# D3.2 Methods and Tools for Model Construction & Proof I

ADVANCE

| | |
|---|---|
| ***Grant Agreement:*** | *287563* |
| ***Date:*** | *2012-07-25* |
| ***Pages:*** | *1* |
| ***Status:*** | *Formal Release* |
| ***Access:*** | *Access List* |
| ***Issue:*** | *1.0* |

***Partners / Clients:***

| | |
|---|---|
| ***FP7 Framework Programme*** | ***European Union*** |

***Consortium Members:***

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

# Contributors

Laurent Voisin (Systerel),
Thomas Muller (Systerel),
Colin Snook (Southampton University),
Andy Edmunds (Southampton University),
Vitaly Savicks (Southampton University),
Lukas Ladenberger (Düsseldorf University),
Ingo Weigelt (Düsseldorf University),
Michael Jastram (Düsseldorf University),
Issam Maamria (Southampton University),
Michael Leuschel (Düsseldorf University),
Daniel Plagge (Düsseldorf University),
Jens Bendisposto (Düsseldorf University),
Michael Butler (Southampton University),
Renato Silva (Southampton University).

# Contents

# 1 Introduction

The ADVANCE D3.2 deliverable is composed of the present document and the Rodin toolset. The considered Rodin toolset consists of the Rodin core platform and the plug-ins created or maintained in the frame of the ADVANCE project. 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 first ten months of the ADVANCE project (Oct 2011 - Jul 2011), and aims to let the reader get a glimpse of the WP3 member's contribution plans and objectives. It is worth reminding that the ADVANCE project time frame overlapped the end of the EU FP7 DEPLOY project[4] from October 2011 to April 2012. Therefore, the commitment of WP3 members that were members of the DEPLOY tooling workpackage as well, was mainly devoted during this period to consolidate the toolset by fixing the identified bugs and mitigating usability issues. These activities encompass the objectives of the task 3.2 mentionned in the ADVANCE Description of Work (DoW).

The document is divided into four parts: general platform maintenance, improvement of automated proof, language extension, model checking, and 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 related.
- Available documentation. Some pointers to the available documentation or related publications are listed.
- Planning. The current status about the topic (as of July 2012), and an overview of the future plans are given.

## References

[1] http://sourceforge.net/projects/rodin-b-sharp/files/Core_Rodin_Platform
[2] http://wiki.event-b.org
[3] http://handbook.event-b.org
[4] http://www.deploy-project.eu

# 2 General Platform Maintenance

This part concerns the general maintenance performed on the Rodin toolset within the first ten months 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. These sections are: core Rodin platform, UML-B improvements, code generation, ProR and Camille. All these subsections maintain the template previously defined in the introduction.

## 2.1. Core Rodin platform

### 2.1.1. Overview

The Rodin platform versions concerned by this deliverable are:

- 2.4 (released on 31.01.2012),
- 2.5 (released on 02.05.2012),
- 2.6 (released on 31.07.2012).

The core Rodin platform maintenance task focused on fixing the identified bugs and mitigating usability issues. During DEPLOY, many features and contributions were added to the toolset as users wishes and requests were collected along. At the same time, the Event-B models and proof got bigger and bigger, in the same way as the experience of the users involved constantly increased ergo the size of the systems they modelled. Scalability issues occured at some point when feature addition was favoured to design refactorings. As the DEPLOY project was nearing its end, it appeared mandatory for the development team, to address the specific bugs and issues reported by the DEPLOY partners and related to scalability or usability, and wished resolved by the end of the project. The various tasks to be performed were scheduled, prioritized and regularly updated during bi-weekly teleconferences.

The following paragraphs will give an overview of the the work that has been performed concerning maintenance on the existing platform components (i.e. core platform and plug-ins). Release Notes[1] and the SourceForge trackers[2] (bugs and feature requests) are available for more details about the previous and upcoming releases of the Rodin platform.

### 2.1.2. Motivations / Decisions

**Provide 64-bit versions of the Rodin platform on Linux and Windows systems**
End users asked the Rodin team to provide 32-bit and 64-bit versions of the Rodin platform for Linux and Windows operating systems. Before Rodin 2.4, the only 64-bit version of Rodin was available on Mac platforms as 32-bit Mac (early 2006) platforms are no longer maintained by Apple. The motivation that would push forward 64-bit architectures is the possibility to increase the available java heap size which is, for example, extensively used during the automated proof. After a phase of testing and despite the drawbacks of assembling and maintaining five platforms instead of three, Linux and Windows 64-bit as well as 32-bit platforms are now systematically made available since the Rodin 2.4 release.

**The Rodin Editor in the Rodin core platform**
According to its role in the Rodin toolset, and its stabilization, the Rodin Editor,[3] has been integrated into the core platform since Rodin release 2.4.

### 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 Rodin platforms. Moreover, two web trackers list and detail the known bugs and open feature requests:

- a sourceforge bug tracker,[4]
- a sourceforge feature requests tracker.[5]

The Event-B wiki,[6] basic source of documentation for the users and developers of the Rodin toolset, was completed by the Rodin handbook, an ultimate source of documentation which reached completion by the end of the DEPLOY project. The handbook aimed to overcome the lack of a centralized 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 as a PDF version, a HTML version, and help contents within Rodin. 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, in order to provide video tutorials about the use of the platform.[7]

### 2.1.4. Planning

Two releases of the core platform have been scheduled after Rodin 2.6 has been published: Rodin 2.7 (end of October 2012) followed by Rodin 3.0 (end of January 2013). Rodin 2.7 will mainly be a corrective release. However, it will integrate the Theory plug-in (which allows to define mathematical extensions and is currently provided as a separate plug-in).

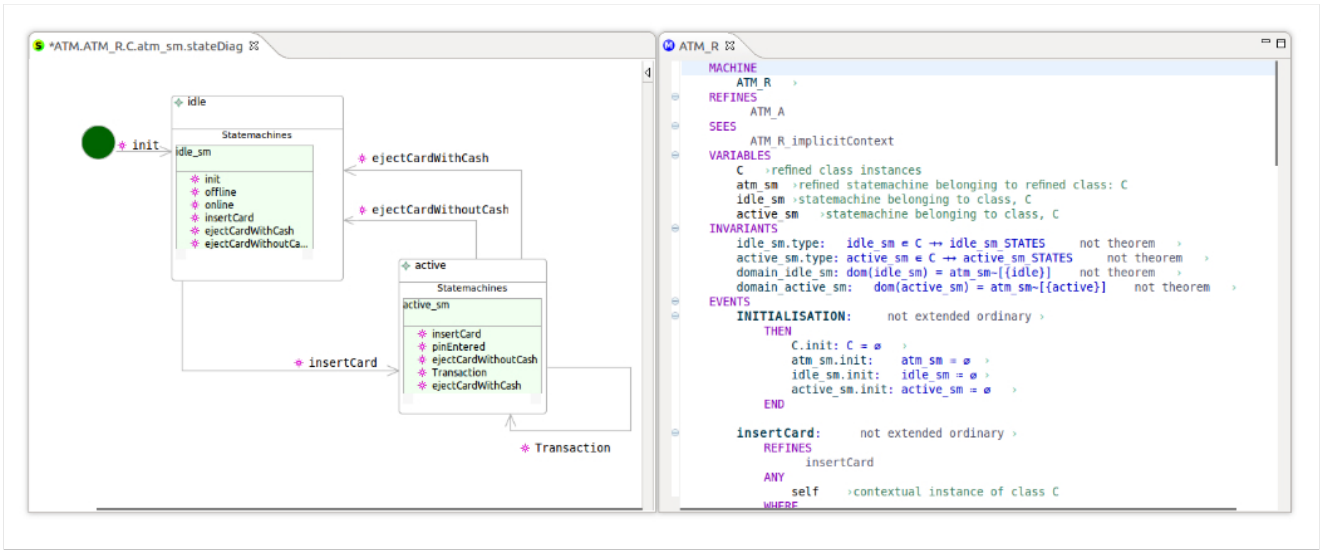
Work on release 3.0 will start concurrently with development of Rodin 2.7 and will include lots of refactoring and evolutions in the published programming interface (API). A non exhaustive list of the planned improvements and refactoring is:

- Binders will be allowed in extensions.
- The platform will be based on Eclipse 4.
- The AST will be made stronger.
- The sequent prover will be enhanced.
- Parent-child element relationship extension points will be moved from the UI plug-in to the EventB core plug-in.
- The Event-B keyboard plug-in will be refactored to separate the UI code from the ASCII/Math translation mechanism.
- The statistics view will be refactored to handle other kinds of component files (currently just Contexts and Machines are supported).

## 2.2. UML-B Improvements

### 2.2.1. Overview

The UML-B plug-in and associated frameworks are developed and maintained by Dr Colin Snook at the University of Southampton. Significant contributions (to the latest version) have been made by Vitaly Savicks (state-machines) and Gintautas Sulskus (Class Diagrams). The UML-B plugin provides UML-like diagrammatic modelling as an extension to Event-B and the Rodin platform. UML-B is an established plug-in which will be developed and improved to support the aims of the ADVANCE project. UML-B was redeveloped during the DEPLOY project to provide closer integration with Event-B. A state-machine diagram editor is already released in this integrated version and a class diagram editor is now being developed as a prototype.



```
MACHINE
    ATM_R  ›
REFINES
      ATM_A
SEES
      ATM_R_implicitContext
VARIABLES
    C   ›refined class instances
    atm_sm  ›refined statemachine belonging to refined class: C
    idle_sm ›statemachine belonging to class, C
    active_sm   ›statemachine belonging to class, C
INVARIANTS
    idle_sm.type:   idle_sm ∈ C ⇸ idle_sm_STATES     not theorem  ›
    active_sm.type: active_sm ∈ C ⇸ active_sm_STATES     not theorem  ›
    domain_idle_sm: dom(idle_sm) = atm_sm∼[{idle}]     not theorem  ›
    domain_active_sm:   dom(active_sm) = atm_sm∼[{active}]     not theorem  ›
EVENTS
    INITIALISATION:     not extended ordinary ›
        THEN
            C.init: C ≔ ∅   ›
            atm_sm.init:    atm_sm ≔ ∅ ›
            idle_sm.init:   idle_sm ≔ ∅ ›
            active_sm.init: active_sm ≔ ∅  ›
        END

    insertCard:     not extended ordinary ›
        REFINES
              insertCard
        ANY
            self    ›contextual instance of class C
        WHERE
```

Other improvements will include new diagrammatic notations which are directly related to the aims of ADVANCE, such as component diagrams, as well as more general improvements, such as usability, and any features required by the projects industrial partners.

### 2.2.2. Motivations / Decisions

The implementation of the UML-B tool is structured to provide re-usable features as much as possible. To achieve this, wherever possible, generic frameworks are developed and the implementation of specific diagram tools is minimised by utilising these frameworks. The following frameworks are now in use by UML-B and available as a basis for future diagrammatic modelling notations.

- The *Event-B EMF framework* provides an EMF basis for Event-B models (developed during the DEPLOY project). Indeed, it provides an EMF representation of Event-B models with persistence into the Rodin database.
- The *Event-B EMF Support for Modelling Extensions framework* provides support for extending Event-B with new modelling features (initially developed during the DEPLOY project and now being extended in the ADVANCE project). It provides Navigator support for EMF-only model elements, a persistence mechanism for model extensions that are not needed to be processed by Rodin and a Generic Refiner for modelling extensions (which has recently been added).
- *The Event-B GMF Diagrams Generic Support framework* provides support for developing new diagram notations (started in the DEPLOY project but mostly developed during the ADVANCE project). It provides generic support for diagrammatic aspects of modelling extensions, and a generic validation and Event-B generation service (which has recently been added).

A new release of the state-machine diagram editor has been made. This release corrected some problems and improved use of the generic framework features.

Work is in progress on a new version of the UML-B Class Diagram editor. This takes the same approach as the state-machines editor in that the models are contained within machines (and, in this case, also contexts) and that diagrammatic model features link to and enhance existing Event-B elements rather than generate everything. The Class diagram editor is currently a prototype and has not been released.

### 2.2.3. Available Documentation

A paper exists about the framework for diagrammatic modelling extension in Rodin: Savicks, Vitaly, Snook, Colin (2012) *A Framework for Diagrammatic Modelling Extensions in Rodin* in Rodin User & Developer Workshop 2012 Proceedings. Newcastle University. (Unpublished) [8]

Wiki pages are available for developers using the Generic extensions and Diagrams frameworks:
http://wiki.event-b.org/index.php/EMF_framework_for_Event-B
http://wiki.event-b.org/index.php/Generic_Event-B_EMF_extensions (Under Construction)

A paper exists about the Event-B state-machines: Savicks, Vitaly, Snook, Colin and Butler, Michael (2009) *Animation of UML-B State-machines* in Rodin User & Developer Workshop 2010 Proceedings. Düsseldorf University, 2010.[9]

Wiki pages are available for users of UML-B, everything starts from http://wiki.event-b.org/index.php/UML-B[10]

### 2.2.4. Planning

The following work is planned:

- Re-base the Event-B EMF framework on the EMF model of the Rodin database (which is now available as a result of the development of a new Rodin editor to replace the form based editor).
- Re-write the Event-B generator of the state-machine diagram editor so that it uses the generic generator from the Event-B GMF Support for Generic Diagrams framework. This will improve performance.
- Improve the state-machine diagram animation interface so that it supports animation of multiple diagrams.
- Complete the development of the new version of the UML-B Class Diagram editor.
- Develop Animation interface for the Class diagram editor.
- Develop a Component diagram editor and associated simulation tools.

## 2.3. Code generation

### 2.3.1. Overview

The code generation feature provides the support for the generation of code from refined Event-B models. The development of the approach, and the tools to support it, involved a number of team members at Southampton. To this end a multi-tasking approach, which is conceptually similar to that of the Ada tasking model, was designed. Tasks are modelled by an extension to Event-B, called Tasking Machines which are an extension of the existing Event-B Machine component. The latest Code Generation feature was released on 30th May 2012. The new features in this release include code generation from state-machine diagrams (see UML-B).

### 2.3.2. Motivations / Decisions

State-machines are frequently used to describe the behaviour of embedded systems. It is a relatively new feature in Event-B, and we augment the tool with the ability to generate code from state-machine diagrams in version 0.2.3 of the code generation feature plug-in. Implementation code is generated from the diagram itself, and no additional mark-up of the model is required; that is, nothing over and above the usual mark-up required for Tasking Event-B, such as identifying non-typing/typing invariants, and guards etc. State-machines are created, using the existing state-machine plug-in, subject to the limitations described after.

The current code generation tool is restricted to generating code for a single Event-B machine, which may contain one or more state-machines. We have yet to explore the decomposition/composition of machines containing state-machines. In principal we should be able to apply decomposition techniques to decompose the single Event-B machine with state-machines into a number of machines, with the state-machines, or the elements of state-machines, distributed between them. Another limitation is that we do not handle nested state-machines, although this should be feasible. Only the state-machines of tasking/environ machines generate code. State-machines of shared machines do not generate code. This should be explored further during research into decomposition.

The translation of the diagrammatic elements to code has been hard-coded in the code generation plug-in. We have introduced new types to the translator's common language model (IL1). We add case-statements, and a container for them (analogous to switch) since these are commonly used to implement state-machines. The code generator navigates through each state of a state-machine, generating an internal representation of the state-machine, which is used to create the IL1 model. The IL1 model is then used to generate code for the various target languages that may have been implemented. We have also updated the IL1-to-target code generators, to generate case/switch statements in Ada, C and Java. Each state-machine has an Enumerated type whose elements take the names of the states. A state variable is created in the target that keeps track of the current state, and has the type of the enumeration.

### 2.3.3. Available Documentation

A specific page on the Event-B wiki[11] is dedicated to Code Generation Updates.

### 2.3.4. Planning

Efforts will focus on extending the existing code generation approach, and developing new techniques, for the simulation of cyber-physical systems.

## 2.4. ProR

### 2.4.1. Overview

The Rodin/ProR 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).[12]

The following paragraphs will give an overview of the the work that has been performed concerning maintenance on the Rodin/ProR plugin.

### 2.4.2. Motivations / Decisions

The motivation of the Rodin/ProR integration plugin was to bring two complimentary fields of research, requirements engineering and formal modelling, closer together. Especially, the traceability within a system description is a challenging problem of requirements engineering. In particular, formal models of the system are often based on informal requirements, but creating and maintaining the traceability between the two can be challenging. In *A Method and Tool for Tracing Requirements into Specifications*[13], we presented an incremental approach for producing a system description from an initial set of requirements. The foundation of the approach is a classification of requirements into artefacts W (domain properties), R (requirements) and S (specification). In addition, the approach uses designated phenomena as the vocabulary employed by the artefacts. The central idea is that adequacy of the system description must be justified, meaning that $W \wedge S \Rightarrow R$. The approach establishes a traceability, and the resulting system description may consist of formal and informal artefacts. We created tool support for this approach by integrating Rodin and ProR. We designed it with the goal to support the approach described in [13], and to ease the integration of natural language requirements and Event-B.

### 2.4.3. Available Documentation

- *A Method and Tool for Tracing Requirements into Specifications*.[13] The paper has been submitted to Science of Computer Programming.
- *Requirements Traceability between Textual Requirements and Formal Models Using ProR*.[14] The paper has been accepted for iFM'2012 & ABZ'2012.
- A Tutorial for the Rodin/ProR integration[15] can be found on the Event-B wiki.
- The User Guide[16] is available on the Event-B wiki and contains an additional tutorials for ProR.

### 2.4.4. Planning

There are still some limitations on the ProR/Rodin integration plugin, however. While all required data structures exist, the plugin would benefit from more sophisticated reporting. In particular, the method mentioned above[13] 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 design decisions and plaform 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. This is already done, to a degree, through the colour highlighting, but a reporting tool that lists all violations of those properties will be implemented.

Currently, each used artifact is identified as belonging to one of the following class: domain properties, requirement items, specification elements, implementation elements, design decisions or platform properties. This in turn determines the different kinds of phenomena that are allowed to be used by the artefact. For the moment, the plugin does not support such classification by phenomenon types. In a next step, we will work on a concept for classifying and maintaining phenomena with ProR.

## 2.5. Camille

### 2.5.1. Overview

The Camille plug-in provides a textual editor for Rodin. Though such a text editor is prefered by many users, Camille currently has the drawback of not supporting extensibility. It only supports the core Event-B language and plug-in-specific additions are simply ignored. Such extensions can not be edited through Camille. Consequently, users have to switch back to Rodin's native Editor to edit plug-in-specific modelling extensions. This has become a major issue during the last years, since many plug-ins have been developed in the meantime that are widely used by many users.

This issue is currently being adressed by Ingo Weigelt at the University of Duesseldorf.

### 2.5.2. Motivations / Decisions

A new version of Camille will be implemented during the ADVANCE project to enable users to edit extended Event-B models solely through a text editor. To plan the new version, the problem and a number of possible solutions have been analysed and related in a technical report[17] from Ingo Weigelt. The results of this work have been dicussed in the Rodin community and one of the proposed solution (a blockparser) has been selected to be implemented during the next months. The dedicated solution promises to provide Camille extensibility with minimal extra workload required from plug-in developers while still being very flexible regarding future, yet unknown, requirements.

### 2.5.3. Available Documentation

- *Architectures for an Extensible Text Editor for Rodin*.[17] Bachelor thesis analysing the problem and discussing possible solutions.
- An earlier version of the thesis has been published as a technical report[18] that has been discussed on the Roding Developers Mailing List and the ADVANCE Progress Meeting in May 2012 in Paris.

### 2.5.4. Planning

The new Camille version will be implemented at the University of Duesseldorf during 2012. The final version is expected in January or February 2013.

## References

[1] http://wiki.event-b.org/index.php/Rodin_Platform_Releases
[2] http://sourceforge.net/projects/rodin-b-sharp/
[3] http://wiki.event-b.org/index.php/Rodin_Editor
[4] http://sourceforge.net/tracker/?group_id=108850&atid=651669
[5] http://sourceforge.net/tracker/?group_id=108850&atid=651672
[6] http://wiki.event-b.org/
[7] http://www.youtube.com/user/EventBTv
[8] http://deploy-eprints.ecs.soton.ac.uk/382/
[9] http://eprints.soton.ac.uk/268261/
[10] http://wiki.event-b.org/index.php/UML-B
[11] http://wiki.event-b.org/index.php/Code_Generation_Activity

[12] http://www.eclipse.org/rmf/
[13] http://www.stups.uni-duesseldorf.de/w/Special:Publication/HalJasLad2012
[14] http://www.stups.uni-duesseldorf.de/w/Special:Publication/LadenbergerJastram_iFMABZ2012
[15] http://wiki.event-b.org/index.php/ProR
[16] http://wiki.eclipse.org/RMF/User_Guide
[17] http://www.stups.uni-duesseldorf.de/mediawiki/images/0/0a/Pub-Weigelt2012.pdf
[18] http://www.stups.uni-duesseldorf.de/w/Special:Publication/Weigelt2012>

# 3 Improvement of automated proof

## 3.1. Overview

In an Event-B development, more than 60% of the time is spent on proofs. That explains why all users are naturally keen for the proofs to be as automatic as possible and why the automated prover enhancement was a continuous task since the birth of the Rodin platform. Enhancing the automated prover can be achieved by core platform refactorings and additions to it, such as the addition of integrated reasoners and tactics, but also by the integration of some external reasoning ability such as external provers (e.g., SMT solvers).
From the core platform point of view, and within the ten first month of ADVANCE, it consisted into two tasks: the addition of rewriting and inference rules, and the addition of a mechanism to allow the customization and the parametrization or combination of tactics. The user is now able to define various types of tactics called 'profiles' which could be customized and parameterized tactics to discharge some specific proof obligations. The user can furthermore share and backup these defined tactics using the provided import/export mechanism.
From an external point of view, the SMT Solver Integration plug-in allowing to use the SMT solvers within Rodin as an effective alternative to the Atelier-B provers, particularly when reasoning on linear arithmetic. It is maintained in the time frame of ADVANCE, and increases the rate of automatically discharged proof obligations.

## 3.2. Motivations / Decisions

The proportion of automatically discharged proof obligations heavily depends on Auto-Tactic configuration. Sometimes, the automatic prover fails because the tactics are applied in an unappropriate order. Since Rodin 2.4, a new tactic combinator 'Attempt after Lasso' is available in the tactic profile editor as well as an import/export feature. Indeed, a user that elaborates a good profile for a certain proof pattern is now able to share or backup this profile thus increasing the number of automatic proofs for a given proof pattern.

Two main reasons mainly motivated the integration of SMT solvers into the Rodin platform. Firstly, to allow Rodin to benefit from the known capacity of such solvers in the field of arithmetics. Secondly, to extract some useful information from the proofs that these solvers produce such as unsatisfiable cores, in order to significantly decrease the proving time of a modified model. The translation of Event-B language into the SMT-LIB language is the main issue of this integration. Two approaches were developed for this. The more efficient one is based on the translation capabilities of the integrated predicate prover of the Rodin platform (PP). It is completed by translating membership using an uninterpreted predicate symbol, refined with an axiom of the set theory.

## 3.3. Available Documentation

A page[1] concerning tactic profiles is available in the user manual.
A page[2] is dedicated to the SMT Solver integration plug-in on the Event-B wiki.

## 3.4. Planning

Enhancement to automated proof will continue during the ADVANCE project. Notably, the remaining missing rewriting[3] and inference rules[4], that have already been documented, will be implemented together with new rules.

Maintenance of the SMT solver integration plug-in will be ensured within the time frame of ADVANCE. In particular, the translation to SMT-LIB will be completed by adding some support for mathematical extensions.

Finally, connecting other provers (such as Super Zenon and iProver) to the platform will be investigated.

## References

[1] http://handbook.event-b.org/current/html/preferences.html#ref_01_preferences_auto_post_tactic
[2] http://wiki.event-b.org/index.php/SMT_Solvers_Plug-in
[3] http://wiki.event-b.org/index.php/All_Rewrite_Rules
[4] http://wiki.event-b.org/index.php/Inference_Rules

# 4 Language extension

## 4.1. 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.

## 4.2. 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. Earlier versions of the Theory plug-in provided support for rewrite rules where soundness is preserved by means of proof obligations. 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 9 months, the development of the Theory plug-in resulted in the following additions:

- Support for polymorphic theorems in proofs: theorems can be used to ensure the operator and datatype definitions capture the intended understanding by the theory developer. However, requests have been made to ensure that theorems are accessible for proofs. It was decided to provide a user interface-based wizard for instantiating the type parameters in order to 'localise' the theorem based on the current sequent.
- Added support of axiomatic definitions: as of version 1.3.2 of the Theory plug-in, only direct and recursive definitions for operators are possible. Requests have been made to add support for axiomatic operator definitions, which can open the door for sophisticated type definitions, e.g., the definition of REALS as an ordered ring with addition and multiplication.
- From version 2.7 of Rodin, the Theory plug-in will ship as part of the core distribution. This particular decision was made given the maturity of the plug-in and its potential usefulness.

## 4.3. 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]

## 4.4. Planning

The implementation of the Theory plug-in is now reaching a stable state. A lot of enhancement were introduced in June and July 2012 so that an integration in Rodin 2.6 (which code freeze was set by mid July) was considered hasty and inappropriate. Thus, the Theory plug-in will become part of the core distribution of Rodin from version 2.7 which is planned for the end of October 2012.

## References

[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_UM.pdf

# 5 Model Checking

## 5.1. Overview

We think 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.

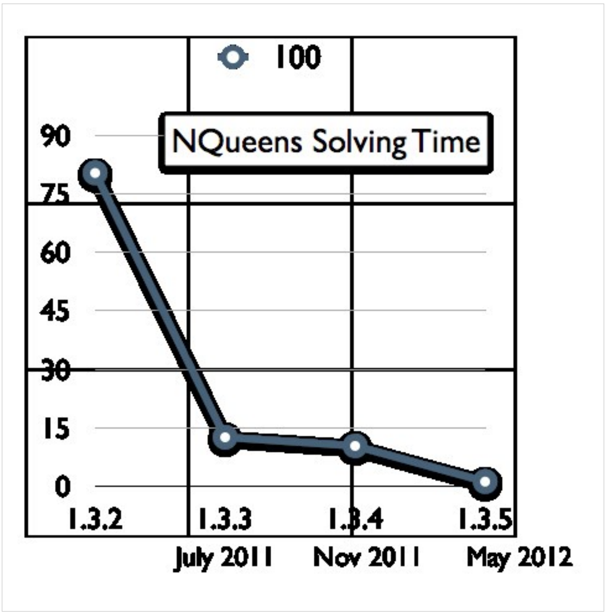
The following activities were pursued within the project:

- The constraint solving capabilities of ProB have been continuously improved along with scalability improvements.
- A conversion from TLA to B has been added. ProB now supports TLA+. The motivation is to extend the reach of the project and to learn from TLA concerning certain features relevant for cyber-physical systems (e.g. real number support).
- There is work in progress towards full support of Theory plug-in: support for external and recursive functions has been added.
- The conversion to the relational logic solver Kodkod has been completed and experiments with Kodkod and SMT translators have been conducted.
- We are working on an analysis of the use of physical units in a formal model.
- We improved the usability of the LTL model checker.
- Regarding BMotion Studio, we focused on fixing identified bugs and rectifying usability issues.

## 5.2. Motivations / Decisions

**Improvements to Constraint-Solving**

ProB's constraint solving capabilities are at the core of many of ProB's features: animation of high-level models with complicated predicates, model-based testing, constraint-based invariant and deadlock checking, etc. It is thus important to improve this aspect of ProB. In particular, we have continuously improved the performance of the kernel, as can be seen in the figure below showing the performance of ProB (in seconds) on the N-Queens problem for 100 queens. Other improvements lie in better expansion of universal and existential quantifiers, reification for the the `bool` operator and support for infinite and recursive functions. The latter is particularly important in light of the Theory plug-in work.



**TLA2B**

TLA+ and B share the common base of predicate logic, arithmetic and set theory. However, there are still considerable differences, such as very different approaches to typing and modularization. Some features of TLA+ are interesting in the context of cyber-physical systems, such as real numbers. There is also considerable difference in the available tool support. In particular, we wanted to compare ProB with TLC and gain insights about performance.

**Physical Units**

Formal models of cyber physical systems will contain variables which represent values with physical units. We are thus exploring to use the ProB model checker as a tool to infer and validate physical units usage in formal models. In particular, we want to make sure that the physical units in a model are used in a consistent way.

**Theory Plug-in and Mathematical Extensions**

In the ProB core, we have improved ProB to better deal with infinite and recursive functions. This can be used to provide formal specifications for mathematical extensions which can be animated and model checked by ProB. Using the newly developed external function mechanism, it should also be possible to support floats or reals, which will be important for certain cyber-physical systems.

On the technical side, we have extended the ProB internal representation of predicates and expressions to support the Theory plug-in. The implementation will be finalize as the Theory plug-in will allow access to definitions will be granted to ProB.

**Kodkod**

We have integrated a translation of B predicates to the relational logic solver "Kodkod" and evaluated how ProB's constraint solving compares to Kodkod's SAT solving. The integration allows to apply SAT solving to predicates where a translation is possible and a fallback to constraint solving for the remaining predicates. Our experiments have shown that the translation can be highly beneficial for certain kinds of constraints, and as such opens up new ways to analyze and validate formal specifications in Event-B. However, the experiments have also shown that the constraint logic programming approach of ProB can be superior in a considerable number of scenarios; the translation to Kodkod and down to SAT is not (yet) the panacea. The same can be said of the existing translations from B to SMT. As such, we believe that much more research required to reap the best of both worlds (SAT/SMT and constraint programming). A

side-effect of the translation to Kodkod is that the ProB toolset now provides a double-chain (relying on technology developed independently and using different programming languages and paradigms) of validation for first-order predicates, which should prove relevant in high safety integrity level contexts.
In comparision with other formalisms Kodkod has the advantage that is provides good support for relations and sets which play an essential role in Event-B's mathematical notation.

**LTL**
ProB supports LTL model checking. One problem when using LTL to validate a model is that counter-examples returned by the model checker are often hard to understand. A counter-example typically consists of a lasso-chaped sequence of states and events. Instead of just loading the sequence into the history of the animator, we have implemented a dedicated visualisation for LTL counter-examples. The visualisation which is now part of ProB's Rodin plug-in shows why an LTL operator is true or false in each state of the sequence.

**CSP and B**
ProB supports also other formalisms like CSP. CSP can also be used to guide B and Event-B models and can also be used for specifying scenarios or for model testing. Within the project this feature of ProB was continuously maintained and improved. We have extended the implementation of the CSP interpreter and animator to be able to support more complex and larger data types (e.g. mixing of dot and non-associative tuples) as well as supporting more complicated pattern matching inside set comprehension formulas and function definitions. Some effort for supporting more of CSP built-in functions (like seq(-), set(-) and card(-))was made as well. Finally, ProB now supports checking LTL-assertions directly from the CSP model by using pragmas ({-# assert_ltl = … #-}). The syntax is the same as for LTL-assertions in DEFINITION clauses of B models.

## 5.3. Available Documentation

**Constraint Solving**
The improvements are available in the nightly builds of ProB.
Two specific pages[1][2] have been added to the ProB user manual.

**TLA2B**
The TLA+ to B translation has been published at the iFM'2012 conference. A technical report is also available.[3] A presentation at the FM'2012 TLA+ workshop will also be made, and a dedicated page[4] has been added to the ProB user manual.

**Physical Units**
This work is still in progress. A first tutorial page[5] is available in the ProB online documentation. Full documentation will be made available later in the project. The latest nightly build of ProB contains an experimental version of the analysis.

**Kodkod**
A technical report[6] has been published on the validation using ProB and Kodkod. The paper has been accepted for FM'2012.

**LTL**
The concept and implemenation of the visualisation is described in the master thesis of Andriy Tolstoy.[7]

**BMotion Studio**
A developer-, user documentation, tutorial and examples are available from a dedicated

webpage[8] hosted by the University of Duesseldorf.

## 5.4. Planning

### Physical Units

Physical units work will be completed. First experiments with industrial models from Alstom are encouraging.

### Kodkod

Currently the translation to Kodkod is only applied to axioms when trying to find values for the constants and during the constraint based deadlock check. We plan to restructure ProB's internal programming interfaces in a way that allows to apply Kodkod more easily and make it available for other checks (e.g. constraint-based invariant check, assertion checks).
We will evaluate how we can employ more SMT based techniques in ProB.

### Constraint Solving

During the further development of ProB's constraint solving it became apparent that it would be helpful to represent the cardinality of a set by a CLP(FD) variable. We plan to change ProB's internal representation of sets in a way that its cardinality can be accessed in this way.
To allow a translation from ProB to Kodkod, we implemented an integer interval analysis. We plan to adapt the analysis to set up sizes of deferred sets. This is necessary because ProB chooses a fixed size for a deferred set and sometimes a model has only solutions for a certain size. Currently a user must supply a size manually.

### LTL

Fairness properties are very common when specifying LTL formula. Fairness can be encoded by using standard LTL, but it makes the formula significantly larger. The complexity of the model checking algorithm grows exponentially with the number of used LTL operators in a formula. We plan to incorporate support for fairness directly into the model checker which should lead to a drastic improvement in performance when fairness is used. Additionally, the usability of the model checker is improved by having the ability to specify fairness conditions seperatly from the rest of the LTL formula.

### BMotion Studio

We will provide a way to link up other Java-based simulation tools with BMotion Studio. Furthermore, beside working on identified bugs and and rectifying usability issues, we want to create more visual elements to aid humans understand large-scale simulations.

## References

[1] http://www.stups.uni-duesseldorf.de/ProB/index.php5/Recursively_Defined_Functions Recursive functions entry in ProB user manual

[2] http://www.stups.uni-duesseldorf.de/ProB/index.php5/External_Functions External functions entry in ProB user manual

[3] http://www.stups.uni-duesseldorf.de/w/Special:Publication/HansenLeuschelTLA2012 Translating TLA+ to B for Validation with ProB. Technical Report, 2012.

[4] http://www.stups.uni-duesseldorf.de/ProB/index.php5/TLA TLA2B entry in ProB user manual

[5] http://www.stups.uni-duesseldorf.de/ProB/index.php5/Tutorial_Unit_Plugin Unit Plug-in Tutorial entry in ProB user manual

[6] http://www.stups.uni-duesseldorf.de/w/Special:Publication/PlaggeLeuschel_Kodkod2012 Validating B,Z and TLA+ using ProB and Kodkod. Technical Report, 2012.

[7] http://www.stups.uni-duesseldorf.de/w/Visualisierung_von_LTL-Gegenbeispielen Andriy Tolstoy: Visualisierung von LTL-Gegenbeispielen, Master thesis, University of Düsseldorf, 2012
[8] http://www.stups.uni-duesseldorf.de/bmotionstudio

# 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 reusability 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 artefact 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 (v1.5), several features were added to the shared event composition tool:

- Proof obligation were added: Well-Definedness (WD) and Invariant preservation (INV) for (composition) invariants and applicable to all composed machines; Gluing Invariant Preservation (INV), Simulation (SIM) and Guard Strengthening (GRD) for refinement; other POs are expected to be proved in directly in the included machines.
- Stronger static checks were added: no shared variable is allowed; all abstract events must be refined; common parameters must have the same type; check that a composed event must have at least one combined event;
- The user interface was improved: using the form-based editor, if the symbol corresponding to a composed event is hovered, a preview of the composed event is displayed

**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 (v1.2.3), the user's inputs and request were taken into account. In particular in terms of better usability of the tool. It is worth citing the following modifications:

- Typing guards (automatically generated) are now marked as theorems.
- Static checks were added to the decomposition file according to the decomposition style chosen.
- Improved UI interface: dialog requesting confirmation to delete possible existing sub-components were removed (as generated sub-components are marked as read-only).
- The suffix of decomposition files was changed to "_DCMP".
- Small bugs were fixed.

## 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].

## 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.

## References

[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