader understand the WP3 member's contribution plans and objectives.

The document is divided according to the work package tasks: general platform maintenance, improvement of automated proof, model checking, language extension, model composition and decomposition.

The common structure which is used for each contribution is the following:

- Overview. The involved partners are identified and an overview of the contribution is given.
- Motivations / Decisions. The motivation for each tool extension and improvement are expressed. The decisions (e.g. design decision) are reported.
- Available documentation. Some pointers to the available documentation or related publications are listed.
- Conclusion. A brief summary about what has been achieved in the topic, and an overview of potential continuations.

---

1 `http://sourceforge.net/projects/rodin-b-sharp/files/Core_Rodin_Platform`
2 `http://wiki.event-b.org`
3 `http://handbook.event-b.org`

2 General Platform Maintenance

This part describes the general maintenance performed on the Rodin toolset within the last year of the ADVANCE project. As the maintenance is a task that concerns the whole toolset, and to ease the reading of this part of the deliverable, the maintenance section has been decomposed in a list of subsections corresponding to scopes of the toolset. All these subsections maintain the template previously defined in the introduction.

2.1 Core Rodin platform

2.1.1 Overview

During the last period of the ADVANCE project, the following versions of the Rodin platform have been released:

- 3.0.0 (2014-03-20)
- 3.0.1 (2014-06-11)
- 3.1.0 (2014-11-28)

The main part of the work was targeting improvements of

- stability of the platform,
- prover capabilities,
- theory support

Other running tasks consisted in answering questions on mailing lists, and processing bug tickets and feature requests.

2.1.2 Motivations / Decisions

The core API has been improved. In particular, the interface with the theory plug-in has been enriched so as to keep track of language changes over time, thus enforcing model and proof consistency.

New proof rules have been implemented (SIMP_EMPTY_PARTITION and SIMP_SINGLE_PARTITION). The proving experience has been enhanced with HMI that provide quick access to user-defined tactics.

The Rodin Editor stability has been improved, by fixing bugs pointed out by users.

The underlying Eclipse Platform has been upgraded to the most recent version (Eclipse 4.4.1 as of Rodin 3.1). This was required by plug-in developers who wanted to take advantage of the newest Eclipse features. While also bringing fixes to some Eclipse bugs, it prepares for future developments based on the latest technology.

For the same reasons, the code has been ported to Java 7, while remaining compatible with Java 6.

The source code is now distributed in a more developer-friendly format, so it's easier for plug-in implementers to extend Rodin as a target platform.

2.1.3 Available Documentation

The release notes, that appear and are maintained on the wiki, and that accompany each release, give useful information about the changes introduced by each. Moreover, two web trackers list and detail the known bugs and open feature requests:

- a sourceforge bug tracker,[1]
- a sourceforge feature requests tracker.[2]

2.1.4 Conclusion

The Rodin Platform has made significant steps in throughout the ADVANCE project. Improvements concern stability, dependability, extensibility, modeling and proving capability and, globally, convergence towards user expectancies.

2.2 UML-B Improvements

2.2.1 Overview

UML-B provides Class diagrams and State-machine diagrams for Event-B modelling. There are two versions of UML-B. The first, Classic UML-B, generates a complete Event-B project and all modelling is diagrammatic. The second, iUML-B, integrates diagrams inside Event-B machines and contexts and a mixture of diagrams and Event-B can be used in modelling.

During the last period of ADVANCE,

- Classic UML-B has been migrated to Rodin 3.x. During this release some improvements and bug fixes were made.

1 http://sourceforge.net/p/rodin-b-sharp/bugs/
2 http://sourceforge.net/p/rodin-b-sharp/feature-requests/

- iUML-B State-machines have undergone major functional and structural improvements including new diagram features requested by users (in particular, Critical Software in WP2) and a new Event-B generator that significantly improves performance. State-machine Animation has been integrated with BMotion Studio so that both can be run in parallel.
- iUML-B Class diagrams have been released as a prototype and are under evaluation by industrial collaborators for use in their development process.

2.2.2 Motivations / Decisions

While Classic UML-B is still maintained and used by some industrial collaborators, most new development focusses on the more popular iUML-B.

The iUML-B Statemachine to Event-B translator was implemented in the QVTo declarative language. This suffered performance problems due to the need to recompile on each invocation. The translation was re-written using our own Java based generator framework which greatly improves performance.

For valiation purposes users find visual animations very powerful and we have used both the Statemachine animation and BMotionStudio in our collaborations with industry. However, previously, only one or the other could be launched at any particular time. A facility to also automatically launch any associated and open BMotionStudio editor was added to the launch process of the state-machine editor. Both animations run from the same ProB animation instance. This provides an extremely strong visualisation of the model for validation purposes as the image representations of BMotion Studio complement the state-machine visualisation and vice versa.

The iUML-B class diagram plug-in was delayed as it was not the highest priority for ADVANCE partners. However, other industrial users (e.g. Thales, Austria) are interested in switching away from the Classic UML-B tool. A prototype release of iUML-B Class diagrams has now been made available to these industrial collaborators for evaluation purposes. A full release will be made by the end of 2014. During recent development of the plug-in, the mechanism for data elaboration has been rationalised and additional modelling features have been added which make the modelling tool more effective.

2.2.3 Available Documentation

An overview and tutorial of both Classic UML-B and iUML-B can be found on the Event-B wiki[3].

3 http://wiki.event-b.org/index.php/UML-B

2.2.4 Conclusion

iUML-B state-machines and animation have been significantly improved throughout ADVANCE and now represent a mature and popular tool. iUML-B class diagrams promise to be equally successful as early indications are that the approach to data elaboration is effective.

Classic UML-B will continued to be maintained but no new features are being added unless specifically requested by an industrial partner.

2.3 ProR/Rodin Integration Plugin

2.3.1 Overview

ProR is a tool for working with requirements in natural language. It is part of the Eclipse Requirements Modeling Framework (RMF).[4] The goal of the ProR/Rodin integration plugin is to bring two complimentary fields of research, requirements engineering and formal modelling, closer together. The ProR/Rodin integration plugin supports the user by maintaining a traceability between natural language requirements and Event-B models.

A requirements Meta-Model for the WP-1 and WP-2 industrial case studies has been developed during the last period of the ADVANCE project. Moreover, several requested features were added. For instance, when the user selects a linked element (guard, invariant, etc.) in the requirements document, the tool takes the user directly to the corresponding element in the Event-B model. First experiments have been made towards a feature to automatically generate reports from ProR[5]. Beside this, general improvements, such as usability improvements have been made on the ProR/Rodin integration plugin during the last period of the ADVANCE project.

2.3.2 Motivations / Decisions

The ProR/Rodin integration plugin provides a default Meta-Model for requirements documents. However, this Meta-Model does not support all specific needs and characteristics of the industrial case studies. As a consequence, we decided to create a new requirements Meta-Model that supports the specific needs of both industrial case studies.

In order to improve the usability and the integration between the ProR tool and the Rodin platform, a new feature has been added to switch between the requirements document and the Event-B model. Whenever the user selects a linked element (guard, invariant, etc.) in the requirements document, the tool takes the user directly to the corresponding element in the Event-B model.

4 http://www.eclipse.org/rmf/
5 http://www.stups.hhu.de/w/Reporting_f%C3%BCr_ReqIF:_Von_der_Analyse_bis_zur_Auswertung

The ability to generate reports is important. For instance, it is desirable to see statistics (the number of requirements currently covered by the model) so that this can be presented to a customer during progress meetings. In order to meet this goal, first experiments towards a reporting tool for ProR have been made during the ADVANCE Project[6].

2.3.3 Available Documentation

- *A Method and Tool for Tracing Requirements into Specifications.*[7] Accepted for Science of Computer Programming.
- *Requirements Traceability between Textual Requirements and Formal Models Using ProR*[8]. The paper has been accepted for iFM'2012 & ABZ'2012.
- A Tutorial for the Rodin/ProR integration[9] can be found on the Event-B wiki.
- The User Guide[10] contains additional tutorials for ProR.

2.3.4 Conclusion

The customized Meta-Models facilitated the requirements maintenance process of both industrial case studies. Moreover, the new Meta-Model is expandable so that new features can be easily added later in the project.

2.4 Camille

2.4.1 Overview

The Camille plug-in provides a textual editor for Rodin. This editor provides the same look and feel as a typical Eclipse text editor, including features most text editors provide, such as copy, paste, syntax highlighting and code completion.

During the last period of the ADVANCE project, three new versions of Camille have been released:

- 3.0.0 - Initial release for version 3 of the Core Rodin platform. This release has been based on Camille 2.1.4.

6 http://www.stups.hhu.de/w/Reporting_f%C3%BCr_ReqIF:_Von_der_Analyse_bis_zur_Auswertung
7 http://www.stups.uni-duesseldorf.de/mediawiki/images/e/ec/Pub-HalJasLad2013.pdf
8 http://www.stups.uni-duesseldorf.de/w/Special:Publication/LadenbergerJastram_iFMABZ2012
9 http://wiki.event-b.org/index.php/ProR
10 http://wiki.eclipse.org/RMF/User_Guide

- 3.0.1 - Port of the changes done in Camille 2.2.0 to version 3. This includes theorems in guards as well as other bugfixes. See D3.3 for details.
- 3.0.2 - Camille's structure parser has been moved to ProB's parser library. A fully automatic build process featuring continuous integration has been set up. This is the first release build by it.

One of the main goals of the last period was the support of Rodin's extensibility in Camille.

2.4.2 Motivations / Decisions

Move to Git / GitHub

The source files for Camille have been moved from the old Rodin SVN repository to their own repository at GitHub. The old source files have been marked deprecated. Furthermore, the move to GitHub allows us to use GitHub's infrastructure for bug tracking and feature requests. We moved old feature requests from the wiki pages to the bug / feature tracking systems at GitHub.

Build Process

Before version 3.0.2 was released, the Camille build was mostly done by hand. This turned out to be slowing down development during the last period of the ADVANCE project. Starting with release 3.0.2 we completely revamped the build process. Camille is now build automatically on each commit using a Jenkins continuous integration server [11]. This facilitates the build as well as the release process for Camille. Furthermore, it should ease collaborative development.

Move Structure Parser to ProB's Parser Library

Camille's internal parser for the structure of Event-B machines and contexts has been split off of Camille and moved to ProB's general parsers library. Hence, the parser is now a completely separate project that can be developed independently of Camille. This further decouples Camille's core, ui and parsers. Externalising the parser is the first step to making Camille more modular in oder to be able to replace the parser by the upcoming block parser. In addition, externalising the parser makes it available for other projects as well.

2.4.3 Available Documentation

- *Architectures for an Extensible Text Editor for Rodin.*[12] Bachelor thesis analysing the problem and discussing possible solutions.
- An earlier version of the thesis has been published as a technical report[13] that has been discussed on the Roding Developers Mailing List and the ADVANCE Progress Meeting in May 2012 in Paris.

---

11 http://www.jenkins-ci.org
12 http://www.stups.uni-duesseldorf.de/mediawiki/images/0/0a/Pub-Weigelt2012.pdf
13 http://www.stups.uni-duesseldorf.de/w/Special:Publication/Weigelt2012>

- *Camille GitHub Repository and Bugtracker*: https://github.com/hhu-stups/camille
- *Camille Wiki*: http://wiki.event-b.org/index.php/Camille_Editor

2.4.4 Conclusion

Camille still has the drawback of not supporting extensibility. It only supports the core Event-B language and plug-in-specific additions are simply ignored. Consequently, users have to switch back to Rodin's native Editor to edit plug-in-specific modelling extensions. However, Camille was extended in oder to preserve plug-in specific additions. This change allows switching between Camille and the other editors without loosing plugin specific information that Camille can not display. The changes and improvements to the development process performed in the last period should finally allow for a new and completely overhauled version of Camille to be implemented.

3 Improvement of automated proof

3.1 Overview

In a regular Event-B modelling activity, most of the time is spent on interactive proofs. Therefore, increasing the rate of automated proofs is a productivity booster which decreases the overall cost of formal modelling. Consequently, enhancing the automated prover has been a continuous task since the inception of the Rodin platform.

During the third period of the ADVANCE project, priority has been given to stability and improvement, and enhanced user experience.

3.2 Motivations / Decisions

SMT Solvers

The SMT Solvers plug-in brings the glue that allows to use external SMT solvers for discharging proof obligations in the Rodin platform. The SMT plug-in stability has been improved by fixing several bugs. We have also added buttons to the Rodin proving interface in order to give direct access to SMT tactics. Finally, the translation to SMT solvers has been enhanced by supporting user-defined operators and passing them as uninterpreted functions. This allows an SMT solver to be used on any proof obligation, even the ones that contain operators defined by the Theory plug-in (as far as the proof does not depend on the operator semantics, which is usually the case). AWE, an external user of Rodin, reported a 31% increase in automatic proof by using the SMT plugin. See AWE presentation at the ADVANCE Industry Day [1].

ProB as a Disprover / Prover

Usage of ProB as a disprover has been facilitated. Counter-examples are now displayed in an easier to understand fashion. Error messages have been cleared up.

A more in-depth empirical evaluation of using ProB as a prover has been performed [2], using a novel plug-in to compare the performance of different provers [3]. As observed in the initial case studies, ProB is able to find proofs that are not found by the integrated provers or the SMT solvers. For

1 `http://www.advance-ict.eu/industry_days`
2 `http://www.stups.uni-duesseldorf.de/mediawiki/images/2/24/Pub-sets14.pdf`
3 `https://github.com/wysiib/ProverEvaluationPlugin`

certain classes of proof obligations, i.e. those that heavily rely on enumerated or otherwise finite sets, ProB performs very favourably.

The disprover plug-in now offers the possibility to export proof obligations from inside Rodin into a custom (Prolog-based) format. These files can then be used as test cases for the disprover. We created several initial test cases that cover both provable as well as unprovable obligations.

Several other changes to the disprover plug-in have been made:

- As ProB itself, the disprover and prover are completely aware of theories. See the general information on ProB's theory support for details.
- Based on the empirical evaluation, several improvements have been made to ProB's kernel. While some of them also improve constraint solving in general, other are especially tailored for proof.
- A new setting has been introduced to allow ProB to react differently in animation / constraint solving and (dis-)proving. Mostly, this changes the treatment of well-definedness conditions during constraint solving.

3.3 Available Documentation

Tactic profiles are described in the Rodin Handbook[4].

The user manual of the *SMT Solvers* plug-in is available on the Event-B wiki[5].

Information regarding the Disprover is collected on [6].

3.4 Conclusion

During the last period of the ADVANCE project, we have been increasing the rate of automatically discharged proofs. It has been achieved by improving existing provers, both internal and external ones. Support for new external provers has been evaluated. Some of them have been rejected because of licence issue (*iProver*), lack of maturity (*Super Zenon*), or lack of evidence of result improvements (*CVC4*). However for *CVC4*, more recent attempts with a different set of proof obligations tend to show otherwise, so its integration can be considered.

4 http://handbook.event-b.org/current/html/preferences.html#ref_01_preferences_auto_post_tactic
5 http://wiki.event-b.org/index.php/SMT_Solvers_Plug-in
6 http://wiki.event-b.org/index.php/Disprover

4 Model Checking

4.1 Overview

In this period we have worked on scalability of the tools, by improving ProB for theories, for the WP1 and WP2 case study demands, and by adding dedicated support for fairness properties. We have also provided a link to the TLC model checker for large state space of more concrete formal models. We have also added new features to the model checking core of ProB, in feedback from the other work packages.

4.2 Motivations / Decisions

4.2.1 Linear Temporal Logic

LTL is a specification language used for specifying temporal properties of a system. A possible requirement that could refer to WP1 may be, for example, *once a block is reserved, it will eventually be occupied*. The requirement could be easily written as an LTL formula using the temporal operators `G` (always) and `F` (eventually):

```
`G ({reserved(B) = TRUE} => F {occupied(B) = TRUE})`, where B is a block of
some track segment in the railway network and the expressions inside the curly
brackets `{...}` are B predicates.
```

LTL Fairness

ProB provides support for checking temporal properties expressed by means of LTL (Linear Temporal Logic) or CTL (Computation Tree Logic).[1] In ProB linear temporal properties can be expressed by an extended version of LTL, denoted as LTL[^{e}], which additionally enables the user to set propositions on transitions. Writing propositions on transitions is allowed by using the con-

1 D. Plagge and M. Leuschel: *Seven at one stroke: LTL model checking for High-level Specifications in B, Z, CSP, and more.* STTT, 12(1): 9-21, Feb 2010

structs `e(...)` and `[...]`. (For more information on the syntax and semantics of LTL[e] consult [2].)

Fairness is a notion where the search for counterexamples is restricted to paths that do not ignore infinitely the execution of a set of enabled actions imposed by the user as fair constraints. One possibility to set fairness constraints in ProB is to encode them in the LTL[e] formula intended to be checked. For example, for a given LTL[e] formula "f" a set of weak fairness conditions {e1,…,en} (where e1,...,en are some events) can be given as follows:

```
(FG e(e1) => GF [e1]) & … & (FG e(en) => GF [en]) => f.
```

In a similar way, strong fairness constraints can be imposed expressed by means of an LTL[e] formula:

```
(GF e(e1) => GF [e1]) & … & (GF e(en) => GF [en]) => f.
```

Checking fairness in this way is very often considered to be inefficient as usually the number of atoms (the possible valuations of the property) of the LTL property is exponential in the size of the formula [3] [4].

For this reason, the search algorithm of the LTL model checker in ProB has been extended in order to allow fairness to be checked efficiently. In addition, new operators have been added to ProB's LTL parser for setting fairness constraints to an LTL[e] property. The new operators are WF(-) and SF(-) and both accept as argument an operation. The fairness constraints must be given by means of implication: "fair => f", where "f" is the property to be checked and "fair" the fairness constraints.

In particular, "fair" can have one of the forms: "wfair", "sfair", "wfair & sfair", and "sfair & wfair", where "wfair" and "sfair" represent the imposed weak and strong fairness constraints, respectively.

Basically, "wfair" and "sfair" are expressed by means of logical formulas having the following syntax:

- Weak fair conditions ("wfair"):
  - `WF(a)`, where `a` is an operation
  - `&` and `or`: conjunction and disjunction
- Strong fair conditions ("sfair"):
  - `SF(a)`, where `a` is an operation

2 ProB User Manual: LTL Model Checking, `http://www.stups.uni-duesseldorf.de/ProB/index.php5/LTL_Model_Checking`

3 O. Lichtenstein and A. Pnueli: *Checking that Finite State Concurrent Programs Satisfy Their Linear Specification*. POPL '85, Proceedings of the 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, ACM, 1985

4 D. Plagge and M. Leuschel: *Seven at one stroke: LTL model checking for High-level Specifications in B, Z, CSP, and more*. STTT, 12(1): 9-21, Feb 2010

- `&` and `or`: conjunction and disjunction

More information on setting fairness to LTL formulas and the LTL Model Checker is available on the ProB User Manual website [5].

Use Case

Consider an Event-B model formalizing an algorithm for mutual exclusion with semaphores for two concurrent processes P_1 and P_2. Each process has been simplified to perform three types of events: *request* (for entering in the critical section), *enter* (entering the critical section), and *release* (exiting the critical section). (For more information on the algorithm and the design of the model see [6]).

Each of the actions of a process are represented by a respective event:

```
event Req1
  when
   @grd1 p1=noncrit1
  then
   @act1 p1 ≔ wait1
  end

event Enter1
  when
   @grd1 p1=wait1
   @grd2 y=1
  then
   @act1 p1 ≔ crit1
   @act2 y≔0
  end

event Rel1
  when
   @grd1 p1=crit1
  then
   @act1 p1 ≔ noncrit1
   @act2 y≔1
  end

event Req2
  when
   @grd1 p2=noncrit2
  then
   @act1 p2 ≔ wait2
  end

event Enter2
  when
   @grd1 p2=wait2
   @grd2 y=1
  then
   @act1 p2 ≔ crit2
   @act2 y≔0
  end

event Rel2
  when
   @grd1 p2=crit2
  then
   @act1 p2 ≔ noncrit2
   @act2 y≔1
  end
```

Figure 1

Each process P_i has three possible states that are denoted by the constants $noncrit_i$ (the state in which P_i performs noncritical actions), $wait_i$ (the state in which P_i waits to enter the critical section), and $crit_i$ (representing the state in which P_i is in the critical section). Both processes share the binary semaphore y where y=1 indicates that the semaphore is free and y=0 that the semaphore is currently processed by one of the processes.

5 ProB User Manual: LTL Model Checking, `http://www.stups.uni-duesseldorf.de/ProB/index.php5/LTL_Model_Checking`

6 C.Baier and J.-P. Katoen. "Principles of Model Checking", The MIT Press, 2008, pages 43-45.

There are several requirements that the mutual exclusion algorithm should satisfy. The most important one is the mutual exclusion property that says *always at most one process is in its critical section*. This can be simply expressed, for example, as an invariant of the respective Event-B model: not(p1 = crit1 & p2 = crit2). However, there are other properties that can be easily expressed by means of LTL formulas and automatically checked on the model. For example, the requirement *each process will enter infinitely often its critical section* can be specified by the LTL formula `GF {p1 = crit1} & GF {p2 = crit2}` or the starvation freedom property that states *each waiting process will eventually enter its critical section*:

```
G ({p1 = wait1} => F {p1 = crit1}) & G ({p2 = wait2} => F {p2=crit2})
```

Running the LTL model checker of ProB will provide for the last two properties above a counterexample since the model permits that a process may perform infinitely often consecutively the events *request*, *enter* and *release*, and in this way to ignore the other process infinitely. An example trace that describes this behavior could be $\langle Req2, Req1, Enter1, Rel1, Req1, Enter1, \ldots\rangle$.

On the other hand, such behaviors can be considered as unrealistic computations for the eventual implementation of the algorithm. Therefore fairness constraints can be set in order to discard such behaviors. For example, checking the property *process* P_1 *will enter its critical section infinitely often* (as LTL property: `GF {p1 = crit1}`) can be checked by restricting that the event `Req1` will not be continuously ignored and that the event `Enter1` will not be ignored infinitely often. Both conditions on the property can be given by means of an LTL[e] formula on the right side of the implication as follows:

```
(FG e(Req1) => GF [Req1]) & (GF e(Enter1) => GF [Enter1]) => GF {p1 = crit1}
```

For checking the formula the LTL model checker generates 13312 atoms and 7515 transitions and needs overall 509 ms to prove the property. On the other hand, using the extension of the LTL model checker for checking fairness (by entering the formula `WF(Req1) & SF(Enter1) => GF {p1=crit1}`), the model checker generates 32 atoms and 39 transitions (the atoms and transitions generated just for `GF {p1 = crit1}`) and an overall time of 50 ms.

For checking the requirement *each process will enter infinitely often its critical section* the LTL formula ` GF {p1 = crit1} & GF {p2 = crit2}` should be checked with the following fairness constraints:

```
(WF(Req1) & WF(Req2)) & (SF(Enter1) & SF(Enter2))
```

Encoding these fairness conditions as an LTL[e] formula will blow up exponentially the number of atoms and the transitions and thus make practically impossible to check the above property in a reasonable time.

New LTL Patterns

The ABZ landing gear case study [7] was formalized in Event-B and validated with ProB. [8] In the course of the validation of the model new features have been developed in ProB for checking relative deadlock freedom and determinism.

Since it has turned out that the features were a key part of the validation of the model and the authors believe that they will be of use for other Event-B system developments, the LTL model checker has been extended to support these features as new patterns within LTL[e] formulas:

- deadlock(e1,e2,...,ek), where e1,e2,...,ek with k>0 are events. Used to check relative deadlocks, more precisely to check if a set of events are disabled in a state.
- deterministic(e1,e2,…,ek), where e1,e2,...,ek with k>0 are events. Used to check that maximum one of the events in the parentheses is enabled in a state. Note that if the atomic proposition is checked in a state where no one of the events e1,e2,...,ek is enabled, the test will succeed.
- controller(e1,e2,…,ek), where e1,e2,...,ek with k>0 are events. Used to check that exactly one of the events e1,e2,...,ek is enabled in a state.

Referring to the landing gear case study from [9], one important issue in the process of validation of the model was to guarantee that the controller behaves in a determinisitc way. In the corresponding model the behavior of the controller is divided into several events: con_stop_stimulate_extend_gear_valve, con_stimulate_extend_gear_valve, con_stop_stimulate_retract_gear_valve, etc. The deterministic behavior of the controller could, for example, be tested by using new LTL pattern `deterministic(e1,e2,…,ek)` within an LTL[e] formula:

```
 G deterministic(con_stop_stimulate_extend_gear_
valve,con_stimulate_extend_gear_valve,con_stop_stimulate_retract_gear_valve,...)
```

and then check the LTL[e] formula with the ProB's LTL model checker.

4.2.2 Theory Support

Theory Support was relevant for a variety of case studies, and is relevant for simulation, model checking and proving. We ensured that the Disprover also works with theories. We have also improved the constraint propagation of the ProB kernel for records and freetypes, which are used to represent Event-B inductive datatypes. (As a side note, this feature is also being used for validating

7 F. Boniol, V. Wiels, "The Landing Gear System Case Study", ABZ 2014

8 D. Hansen, L. Ladenberger, H. Wiegard, J. Bendisposto, M. Leuschel, "Validation of the ABZ Landing Gear System using ProB", ABZ 2014: The Landing Gear Case Study, 2014, pages 66-79, Springer International Publisher.

9 D. Hansen, L. Ladenberger, H. Wiegard, J. Bendisposto, M. Leuschel, "Validation of the ABZ Landing Gear System using ProB", ABZ 2014: The Landing Gear Case Study, 2014, pages 66-79, Springer International Publisher.

VDM specifications using ProB within the Ouverture[10] tool [11]. Ouverture is the subject of ongoing and past EU projects, e.g., Destecs[12] or Compass[13].). Finally, the treatment of recursive functions within the ProB kernel has been improved, also in light of dealing with recursive operators of Event-B Theories. More details can be found in deliverable D4.4 (ProB constraint solving kernel).

4.2.3 Physical Units

The physical units analysis has been further stabilised, several reported bugs have been fixed. Support for physical units has been extended to theories along with the general theory-related improvements of ProB mentioned in the previous paragraph. The plug-in was ported to Rodin 3, all bugfixes and changes could be back ported to Rodin 2 successfully.

Further extension to the unit analysis include:

- Support for the analysis of units throughout refinement relations.
- Support for abstract units like "length" that can later be concretised to standard SI units.

A journal version of the SEFM'13 article [14] has been submitted.

4.2.4 B to TLA+

We developed a translation from B to TLA+ to verify B specifications with TLC. TLC is an explicit state model checker for TLA+ providing a parallel and a distributed mode. It is particularly good for lower level specifications, where it can be substantially faster than ProB's own model checker.

Moreover, we are interested in validating the correctness of our translation from B to TLA+. Hence, we have conducted extensive tests to validate our approach. For example, we use a range of models encoding mathematical laws to stress test our translation. These have proven to be very useful for detecting bugs in our translation and libraries. In addition, we have uncovered a bug in the model checker TLC. Moreover, we use a wide variety of benchmarks, checking that ProB and TLC produce the same result and generate the same number of states

The current version of our translator covers almost all operators of a abstract B machine. Moreover, TLC can be used to validate liveness properties (LTL formulas) for B specifications under

---

10 http://overturetool.org

11 Lausdahl, Kenneth; Ishikawa, Hiroshi; Larsen, Peter Gorm. Interpreting Implicit VDM Specifications using ProB. 2014. Abstract from 12th Overture Workshop on VDM, Newcastle, United Kingdom. http://wiki.overturetool.org/index.php/12th_Overture_Workshop.

12 http://destecs.org

13 http://www.compass-research.eu

14 Sebastian Krings, Michael Leuschel, "Inferring Physical Units in B Models. SEFM'2013, LNCS 8137, p. 137-151."

fairness conditions. Our approach has been published at the ABZ'2014 conference in Toulouse [15]. A technical report is available [16].

4.2.5 Performance Improvements

Various performance improvements have been made to the model checker and animator for Event-B models, both in terms of memory consumption and speed. For example, ProB now executes the models from WP1 and WP2 considerably faster than at the beginning of the project and than at the beginning of the last period of the project. Another example is an Event-B model of the Early parser by JR Abrial (a standard benchmark used for ProB regression testing) is now running an order of magnitude faster than before the beginning of the project.

4.3 Available Documentation

ProB

The ProB Website[17] is the place where we collect information on the ProB toolset. There are several tutorials on ProB available in the User manual section[18]. We also supply documentation on extending ProB for developers[19]. Documentation and tutorials on the new LTL features available[20]. Recently, a tutorial for the LTL counter-example view in Rodin has been written[21].

The physical unit support is explained here[22].

The TLC for B model checking support is explained here[23].

In addition we run a bug tracking system[24] to document known bugs, workarounds and feature requests.

15 D. Hansen and M. Leuschel: *Translating B to TLA + for Validation with TLC*. ABZ'14, LNCS 8477, p.40-55, June 2014
16 `http://stups.hhu.de/w/Special:Publication/HansenLeuschel_TLC4B_techreport`
17 ProB Website: `http://www.stups.uni-duesseldorf.de/ProB/`
18 ProB User Manual: `http://www.stups.uni-duesseldorf.de/ProB/index.php5/User_Manual`
19 ProB Developer Manual: `http://www.stups.uni-duesseldorf.de/ProB/index.php5/Developer_Manual`
20 ProB User Manual: LTL Model Checking, `http://www.stups.uni-duesseldorf.de/ProB/index.php5/LTL_Model_Checking`
21 LTL View in Rodin: `http://www.stups.uni-duesseldorf.de/ProB/index.php5/Tutorial_LTL_Counter-example_View`
22 Unit Plugin: `http://www.stups.uni-duesseldorf.de/ProB/index.php5/Tutorial_Unit_Plugin`
23 TLC: `http://www.stups.uni-duesseldorf.de/ProB/index.php5/TLC`
24 Bug Tracking System: `http://jira.cobra.cs.uni-duesseldorf.de/`

4.4 Conclusion

With TLC4B we have provided a new scalable model checking approach for low-level B models. It should be particularly interesting for hardware models or lower-level models with very large state spaces. Fairness is often important for real-life temporal properties; by adding these we have made the ProB LTL model checker much more convenient and effective to use. The new LTL patterns arose multiple times in the various case studies; by providing them directly within the LTL language we have made the specification task much easier. Finally, theories in Rodin are very important and have played an important role in both WP1 and WP2 case studies. The support provided by ProB was thus essential for the success in these work packages.

5 Language extension

5.1 Overview

Mathematical extensions have been co-developed by Systerel (for the Core Rodin Platform) and Southampton (for the Theory plug-in). The main purpose of this feature was to provide the Rodin user with a way to extend the standard Event-B mathematical language by supporting user-defined operators, basic predicates and algebraic types. Along with these additional notations, the user can also define new proof rules (proof extensions).

The Theory plug-in provides, among other things, a user-friendly mechanism to extend the Event-B mathematical language as well as the prover. A theory is the dedicated component used to hold mathematical extensions (datatypes, operators with direct definitions, operators with recursive definitions and operators with axiomatic definitions), and proof extensions (polymorphic theorems, rewrite and inference rules). Theories are developed in the workspace (akin to models), and are subject to static checking and proof obligation generation. Proof obligations generated from theories ensure any contributed extensions do not compromise the soundness of the existing infrastructure for modelling and proof. In essence, the Theory plug-in provides a systematic platform for defining, validating and using extensions while exploiting the benefits brought by proof obligations.

5.2 Motivations / Decisions

Supporting mathematical and proof extensions has been a longing for the Event-B community for considerable time. Serious considerations have been made to ensure any support ensures: 1) ease of use, and 2) soundness preservation. The Theory plug-in became a natural candidate to provide support for mathematical and proof extensions. The use of proof obligations goes a long way in preserving the soundness of the underlying Event-B formalism.

In the past 5 months:

- Two versions of the Theory plug-in is released. [1] The releases include several bug fixes.
- A set of standard theories and some models using these theories are developed . The standard library of the theories is available to download here[2].

1 http://wiki.event-b.org/index.php/Theory_Release_History

2 https://sourceforge.net/projects/rodin-b-sharp/files/Theory_StdLib/StandardTheory0.1.zip/download

This library includes:

- BasicTheory project: including theories of BinaryTree, BoolOps, List, PEANO, SUMand-PRODUCT and Seq.
- RelationOrderTheory project: including theories of Connectivity, FixPoint, Relation, Well_Fondation, closure, complement and galois.
- RealTheory project: including theory of Real.

Also it includes three Event-B models that use these theories:

- Data project: using SUMandPRODUCT theory
- Queue project: using Seq theory
- SimpleNetwork project: using closure theory

During last 4 months, the migration of Theory plug-in for Rodin v3.1 is being perfumed. This work involves major changes in managing the compatibilities in the Theory plug-in using the APIs from Rodin core v3.1.

5.3 Available Documentation

Pre-studies (states of the art, proposals, discussions):

- Proposals for Mathematical Extensions for Event-B.[3]
- Mathematical Extension in Event-B through the Rodin Theory Component.[4]
- Generic Parser's Design Alternatives.[5]

Technical details (specifications):

- Mathematical_Extensions wiki page.[6]
- Constrained Dynamic Lexer wiki page.[7]
- Constrained Dynamic Parser wiki page.[8]
- Theory plug-in wiki page.[9]

User's guides:

- Theory Plug-in User Manual.[10]

---

3 http://deploy-eprints.ecs.soton.ac.uk/216/
4 http://deploy-eprints.ecs.soton.ac.uk/251/
5 http://wiki.event-b.org/index.php/Constrained_Dynamic_Parser#Design_Alternatives
6 http://wiki.event-b.org/index.php/Mathematical_Extensions
7 http://wiki.event-b.org/index.php/Constrained_Dynamic_Lexer
8 http://wiki.event-b.org/index.php/Constrained_Dynamic_Parser
9 http://wiki.event-b.org/index.php/Theory_Plug-in
10 http://wiki.event-b.org/images/Theory_Plugin.pdf

5.4 Conclusion

The Theory plugin plays an important role in enhancing the useability of the Rodin in terms of extending the Event-B modelling language and provers. The latest release (2.0.2) can be considered as a stable release for Rodin v2.8, and is used in WP1 and WP2 as below:

- In WP1 it was used to develop theories of graphs and chains to represent rail network topology, train positions and train movements.
- In WP2 is was used for modelling arithmetic operations on data collections (generalised product and sum) for the voltage controller.

Also an external industrial user, Thales, has used the Theory plug-in for representing variability in modelling of railway interlocking systems. In particular Thales developed a generic model of an interlocking system and represented operational rules specific to particular rail operators as mathematical operators in theories. Different rules are represented as different theories for the same generic model. The Thales presentation at ADVANCE Industry Day is available here[11].

11 http://www.advance-ict.eu/industry_days

6 Model Composition and Decomposition

6.1 Overview

Composition is the process by which it is possible to combine different sub-systems into a larger system. Known and studied in several areas, this has the advantage of re-usability and combination of systems especially when it comes to distributed systems. One of the most important feature of the Event-B approach is the possibility to introduce new events during refinement steps, but a consequence is an increasing complexity of the refinement process when having to deal with many events and many state variables. Model decomposition is a powerful technique to scale the design of large and complex systems. It enables first developers to separate components development from the concerns of their integration and orchestration. Moreover, it tackles the complexity problem mentioned above, since decomposition allows the partitioning the complexity of the original model into separated components. This allows a decomposed part of the model to be treated as an independent artifact so that the modeller can concentrate on this part and does not have to worry about the other parts. Composition and decomposition can be seen as inverse operations: while composition starts with different components that can be assembled together, decomposition starts with a single components that can be partitioned into different components.

6.2 Motivations / Decisions

Recent updates to the composition and decomposition features are:

1) Support for flattening: this is needed in order to compose refinement chains, rather than just machines at a single abstraction level. It was requested by the Critical Software partner, in order to apply composition to the smart grid case study.

2) Porting to Rodin 3 as part of the tool maintenance activity.

6.3 Available Documentation

Details about decomposition can be found here[1]

1 `http://wiki.event-b.org/index.php/Decomposition_Plug-in_User_Guide`

Details about composition can be found here[2]

6.4 Conclusion

Composition/decomposition have been applied in the interlocking case study of WP1 and the smart grid case study of WP2. WP2 required the addition of a feature to flat refinement chains in order to compose them. Composition/decomposition has been ported to Rodin 3.

2 http://wiki.event-b.org/index.php/Parallel_Composition_using_Event-B