ADVANCED DESIGN AND VERIFICATION ENVIRONMENT
FOR CYBER-PHYSICAL SYSTEM ENGINEERING
www.advance-ict.eu

D.3.3 - METHODS AND TOOLS FOR MODEL CONSTRUCTION AND PROOF II

ADVANCE

Grant Agreement: *287563*
Date: *29/11/2013*
Pages: *33*
Status: *Final*
Access: *Access List*
Reference: *D3.3*
Issue: *1*

Partners / Clients:

FP7 Framework Programme ***European Union***

Consortium Members:

University of Southampton ***Critical Software Technologies*** ***Alstom Transport*** ***Systerel*** ***Heinrich Heine Universität Düsseldorf***

Project ADVANCE

Grant Agreement 287563

"Advanced Design and Verification Environment for Cyber-physical System Engineering"

ADVANCE Deliverable D3.3

Methods and tools for model construction and proof II

Public Document

November 29, 2013

http://www.advance-ict.eu

Contributors:

Laurent Voisin	Systerel
Nicolas Beauger	Systerel
Asieh Salehi	University of Southampton
Colin Snook	University of Southampton
Andy Edmunds	University of Southampton
Lukas Ladenberger	University of Dusseldorf
Jens Bendisposto	University of Dusseldorf
Daniel Plagge	University of Dusseldorf

Reviewers:

Fernando Mejia Alstom

Contents

1 Introduction

The ADVANCE D3.3 deliverable is composed of the present document and new extensions to the Rodin toolset as of September 2013. 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 the work package 3: *Methods and Tools for Model Construction and Proof*, during the second period of the ADVANCE project (Sept 2012 - Sept 2013), 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.
- Planning. The current status about the topic, and an overview of the future plans are given.

Finally, a section is devoted to link the work planned for the last period of the project with the needs of WP1 and WP2 case studies.

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

- 2.7 (2012-11-05),
- 2.8 (2013-06-20),
- 3.0 (2013-09-30).

The main focus of the maintenance activities was twofold : firstly, to accommodate the needs of the case-study workpackages WP1 and WP2, and secondly to improve the soundness of the core tools and to ease the development of external plug-ins by providing easier to use Application Programming Interface (API). Thus, releases 2.7 and 2.8 mainly contain bug fixes and small improvements, while version 3.0 is a clean up of the core mechanisms preparing for the future.

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

2.1.2 Motivations / Decisions

Besides implementing the features needed and fixing the bugs detected by the case studies, which always have top priority, the maintenance of the core platform in the second period of the project was devoted to fixing soundness bugs in the core platform, cleaning up the API and providing structure to the Rodin database.

Starting from a usability issue reported by the WP2 case study, we dug into the roots of the reported problem. This led us to find out that the core platform, although working well in general, was too permissive as concerns mathematical extensions. It was possible to declare meaningless empty data

types, to freely mix incompatible mathematical extensions and several other oddities. In all these cases, the invariants normally guaranteed by the core platform and on which the soundness of the proofs rely were violated. Although the Rodin editors did not allow the users to input such invalid entries, these API weaknesses were a major concern as regards overall platform reliability. It was therefore needed to strengthen the way mathematical extensions could be added to the mathematical language to ensure that proofs are always sound.

The latest major version of the Rodin platform was Rodin 2.0, released in October 2010. Since that release, a lot of new features have been developed, but always keeping the API backward compatible, to comply with the Eclipse standard of release numbering. Consequently, old versions of features, marked as deprecated, had to be kept along with new ones. The middle of the ADVANCE project was considered the best time for performing a clean up of old API parts by releasing a new major version.

Finally, the Rodin database up to Rodin 3.0 was not enforcing any database schema. This seemed a good idea initially, but in retrospect, this lack of schema made life unnecessarily difficult for plug-in developers. Firstly, it was difficult to find out which database items were considered by each plug-in, as the only place where this information was present was in either the plug-in documentation (if present) or buried into the plug-in code. Moreover, if by chance a plug-in wrote database elements in the wrong place, nothing would indicate it, which led to nasty and difficult to analyse bugs. We have therefore added a notion of schema in the database of Rodin 3.0.

Concerning model editors, we have improved readability in Event-B editor by moving a few buttons outside the editing area. This enhancement originated from a user who has contributed a patch for it. The stability of the newer generic Rodin editor has also been improved.

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]

Specially for Rodin 3.0 which introduces incompatible changes, a Plug-in Migration Guide[3] has been written.

The Event-B wiki[4], basic source of documentation for the users and developers of the Rodin toolset, is completed by the Rodin handbook, an ultimate source of documentation which reached completion by the end of the DEPLOY project. The handbook aims at overcomimg the lack of a centralized

1 http://sourceforge.net/p/rodin-b-sharp/bugs/
2 http://sourceforge.net/p/rodin-b-sharp/feature-requests/
3 http://wiki.event-b.org/index.php/Rodin_3.0_Plug-in_Migration_Guide
4 http://wiki.event-b.org/

source of information providing the necessary assistance to an engineer in the need to be productive using Event-B and minimize the access to an expert user. It is continuously maintained by the various actors involved in the environment of the Rodin toolset and is available in PDF and HTML format, as well as integrated in the on-line help within the Rodin platform. Both the Rodin handbook and the Event-B wiki represent the main source of documentation about Event-B and the Rodin toolset.

Finally, a channel has been created on Youtube. It provides video tutorials about using the platform.[5]

2.1.4 Planning

The plans for the third and last period of the ADVANCE project are to come back to a four month schedule between releases:

- 3.1 in January 2014,
- 3.2 in May 2014 and
- 3.3 in September 2014.

As usual the main driver for the maintenance activity will be the feedback provided by the case studies.

2.2 UML-B Improvements

2.2.1 Overview

UML-B provides a diagrammatic front end to Event-B to assist in constructing models. UML-B is UML-like. It is semantically and syntactically different from UML but should feel familiar to UML users. There are two variants of UML-B. Classic UML-B is self contained (i.e. all modelling is done on the diagram) and a read-only Event-B project is automatically generated for verification. Classic UML-B provides project diagrams to show the machine refinement structure, class diagrams to provide object oriented lifting and state-machine diagrams to provide event sequencing. The more recent iUML-B (i for integrated) consists of a collection of diagrams which contribute additional modelling aspects to an extant Event-B machine (i.e. some of the modelling is still done in an Event-B machine). iUML-B currently provides a project diagram, and state-machine diagram. iUML-B class diagrams are currently under development and component diagrams are planned for future developemnt. The University of Southampton is responsible for UML-B.

5 `http://www.youtube.com/user/EventBTv`

2.2.2 Motivations / Decisions

In general, classic UML-B is maintained but not substantially developed. Some improvements have been made in response to user requests. These include improved property fields for text entry and fixing a handles leak that causes problems on the Windows platform.

The development of iUML-B comprises improvement of the State-machines plug-in and development of a new class diagram plug-in.

The following improvements have been made to the state-machines plug-in:

- It is now possible to animate several state-machines simultaneously within the same machine.
- It is often necessary to have several editors open on the same machine which can lead to conflicting changes. The default option has been changed to save automatically whenever the diagram editor loses focus.
- Often, it is desired to change or delete an event while working on transitions. A property sheet button to remove an event from a transition and delete it at the same time has been added.
- If a diagram is deleted from a machine, any associated diagram files are now deleted at the same time.
- A number of inconsistencies in the translation to Event-B have been corrected

The following improvements are currently being implemented:

- It is sometimes needed to elaborate several events but apply a common guard and/or action to them all. Transitions may now own guards and actions which are generated into all of its elaborated events.
- The current translations require a new enumeration for every state-machine. Often, it would be convenient to have several state-machines with the same type or to utilise an existing enumeration for type. A special case of this is a 2-state state machine that would most naturally be represented as a single Boolean value. This will be done by allowing a state machine to be linked to an existing variable.
- In some cases it would be useful to split a state-machine into several overlaid diagrams, for example, to segregate some kinds of transitions. This will be done by allowing a state machine to be linked to an existing variable.
- There is sometimes a requirement for a transition to originate from a group of source states and it is not always possible to deal with this via hierarchical states. This will be implemented via a pseudo-state that represents the merging of the source transitions.

The major features of the class-diagram plug-in are:

- Classes, attributes and associations link to (elaborate) existing data items (sets, constants or variables) that are in scope of the containing machine or context (via refines, sees or extends). This allows further attributes and associations to be added to an existing class which may be based in another machine or context.
- Class methods are linked to existing Event-B events in a many-many relationship providing a flexible mechanism to allocate parts of an event to classes based on responsibility.

2.2.3 Available Documentation

Documentation on classic UML-B: http://wiki.event-b.org/index.php/Original_UML-B

Documentation on iUML-B: http://wiki.event-b.org/index.php/IUML-B

Tutorial on iUML-B statemachines: http://www.youtube.com/watch?v=nz7ZpL2JtAM

2.2.4 Planning

The current aims are to complete the remaining improvements to the statemachines plug-in and complete the release of the Class diagrams plug-in. It is hoped that these plug-ins will provide a very flexibly diagrammatic modelling notation. The plug-ins will then be evaluated on a case study.

2.3 ProR

2.3.1 Overview

The ProR/Rodin integration plugin is developed and maintained by Lukas Ladenberger and Michael Jastram at the University of Duesseldorf. ProR is a tool for working with requirements in natural language. It is part of the Eclipse Requirements Modeling Framework (RMF).[6] 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 formal models.

The following significant contributions to the latest version of the ProR/Rodin integration plugin have been made during the second period of the ADVANCE project:

- Phenomena support (maintaining a glossary and tracing phenomena)
- Integration of the tracing method for producing a high quality system description[7]

Other improvements will include more general improvements, such as usability, and any features required by the projects industrial partners.

2.3.2 Motivations / Decisions

Phenomena Support

6 http://www.eclipse.org/rmf/

7 http://www.stups.uni-duesseldorf.de/mediawiki/images/e/ec/Pub-HalJasLad2013.pdf

System descriptions are composed of phenomena and artefacts: phenomena describe the state space and state transitions of an environment and a system, while artefacts describe constraints on the state space and the state transitions. A (concrete) collection of all phenomena of a system description is called glossary of phenomena. The user is now able to maintain a glossary of phenomena in the ProR/Rodin integration plugin. The phenomena are automatically available in the different artefacts. For instance, an auto complete feature is available to the user while editing an artefact. In addition the user can define synonyms for phenomena as well as custom colours for highlighting the different phenomena in the artefacts.

Tracing Method Integration

While all required data structures exist to support the tracing method, the ProR/Rodin integration plugin would benefit from more sophisticated reporting. In particular, the tracing method lists a number of properties of a correct system description. For example, the existence of a trace between an artifact and its used phenomenon, or the fact that domain properties, requirement items and design decisions may only be expressed referring to phenomena that are visible in the environment, whereas specification elements and platform properties may only be expressed referring to phenomena that are visible to the system. While the presence of such properties does not guarantee correctness, their absence indicates a problem. A *requriements analyzer* that lists all violations of those properties would be valuable to support the user by maintaining a system description of high quality. We developed a first system description for such requriements analyzer by means of the tracing method. We applied Event-B for modelling and validating the different artefacts of the system description and demonstrated how the formal model can be used as a first prototype in the early development phase.

2.3.3 Available Documentation

- *A Method and Tool for Tracing Requirements into Specifications*.[8] Accepted for Science of Computer Programming.
- *Requirements Traceability between Textual Requirements and Formal Models Using ProR*[9]. The paper has been accepted for iFM'2012 & ABZ'2012.
- A Tutorial for the Rodin/ProR integration[10] can be found on the Event-B wiki.
- The User Guide[11] contains additional tutorials for ProR.

2.3.4 Planning

The following work is planned:

8 http://www.stups.uni-duesseldorf.de/mediawiki/images/e/ec/Pub-HalJasLad2013.pdf
9 http://www.stups.uni-duesseldorf.de/w/Special:Publication/LadenbergerJastram_iFMABZ2012
10 http://wiki.event-b.org/index.php/ProR
11 http://wiki.eclipse.org/RMF/User_Guide

- Further work on the development of the system description for the requriements analyzer tool that supports the user by maintaining a consistent system description
- Applying the tracing method on the case studies of the industrial partners

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.

The latest release finally supports the full core Event-B language as it is supported by the built-in editors too.

2.4.2 Motivations / Decisions

A new version of the editor has been published to support the full feature of the core Event-B language. Event guards can now be defined as theorems as it is possible in the other Rodin editors. With this version all core Event-B models can be edited solely with Camille. It is no longer necessary to switch between different Editors.

To further improve the user experience, the syntax was slightly modified. With the new syntax all model elements now occur in the same order in every editor and pretty print. This finally lead to a more consistent modelling experience within the Rodin platform.

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.

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>

2.4.4 Planning

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. A new version of Camille will be implemented during the ADVANCE project. Plug-in Developers will be able to provide syntax contributions and parsers for their extensions. By this users will be able to edit extended Event-B models solely through a text editor.

3 Improvement of automated proof

3.1 Overview

In a regular Event-B modelling activity, more than 60% of the time is spent on 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.

This task can be achieved by refactoring the core platform, by adding new features to it, such as new integrated reasoners and tactics, but also by connecting some external reasoning ability such as external provers (e.g., SMT solvers).

During the second period of the ADVANCE project, all these three kinds of activities have been performed.

3.2 Motivations / Decisions

Internal code refactoring

The *membership goal* reasoner has been part of the integrated sequent prover since Rodin 2.3. This tool provides specialized reasoning capability about pure set membership and chains of set inclusion. For instance, it can be used to discharge sequents like $x \in A, A \subset B, B \subset C \vdash x \in C$.

Internally, the reasoner used a specialized algorithm to discover chains of inclusions and all hypotheses matching $x \in \_$. Trying to extend the reasoner to other cases of inclusion such as $A \cup B \subset C$, we found out that the reasoner was in fact solving a SAT problem where all atoms are of the form $x \in \_$. So, we stepped back and replaced the specialized machinery by a call to the SAT4J solver which is already part of Eclipse (it is used to solve dependencies when installing a new plug-in). This refactoring, although not increasing yet the rate of automated proofs, opens the way for further improvements of this reasoner, and this without any penalty, as the new version of the reasoner takes the same amount of time to run (as shown by benchmarks).

Integrated provers

During this period, we have implemented new normalization rules about empty sets and their dual : complete sets (that is sets that fill entirely their type). These new rules are now part of the normalizer

which is run at each step of the interactive prover. They reduce the diversity of predicates and increase the automated proofs of trivial proof obligations often encountered in degenerated cases (that is not interesting to the modeller, but that need to be carried out nevertheless).

In a similar vein, we have also improved the *HYP*, *HYP_OR*, *CNTR* and *GEN_MP* reasoners. All these reasoners have in common to search for a hypothesis matching a certain pattern. Until Rodin 3.0, only exact matches were sought for. Now, the reasoners also look for variations of a predicate, that is either equivalent predicates or predicates that provide a sufficient condition. Benchmarking this improvement, we measured that we had an overall 4 % increase in automated proofs compared to the previous version. As for the *membership goal* reasoner, this improvement comes at no cost as the new versions of these reasoners does not take more time to execute.

Tactic profiles

The proportion of automatically discharged proof obligations heavily depends on the auto-tactic configuration. Sometimes, the automatic prover fails because the tactics are applied in an unappropriate order. Since Rodin 3.0, plug-ins can contribute their own default profiles (through extension points, for convenience) which allows users to pick up their auto-tactic from a portfolio, rather than having to define a new one by themselves. This therefore improves the usability of the auto-prover and allows to capitalize easily on best practices.

SMT Solvers

At the beginning of the ADVANCE project, an initial version of the *SMT solvers* plug-in was available. However, this early version was still very fragile and most of the times was producing proofs that could not be saved, rendering it useless. During the first and second period of the project, as a background task, this plug-in has been strengthened and brought to industrial grade. Version 1.0 of the plug-in has been released in June 2013.

ProB as a Disprover / Prover We reactivated a plug-in for Rodin, that uses the ProB animator and model-checker to generate counterexamples for proof obligations. Searching for a counterexample can be done effortlessly and without user interaction. If successful, the futility of a manual proof attempt is detected early. Furthermore, the counterexample itself provides an insight into the problem and aids in debugging the model.

Additionally, we investigated the use of ProB as a prover. By observing the number and quality of occurring enumerations, ProB is able to tell if the search for a counterexample was done exhaustively. If this is the case, the proof obligation in question is discharged. An initial empirical evaluation was performed. For certain proof obligations containing variables with finite data types, ProB is able to find proofs that are not found by the integrated provers or the SMT solvers.

Other external provers We initially planned to connect the *Super Zenon*[1] and the *iProver*[2] external provers to the Rodin platform. The development of *iProver* seems to have gone closed source : there

1 `http://cedric.cnam.fr/~delahaye/super-zenon/`
2 `http://www.cs.man.ac.uk/~korovink/iprover/`

is no development anymore on the Google project, although a new version 1.0 has been presented to the latest CASC-24 competition. Consequently, we could not make any progress with this prover. *Super Zenon* sources have not yet been made publicly available, but it is planned to be released under a GPL license by the end of 2013. We will then work on its integration into Rodin.

3.3 Available Documentation

Tactic profiles are described in the Rodin Handbook[3].

The user manual of the *SMT Solvers* plug-in is available on the Event-B wiki[4].

The introduction of variations in the HYP, HYP_OR, CNTR and GenMP reasoners is described in the Event-B wiki [5].

The design of the *Membership Goal* reasoner[6] has been updated to reflect the use of a SAT solver.

3.4 Planning

During the last period of the ADVANCE project, we plan to continue to improve the rate of automated proofs based on feedback from the case studies.

As concerns the SMT solver integration plug-in, we will benchmark it on the case study models and investigate the integration of new solvers such as CVC4.

Finally, as concerns external provers, we plan to work on *Super Zenon* integration once it is made open source. We will also check regularly for public releases of new provers that could be integrated.

We will evaluate the ProB Disprover in particular with respect to correctness. We will also prepare a set of automatic tests to ensure that we don't regress.

3 http://handbook.event-b.org/current/html/preferences.html#ref_01_preferences_auto_post_tactic
4 http://wiki.event-b.org/index.php/SMT_Solvers_Plug-in
5 http://wiki.event-b.org/index.php/Variations_in_HYP,_CNTR_and_GenMP
6 http://wiki.event-b.org/index.php/Membership_in_Goal

4 Model Checking

4.1 Overview

Experience with industrial case studies, such as those of WP1 and WP2, have demonstrated that animation and model checking are important tools when building a model. Animation allows the user to validate if the model corresponds to the user's intentions. Model checking allows to check if the model contains errors and provides counter-examples that help to understand the problem beforehand. Moreover, it allows to reason with domains (like physical units) and verify some properties (like temporal logic ones), that have currently no matching proof support. The following activities were pursued within the project:

- The constraint solving capabilities of ProB have been continuously improved along with scalability improvements.
- Previously we implemented a TLA to B compiler to use ProB on TLA$^+$ specifications. We now also support the direction from B to TLA$^+$. This allows to use the TCL model checker on B specifications.
- There is work in progress towards support of the upcoming 2.0 version of the Theory plug-in.
- We are working on a secondary independent toolchain for animation.
- We finalized and released the of physical units plug-in.

Outside ADVANCE we improved the model checker by adding support for distributed model checking, partial order reduction and partial guard evaluation. We think that these improvements are also valuable within the ADVANCE project.

4.2 Motivations / Decisions

Secondary toolchain

Independently of ProB we completed the development of a tool, which is able read solutions (constant and variable-values) generated by ProB and evaluate predicates like the properties or the invariant of a B model. Also it is possible to evaluate expression in an interactive mode or animate B machines. The tool is (with the exception of the parser) a complete new implementation of a predicate/expression B-evaluator and animator in python. The goal of the reimplementation is to establish a second independent toolchain to crosscheck results of ProB. The python tool has been

successfully tested with industrial B Models from the railway domain. The tool is available from http://github.com/hhu-stups/pyB[1]

B to TLA$^+$

We developed a transcompiler from B to TLA$^+$ (Temporal Logic of Actions) to verify B specifications with TLC. TLC is an explicit state model checker for TLA$^+$ with an efficient disk-based algorithm and support for temporal formulas and fairness.Our translator is full automatic and supports a large subset of B. Moreover it uses static analyses to verify the validity of the B specification (e.g. type checking and scope checking) and creates an optimized translation for the validation with TLC. Error traces (e.g. leading to deadlocks or invariant violations) detected by TLC are translated back to B and can be verified by ProB.

Kodkod

Kodkod is a Java library that provides an interface to SAT solvers for solving relational problems. In [2] we described a translation from B/Event-B to Kodkod where we applied the translation to improve finding values for the constants of a model and ProB's constraint based deadlock check. We have now applied the technique to check theories of a context in Event-B, check assertions in classical B and to the constraint-based invariant check. It became apparent that reducing the model to a relevant set of variables and predicates can improve the applicability of the method because often only a part of a model can be translated to Kodkod. We have implemented such a *slicing* of the model and plan to validate if this leads to a performance improvement when we use only ProB's constraint solver without Kodkod.

Theory Plug-in support

We adapted ProB to the changes that had been made to the Theory Plug-in in preparation of the upcoming release. ProB had support for recursive data types and operators that were defined by direct definitions. Also the SUM and PRODUCT operators were explicitly supported. Additionally we implemented now support for the theory of transitive closures. From the current set of standard theories, ProB supports animation of models that use operators of the theories "BoolOps", "FixPoint", "Seq", "SUMandPRODUCT" and "closure". Still unsupported are the operators of the remaining standard theories "BinaryTree" and "List" which use recursively defined operators. We plan to implement support for them for the next release.

LTL Fairness

We currently explore ways to implement Fairness into ProB's LTL model checker. However, it is already possible to use fairness by using the B to TLA translator and running TLC.

Physical Units

The work on physical unit support has been completed and is available as an optional plug-in available from the ProB update site.

---

1 https://github.com/hhu-stups/pyB

2 http://www.stups.uni-duesseldorf.de/w/Special:Publication/PlaggeLeuschel_Kodkod2012

4.3 Available Documentation

ProB

The ProB Website[3] is the place where we collect information on the ProB toolset. There are several tutorials on ProB available in the User manual section. We also supply documentation on extending ProB for developers.

In addition we run a bug tracking system[4] to document known bugs, workarounds and feature requests.

BMotion Studio

A developer-, user documentation, tutorial and examples are available from a dedicated webpage[5] hosted by the University of Duesseldorf.

4.4 Planning

- We will continue to work on the translation of B to TLA+ in order to use the TLC model checker. The translation needs to be validated and more of the full classical B language needs to be covered. We will also strive to make TLC available within Rodin for Event-B specifications.
- We will also continue to try to add fairness constraints efficiently both to ProB's own model checker and via the translation to TLC.
- We will continue our work on porting BMotionStudio to HTML. We want to define an open architecture that allows re-using BMotionStudio outside of ProB.
- We will further improve support for theories with recursive functions and try to make recursive functions more efficient.
- We will continue evaluating and improving our SAT backend via Kodkod and also evaluate the possibility of using SMT backends.

3 http://www.stups.uni-duesseldorf.de/ProB
4 http://jira.cobra.cs.uni-duesseldorf.de/
5 http://www.stups.uni-duesseldorf.de/bmotionstudio

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 new 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 12 months, the capability of defining axiomatic definitions has been provided. Axiomatic definitions are defined by supplying the types, a set of operators, and a set of axioms. This feature has allowed us to create a few significant theories (language extensions), including the generalized sum/product and real numbers. Also it aids to model the hybrid systems. These extensions are being evaluated in the WP1 and WP2 case studies.

In the past 5 months, the procedure of accessibility scope definition is enhanced as below:

- A theory can access other global theories directly.

- A theory (wither local/global) will be introduced to a context/machine scope by means of a "theorypath" file.

Also the below additions are added to the plug-in:

- Conflict checking during process of importing a theory or introducing a theory.
- Several bugs are fixed, including the colour coding.

5.3 Available Documentation

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

- Proposals for Mathematical Extensions for Event-B.[1]
- Mathematical Extension in Event-B through the Rodin Theory Component.[2]
- Generic Parser's Design Alternatives.[3]

Technical details (specifications):

- Mathematical_Extensions wiki page.[4]
- Constrained Dynamic Lexer wiki page.[5]
- Constrained Dynamic Parser wiki page.[6]
- Theory plug-in wiki page.[7]

User's guides:

- Theory Plug-in User Manual.[8]

5.4 Planning

The implementation of the Theory plug-in is now reaching a stable state. A lot of enhancement were introduced during June-August 2013, which makes the Theory plug-in ready to be integrated as a part of the core distribution of Rodin v2.8. Also the following works are planned for the third period:

- Enhancement of the deploy process: providing a read-only view of a deployed theory
- Improvements of the plugin structure in order to be supported by ProB

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

- Optimization of the HTML editor of a theory
  - colour coding to be compatible with the Event-B editor
  - enhancing the view of the conditional rewrite rules
- Providing a facility to automatic generation of rewrite/inference rules (from axiom/theorem)

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

Composition

While applying composition, properties must be maintained and proofs obligations need to be discharged in order for the final result to be considered valid. Since the composition maintains the monotonicity property of the systems, the sub-systems can be refined independently on a further stage, preserving composition properties.

In the latest version released (v1.6.1), some bugs are fixed to make it compatible with Rodin v2.8.

Decomposition

The main goal of Decomposition is to "cut" a model M into sub-models $M_1, ..., M_n$, which can be refined separately and more comfortably than the whole. The constraint that shall be satisfied by the decomposition is that these refined models might be recomposed into a whole model MR in a way that guarantees that MR refines M. Note that this particular recomposition will never be performed in practice.

In an shared variable decomposition, the events of a model are partitioned to form the events of the sub-models. In parallel, the variables on which these events act are distributed among the sub-models.

In an shared event decomposition, the variables of a model are partitioned among the sub-models. Consequently, the events are partitioned accordingly to each sub-model.

In the latest version released (v1.2.6) compatible with Rodin v2.8, some bugs are fixed.

Composition and decomposition have already been identified as important features for the WP1 and WP2 case studies and they will continue to be evaluated by these WPs.

6.3 Available Documentation

Documentation for composition is available in:

- A user guide and release history on the Event-B wiki[1].
- A paper: *Towards the Composition of Specifications in Event-B*[2].

Documentation for decomposition is available in:

- The decomposition user guide[3].
- The decomposition description[4].
- A paper on the decomposition Tool for Event-B[5].
- A Survey on Event-B Decomposition[6].

1 http://wiki.event-b.org/index.php/Parallel_Composition_using_Event-B
2 http://eprints.soton.ac.uk/272177/
3 http://wiki.event-b.org/index.php/Decomposition_Plug-in_User_Guide
4 http://wiki.event-b.org/index.php/Event_Model_Decomposition
5 http://eprints.soton.ac.uk/271714/
6 http://e-collection.library.ethz.ch/eserv/eth:5537/eth-5537-01.pdf

6.4 Planning

Composition

Support for animation directly over a composed machine is planned. This will allow multiple machine to interact with each other. Improvements in usability of the tool will be made.

Decomposition

Although the actual decomposition operation is mature enough, the feeling is that the steps leading to that stage require some additional support. The preparation steps preceding the decomposition usually require some adjustments to the model in order to the decomposition to be beneficial in the following refinements of each generated sub-component. Therefore we aim to:

- Receive more feedback from the users to improve tool usability.
- Auto fixing of problems before executing decomposition: building a set of problem templates and respective solutions to be suggested to the modeller before executing decomposition.
- Build a list of architectures templates to be used for decomposition according to the number of sub-components to be generated/kind of architecture to be followed/etc.

7 Revised Roadmap

This section describes the needs of the industrial case studies (WP1 and WP2) which justify the further described revision of the tooling roadmap.

During the ADVANCE plenary meeting which was held in Winchester November 7th and 8th, the industrial partners which are involved in work packages 1 and 2 expressed their tooling needs to support the case studies. Consequently, ADVANCE partners then agreed to revise together the roadmap which appears in the deliverable D3.1 by giving priority to the tasks addressing these needs by the end of the project.

7.1 Common needs across both work packages

For both WP1 and WP2 case studies, the need of a better support of theories, better automatic proving capability and customised model animation was expressed. As concerns the support of theories, two difficulties are met. On the first hand, the rapid evolution of the theory plug-in led to major refactorings of its code which affected its stability. Consolidation is needed and will be performed during the last period of the project to ensure the seamless integration of the plug-in within the Rodin platform. On the other hand, partners are in need of a better support for proving with theories. The available tactics provided are not sufficient, and the integration with external provers is not yet satisfactory. Focus will be given to enhancements on these aspects which will accompany the consolidation work.

While the complexity of case study models increased concurrently to their size, the ratio of automatically discharged proof obligations decreased. Often, the application of small manual proof steps suffices to let the existing proving tools discharge the proof obligations. This situation although already encountered in previous projects involving the Rodin toolset shows that the addition of new tactics, and enhancement of the existing proof tooling is a continuous duty which has to be carried on until the end of the ADVANCE project. Note that this issue will also be partially addressed by the planned work of integrating the Super Zenon theorem prover.

Finally, customised animation of high-level models appeared to be a major concern to industrial partners. Indeed, it is a priority task to get a domain-specific point of view on high-level Event-B models and be able to validate their behaviour using the animation. Some enhancements directed by feedback from case study work will take place by the end of the project.

7.2 Work package specific needs

Partners of WP1 expressed specific tooling needs on the following aspects : decomposition, testing, visualisation, and in linking the STPA method with Event-B and system developement lifecycle. WP2 partners expressed tooling needs according to the evolution of their work. In a first time, they are in need of support for composition. In a second time, they will need support for multi-simulation, test-case generation, and code generation.

WP1 case study models represent a large-scale system involving heterogeneous actors and which interfaces with a rich environment. When refining the abstract model, it appears that several entities can be decomposed. A seamless support of modifications within decomposed modelling artifacts is needed. Some work guided by WP1 feedback on decomposition limitations will be carried on.

The WP2 case study tackles a massively distributed system where several identical entities can be recomposed to represent the global system. Help and improvements on composition and decomposition will then be provided to WP1 and WP2 according to their feedback on existing tooling limitations.

The other tasks mentioned above are related to work package 4, but may need some evolutions from the modelling and proof tooling support. Priority will be given to such needs and enhancements of the Rodin platform will be performed by WP3 partners.

7.3 Conclusion

As seen in this section, priority will be given to the tasks needed to complete successfully the case studies. Thus, the other tasks which are planned and which appear in the roadmap of D3.1 will be performed in consideration of the remaining resources available once the above prioritized tasks have been completed.