Project DEPLOY
Grant Agreement 214158
"Industrial deployment of advanced system engineering methods for high productivity and dependability"

DEPLOY Deliverable D32

D9.3 Model Construction Tools and Analysis Tools III

Editor: *Thomas Muller (Systerel)*
Internal reviewers: *Felix Loesch (Bosch), Andreas Roth (SAP), Sebastian Wieczorek (SAP)*
Contributors: *Laurent Voisin (Systerel), Renato Silva (Southampton University) Nicolas Beauger (Systerel), Issam Maamria (Southampton University), Colin Snook (Southampton University), Matthias Schmaltz (ETH Zurich), Vitaly Savicks (Southampton University), Andy Edmunds (Southampton University), Alexei Iliasov (Newcastle University), Ilya Lopatkin (Newcastle University), Thai Son Hoang (ETH Zurich), Michael Leuschel (Düsseldorf University), Daniel Plagge (Düsseldorf University), Alin Stefanescu (Pitesti University)*

Public Document

28 January 2011

Contents

1 Introduction

The DEPLOY deliverable D32 is composed of:

- the Rodin core platform and plug-ins (i.e. the DEPLOY tools),
- this document.

The Rodin platform can be downloaded from the SourceForge site ([1]). The Event-B wiki ([2]) hosts the documentation of the tool.

This document gives an insight into the work achieved throughout the WP9 *Tooling research and development* work package, during the third year of the DEPLOY project (Feb 2010-Jan 2011), and depicts the WP9 partner's objectives for the coming and last year of the project.

Among the aims that WP9 partners reached in the past year, it is worth citing :

- Improved scalability and teamwork ability of the Rodin platform to support industrial deployments, through GUI refactoring and new features, Subversion model storage, decomposition, modularisation, flow support, as well as qualitative probabilistic reasoning and others,
- Mathematical extensions are now supported in Rodin. The core of the Rodin platform has been modified and the Theory plug-in has been developed to allow the definition of new basic predicates, new operators and new algebraic types,
- Prover performance has increased through the addition of a relevance filtering plug-in which raises the number of automatically discharged proof obligations. Moreover, work has been done to establish the soundness of provers and improve the generation of well-definedness proof obligations,
- Model animation has been improved: it now supports multi-level animation and has been applied in WP1-4 deployment workpackages,
- Model testing was guided by the needs of WP1-4 partners and several approaches have been investigated,
- Structured types can now be directly defined and used in Rodin through the Records plug-in,
- UML is more tightly integrated in Rodin, through new features implementation or state-machine animation,
- Code generation, to enable complete support for development, from high-level Event-B models down to executable implementations. A demonstrator tool has been developed.

The various parts making up this document are the following: general platform maintenance, mathematical extensions, provers, UML-B improvements, code generation, teamwork, scalability, model animation, and model-based testing.

Note that each of these parts is describing the improvements made, and is structured as follows:

- Overview. The involved partners are identified and an overview of the contribution is given.
- Motivations. The motivations for each tool extension and improvement are expressed.
- Choices / decisions. The decisions (e.g. design decisions) are justified.
- Available documentation. Some pointers to the Event-B wiki or related publications are listed.
- Planning. A timeline and the current status (as of 28 Jan 2011) is given.

References

[1] http://sourceforge.net/project/showfiles.php?group_id=108850&package_id=181714
[2] http://wiki.event-b.org

2 General Platform Maintenance

2.1. Overview

The main goal of the platform corrective and evolutive maintenance is to fix the listed known bugs, and implement some new requested features. As in the previous years of DEPLOY, these bugs and features are reported either by mail or through dedicated SourceForge trackers.

The terse list below gives an overwiew of the noteworthy features added in the main platform during the past year:

- Proof replay on undischarged POs (since release 1.3)

  It often happens, while modifying a model, that a set of previously manually discharged POs have slightly changed and need to be discharged again. However, replaying the proof for these POs could most of the time be enough to discharge it. Hence, a command was added to manually try replaying the proofs for a set of undischarged POs. This request comes directly from end users[1] . See [2] .

- Rule Details View (since release 2.0)

  When doing an interactive proof, one is guided by the proof tree appearing on the proof tree view. However, it is sometimes needed to get more information about the rules involved in a proof, such as instantiation details, used hypotheses, etc. The Rule Details View[3] displaying such details has been added.

- Refactory plug-in (since release 1.2)

  The Refactory[4] plug-in allows users of the Rodin platform to rename modelling elements. With a unique operation, both declaration and occurrences of an element are renamed. Moreover, renaming an element also modifies the corresponding proof, so that renaming does not change the proof status (no loss of proof).

- Mathematical extensions (since release 2.0)

  The integration of mathematical extensions required a major rework of the deep internals of the platform (in particular all code related to the manipulation of mathematical formulas). See [5] .

- Documentation

  Plug-in developers expressed their need to get a detailed documentation about Rodin extension ability. A dedicated tutorial[6] [7] has been written accordingly, and was the support of a full-day tutorial session given at the Rodin User and Developer Workshop[8] in Düsseldorf this year.

  The user manual, user tutorial and other developer documentation on the wiki[9] are continuously, and collaboratively updated and enhanced. Moreover, as soon as a new feature is added to the platform, the corresponding user documentation is created on the Wiki.

See the Release Notes[7] and the SourceForge[7] databases (bugs and feature requests) for details about the previous and upcoming releases of the Rodin platform.

2.2. Motivations

The evolutive maintenance (resp. corrective maintenance) has its origin in the DEPLOY description of work, and the various requests (resp. bug reports) listed by WP1-4 partners, developers and users. Since the DEPLOY project inception, various streams have been used to request new features or track known bugs:

- dedicated trackers[10] [11] ,
- platform mailing lists [12]
- DEPLOY WP9 mailing list.

Maintenance tasks to perform are collected from the aforementioned streams and scheduled during WP9 meetings. These tasks are processed in the same way as the task planned in the description of work.

The following table describes the main tasks (either performed or scheduled) motivating the evolutive maintenance:

Origin	Maintenance Task	Done in 2010	Scheduled in 2011
DoW / WP1-4 partners	Prover efficiency and integrity	x	x
Deliverable D25	Test reports and test coverage		x
WP1-4 partners	Updating fields of records	x	
WP1-4 partners	Team work	x	
WP1-4 partners	Edition		x
WP1-4 partners	Increase platform stability		x
WP1-4 partners	Comments everywhere [13]		x
WP1-4 partners	Plug-in incompatibilities		x
WP1-4 partners	Search in goal window [14]		x
WP1-4 partners	Preferences for the automatic tactics [15]		x
WP1-4 partners	Hierarchy / refinement view[16]	x	x
Plug-in developers	API to extend the Pretty Printer view [17]	x	
Plug-in developers	View the error log [18]	x	
Plug-in developers	Prover API	x	
Plug-in developers	A different update site for unstable plug-ins		x
End Users	64-bit Rodin for Mac	x	
End Users	Adding a replay proof command in the Event-B explorer [19]	x	
End Users	Having auto-completion in proof control [20]	x	
End Users	Displaying instantiated hypotheses [21]	x	
End Users	Displaying the inherited elements		x

2.3. Choices / Decisions

- Task priority

  Listed tasks are being given a priority during WP9 bi-weekly meetings, and then assigned to partners in charge of their processing. A higher priority is given to requests originating from deployment parteners.

- 64-bit release of Rodin for Mac platforms

  A major UI bug, due to some incompatibilities between Eclipse 3.5 and Java 1.6 on Mac platforms motivated the migration to the Eclipse 3.6 as basis for the Rodin 2.0 platform. In the meantime, as the 32-bit Java Virtual Machine is no longer supported on Mac platforms, Rodin migrated to Java 1.6, so that the release 2.0 of Rodin became a 64-bit Mac platform only.

  The Rodin platforms family is then composed of three executables : 32-bit platforms for Linux and Windows environments and a 64-bit platform for Mac computers.

- Rodin sources

  The sources of Rodin are now bundled together with the binary platform. It provides developers with a convenient alternative to the available sources[22] on SourceForge.

- Release notes

  The release notes contain information about the released plug-ins and centralise the requirements or existing issues which could not be stated at the main platform release date. Thus, since Rodin 2.0 release, it has been chosen to link the contents of the release notes text file included in Rodin releases, with the contents of the dedicated Wiki page.

2.4. Available Documentation

The following pages give useful information about the Rodin platform releases:

- Release notes[23] .
- Bugs[24] .
- Feature requests[25] .

2.5. Planning

For the coming year, the following topics pointed out at the last plenary meeting by the WP1-WP4 partners and encompassing end-user requests (see scheduled tasks in [26]) will be favoured by the WP9 partners:

- Platform stability and performances

  Currently, users struggle with editing or proving a model due to performance issues in the Rodin platform. Solving these issues represents a real challenge for the coming year and is mandatory for the industrial adoption of the Event-B methodology and Rodin platform.

- Prover efficiency and integrity

  Having all models automatically proved is the ideal goal. Thus, enhancing provers is a continuous task to be performed until the end of the DEPLOY project. Ensuring provers correctness and improving confidence in them is another important goal that will be pursued in the coming year.

- Plug-in incompatibilities

When several plug-ins are installed, conflicts between them can arise. The cumbersome behaviour spawned by such incompatibilities leads to users' disappointment or can render the platform unusable. Special efforts will be made to identify the source of incompatibilities among plug-ins. Moreover, necessary corrective maintenance tasks and assignments will be coordinated and executed.

References

[1] https://sourceforge.net/tracker/?func=detail&aid=2949606&group_id=108850&atid=651672
[2] http://wiki.event-b.org/index.php/Proof_Obligation_Commands
[3] http://wiki.event-b.org/index.php/Rodin_Proving_Perspective#Rule_Details_View
[4] http://wiki.event-b.org/index.php/Refactoring_Framework
[5] http://wiki.event-b.org/index.php/D32_Mathematical_Extensions
[6] http://wiki.event-b.org/index.php/Plug-in_Tutorial
[7] http://wiki.event-b.org/index.php/D32_General_Platform_Maintenance#Available_Documentation
[8] http://www.event-b.org/rodin10.html
[9] http://wiki.event-b.org
[10] http://sourceforge.net/tracker/?group_id=108850&atid=651669
[11] http://sourceforge.net/tracker/?group_id=108850&atid=651672
[12] http://sourceforge.net/mail/?group_id=108850
[13] https://sourceforge.net/tracker/index.php?func=detail&aid=3007797&group_id=108850&atid=651672
[14] https://sourceforge.net/tracker/?func=detail&atid=651672&aid=3092835&group_id=108850
[15] http://sourceforge.net/tracker/index.php?func=detail&aid=1581775&group_id=108850&atid=651672
[16] http://wiki.event-b.org/index.php/Project_Diagram
[17] http://sourceforge.net/tracker/?func=detail&aid=2926238&group_id=108850&atid=651672
[18] http://sourceforge.net/tracker/?func=detail&aid=2990974&group_id=108850&atid=651672
[19] http://sourceforge.net/tracker/?func=detail&aid=2949606&group_id=108850&atid=651672
[20] http://sourceforge.net/tracker/?func=detail&aid=2979367&group_id=108850&atid=651672
[21] http://sourceforge.net/tracker/?func=detail&aid=3008636&group_id=108850&atid=651672
[22] http://wiki.event-b.org/index.php/Using_Rodin_as_Target_Platform
[23] http://wiki.event-b.org/index.php/Rodin_Platform_Releases
[24] http://sourceforge.net/tracker/?atid=651669&group_id=108850
[25] http://sourceforge.net/tracker/?group_id=108850&atid=651672
[26] http://wiki.event-b.org/index.php/D32_General_Platform_Maintenance#Motivations

3 Mathematical Extensions

3.1. 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 (prover extensions).

A theory is a file that can be used to define new algebraic types, new operators/predicates and new proof rules. Theories are developed in the Rodin workspace, and proof obligations are generated to validate prover and mathematical extensions. When a theory is completed and (optionally) validated, the user can make it available for use in models (this action is called the deployment of a theory). Theories are deployed to the current workspace (i.e., Workspace Scope), and the user can use any defined extensions in any project within the workspace.

Records Plug-in has been developed by University of Southampton before the mathematical extensions as a new feature to provide structured types in Event-B. The plug-in extends Rodin standard context editor with a new modelling construct to provide support for structured types, which can be defined in terms of two new clauses: record declarations and record extensions. Both enable users to define their custom reusable types, that are treated underline by Rodin as Event-B constant sets and relations, supported by additional axioms, which the plug-in generates to simplify the proofs.

3.2. Motivations

Main reasons for implementing mathematical extensions are:

- increased readability ($a \text{ OR } b$ rather than $\text{bool}(a = TRUE \ \vee \ b = TRUE)$)
- polymorphism ($l \in List(S \times T)$)
- decreased proving effort, thanks to extension specific proof rules instead of general purpose ones

The Theory plug-in superseded the Rule-based Prover v0.3 plug-in, and is the placeholder for mathematical and prover extensions. It provides a high-level interface to the Rodin Core capabilities with regards to mathematical extensions. The Rule-based Prover was originally devised to provide an usable mechanism for user-defined rewrite rules through theories. Theories were, then, deemed a natural choice for defining mathematical extensions as well as proof rules to reason about such extensions. In essence, the Theory plug-in provides a systematic platform for defining and validating extensions through a familiar technique: proof obligations.

The motivation for development of Records plug-in was to fill the gap in Event-B language - a missing support of a syntax for the direct definition of structured types. Some of the industrial partners expressed a desire to have this missing feature in Event-B, that would allow them to define their own structured types such as records or classes. Theoretically these structures could be modelled with existing Event-B capabilities via projection functions. Backed up by a refined theoretical proposal Records plug-in was developed to extend the standard Event-B notation with requested capability.

3.3. Choices / Decisions

On the Core Rodin Platform side, implementing mathematical extensions required to make some parts of the code extensible, that were not designed to be so, namely the lexer and the parser. We were using tools that automatically generated them from a fixed grammar description, so we had to change to other technologies. A study [1] has been made on available technologies. The Pratt algorithm was selected for its adequation with the purpose and it did not have the drawbacks of other technologies:

- foreign language integration
- overhead due to over generality

After a mocking up phase to verify feasibility, the Pratt algorithm has been confirmed as the chosen option and implemented in the Rodin Platform.

Besides, we wanted to set up a way to publish and share theories for Rodin users, in order to constitute a database of pre-built theories for everyone to use and contribute. This has been realised by adding a new tracker on SourceForge site ([2]).

The Theory plug-in contributes a theory construct to the Rodin database. Theories were used in the Rule-based Prover (before it was discontinued) as a placeholder for rewrite rules. Given the usability advantages of the theory component, it was decided to use it to define mathematical extensions (new operators and new datatypes). Another advantage of using the theory construct is the possibility of using proof obligations to ensure that the soundness of the formalism is not compromised. Proof obligations are generated to validate any properties of new operators (e.g., associativity). With regards to prover extensions, it was decided that the Theory plug-in inherits the capabilities to define and validate rewrite rules from the Rule-based Prover. Furthermore, support for a simple yet powerful subset of inference rules is added, and polymorphic theorems can be defined within the same setting. Proof obligations are, again, used as a filter against potentially unsound proof rules.

Records plug-in required the extension of the Rodin database with the new constructs to support structured types. On the other hand the Event-B language itself did not support extension at that time. For that reason the decision was made to address extensibility problem at the lowest level possible, which was Rodin database, but to model structured types using standard Event-B notation at the level below. The translation from extended to standard syntax has been entrusted to the static checker, that was also extended for this purpose. Thus the plug-in provides the users with notation for record declarations and extensions in unchecked models, but the checked versions operate with standard Event-B constructs.

3.4. Available Documentation

- Pre-studies (states of the art, proposals, discussions).
  - *Proposals for Mathematical Extensions for Event-B* [3]
  - *Mathematical Extension in Event-B through the Rodin Theory Component* [4]
  - *Generic Parser's Design Alternatives* [5]
  - *Theoretical Description of Structured Types* [6]
- Technical details (specifications).
  - *Mathematical_Extensions wiki page* [7]
  - *Constrained Dynamic Lexer wiki page* [8]
  - *Constrained Dynamic Parser wiki page* [1]

  - *Theory plug-in wiki page* [9]
  - *Records Extension Documentation on wiki* [10]
- Teaching materials (tutorials).
- User's guides.
  - *Theory Plug-in User Manual* [11]

3.5. Planning

The Theory plug-in v2.1 is released. Work will continue on general maintenance, bug fixes as well adding new features as requested by the users of the plug-in.

References

[1] http://wiki.event-b.org/index.php/Constrained_Dynamic_Parser
[2] http://sourceforge.net/tracker/?group_id=108850&atid=1558661
[3] http://deploy-eprints.ecs.soton.ac.uk/216/
[4] http://deploy-eprints.ecs.soton.ac.uk/251/
[5] http://wiki.event-b.org/index.php/Constrained_Dynamic_Parser#Design_Alternatives
[6] http://wiki.event-b.org/index.php/Structured_Types
[7] http://wiki.event-b.org/index.php/Mathematical_Extensions
[8] http://wiki.event-b.org/index.php/Constrained_Dynamic_Lexer
[9] http://wiki.event-b.org/index.php/Theory_Plug-in
[10] http://wiki.event-b.org/index.php/Records_Extension
[11] http://wiki.event-b.org/images/Theory_UM.pdf

4 Provers

4.1. Overview

Concerning Rodin's provers the following contributions have been made:

- Jann Röder (ETH Zurich) has developed a relevance filter plug-in. The plug-in provides a proof tactic that first removes hypotheses from a given sequent according to several heuristics. The tactic then inputs the reduced sequent to one or several of Rodin's external provers (PP, newPP, ML). Jann Röder carried out experiments using Event-B models from different domains and observed that his tactic significantly increases the number of proof obligations proved automatically.
- Matthias Schmalz (ETH Zurich) formally expressed the theoretical foundations of Event-B's logic. "Event-B's logic" stands for the formalism in which, e.g., guards, invariants, axioms, and theorems are expressed, and proof obligations are expressed and proved. He provides a rigorous specification of syntax, semantics, proofs, theories, and mathematical extensions [1] in one document. The document encompasses a small theory "Core", proves "Core"'s soundness, and shows how to define the remaining operators, types, and binders available in Rodin using mathematical extensions. The document thus provides a proof calculus for Event-B that is sound by construction, and a methodology for reasoning about the soundness of Event-B proof rules within Event-B. The document also allows users to look-up definitions of predefined operators and binders, answering questions like "what is the meaning of $x \div y$ if x or y is negative". For developers, it sheds some light on intricate questions concerning partial functions, e.g., why it is sound to rewrite $x \in \{y \mid \varphi(y)\}$ to $\varphi(x)$ but unsound (in general) to rewrite $\varphi(x)$ to $x \in \{y \mid \varphi(y)\}$.
- Systerel improved the support for well-definedness in the sequent prover. The new implementation generates much smaller well-definedness predicates and new automated tactics have been added to discharge most of the well-definedness subgoals, thus rendering well-definedness almost transparent to the end-user.

4.2. Motivations

4.2.1. Relevance Filtering

Rodin's external provers (PP, newPP, and sometimes also ML) tend to perform poorly in the presence of irrelevant hypotheses. For PP and newPP the user can still manually select the hypotheses he considers relevant, but that is a tedious and error-prone process, in particular for large models. Several heuristics for selecting relevant hypotheses have been proposed in the literature[2] [3] [4] [5] . The relevance filter plug-in implements these and other heuristics, and provides a default configuration that has been shown to be almost optimal on a given collection of models from different domains[6] . The relevance filter plug-in has also significantly increased the number of automatically discharged proof obligations on models of deployment partners, which had not been used for fine tuning the heuristics.

4.2.2. Foundations of Event-B's Logic

As Rodin is used to develop safety critical systems, bugs in Rodin's theorem prover constitute a serious problem. Unfortunately, several bugs have been discovered that make Rodin's theorem prover unsound. Obviously, any examination of soundness presupposes a clearly written specification of the logic's syntax, semantics, and proof calculus. There are several publications on the logic of Event-B, but they fail to serve as specification documents, because the logic defined therein is inconsistent [7] or only fragments of the logic implemented in Rodin are considered [8] [9] . Therefore we have devised a rigorous specification document for the logic of Event-B [10] .

Mathematical extensions[1] play an important role in avoiding unsoundness, because they allow the user to define new operators, binders, types, and inference and rewrite rules in a soundness preserving fashion. The specification document [10] also devises the theoretical foundations of mathematical extensions. Note that mathematical extensions are well-understood for, e.g., HOL[11] , but the extension methods for HOL cannot be straightforwardly adopted for Event-B because of Event-B's well-definedness [12] mechanism and non-standard term rewriting.

4.2.3. Improved WD Support

When the user enters an expression or predicate that is possibly ill-defined (such as applying a partial function), the Rodin platform insists that the user demonstrates that this formula is indeed well-defined (e.g., the partial function is applied to an element of its domain) before using it. This verification is implemented by generating a well-definedness (WD) predicate, based on the syntax of the input formula.

In previous releases of the Rodin platform, the generation of WD predicates was implemented in a very simple manner, and the generated predicate was usually highly redundant. Moreover, the support by automated tactics for discharging such predicates was not always appropriate. Consequently, many well-definedness subproofs needed to be carried out interactively in a very cumbersome and repetitive manner.

4.3. Choices / Decisions

4.3.1. Relevance Filtering

The relevance filter heuristics we have considered do not work out of the box - their parameters need to be carefully adjusted. The major design decision concerned how to carry out the process of fine tuning. We started with an ad-hoc benchmark containing models of several problem domains and aimed for maximizing the number of automatically discharged proof obligations among this benchmark while minimizing the amount of time spent for proving. We experimented with different filter configurations, i.e., combinations of heuristics, heuristic parameters, provers (PP, newPP, or ML) and prover timeouts. Finally, the parameters and timeouts were chosen such that

- the number of automatically discharged proof obligations is almost maximal among all considered filter configurations, and
- decreasing the timeouts would significantly decrease the number of automatically discharged proof obligations.

To rebut criticism of overfitting, we tested the final filter configuration on a validation benchmark (based on deployment partners' models), which was chosen independently from the benchmark used for fine-tuning. We observed that the final filter configuration significantly increases the number of automatically discharged proof obligations among the validation benchmark in comparison to not using relevance filtering.

4.3.2. Foundations of Event-B's Logic

The major design decision concerned the logic in which the semantics of Event-B's logic is formalized. We experimented with ZF set theory and HOL. Finally, we decided to define semantics in terms of a (shallow) embedding into HOL, because that allows us to carry out vast parts of our soundness proofs using Isabelle/HOL[13] . In the long term, the embedding allows us to use Isabelle/HOL as an external theorem prover for Rodin.

Other design decisions, e.g., concerning terminology, are discussed in [10] .

4.3.3. Improved WD Support

To improve generated WD lemmas, the generating algorithm has not been changed (to ensure safety) but enhanced by a back-end that simplifies the generated lemma after the fact. The enhancement consists in removing all sub-predicates that are subsumed within the WD lemma.

Also, as WD lemmas were changing between two releases of the platform, the automated proof replay mechanism needed to better tackle changes in proof obligations (when they get simpler). This allows user to retain their proof status, although proof obligations have changed.

As concerns automated support, it has been chosen not to add new reasoners (to avoid expanding the trusted base of the sequent prover) but rather to work on the outside by adding new tactics that schedule the existing reasoners to discharge the WD subgoals. This approach also allowed to start introducing speculative reasoning within tactics (attempt proofs).

4.4. Available Documentation

- The internals of the relevance filter plug-in and the process of fine tuning are documented in J. Röder's Master thesis.[6]
- A rigorous specification of Event-B's logic (for Rodin developers) and a reference document containing the definitions of built-in symbols (for Rodin developers and users) can be found in "The logic of Event-B" report.[10]
- The specification of the Improved WD Lemma Generation [14] is available from the Rodin Wiki.

4.5. Planning

In DEPLOY's fourth year, we intend to provide a link-up between Rodin and Isabelle/HOL. That allows us to implement proof tactics that internally use Isabelle/HOL to discharge the given sequent. Consistency of these tactics depends merely on the consistency of Isabelle/HOL and correctness of the translation from Event-B to Isabelle/HOL, which is quite straightforward. As Isabelle/HOL comes with link-ups to first-order solvers such as E[15] , Spass[16] , and Vampire[17] and SMT solvers such as Z3[18] , a link-up between Rodin and Isabelle/HOL makes these solvers also available to Rodin.

References

[1] http://wiki.event-b.org/index.php/D32_Mathematical_Extensions
[2] K. Hoder. SUMO infernce engine. (http://www.cs.manchester.ac.uk/~hoderk/sine)
[3] J. Meng and L. C. Paulson. Lightweight relevance filtering for machine-generated resolution problems. Journal of Applied Logic, 7(1);41-57, 2009.
[4] A. Roederer, Y. Puzis, and G. Sutcliffe. Divvy: an atp meta-system based on axiom relevance ordering. In CADE, pages 157-162, 2009.
[5] G. Sutcliffe and Y. Puzis. SRASS - a semantic relevance axiom selection system. In CADE, pages 295-310, 2007.
[6] J. Röder. Relevance filters for Event-B. Master Thesis, ETH Zurich, 2010. (http://e-collection.ethbib.ethz.ch/view/eth:2278?q=event-b)
[7] J.-R. Abrial. Modeling in Event-B: system and software engineering. Cambridge University Press, 2010 (http://www.event-b.org/abook.html)
[8] F. D. Mehta. Proofs for the working engineer. PhD Thesis, ETH Zurich, 2008. (http://e-collection.ethbib.ethz.ch/eserv/eth:30601/eth-30601-02.pdf)
[9] C. Metayer and L. Voisin. The Event-B mathematical language, 2009. (http://deploy-eprints.ecs.soton.ac.uk/11/4/kernel_lang.pdf)
[10] M. Schmalz. The logic of Event-B. Technical Report 698, ETH Zurich, Switzerland, 2010. (ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/6xx/698.pdf)
[11] M. J. C. Gordon and T. F. Melham. Introduction to HOL. Cambridge University Press, 1993.
[12] http://wiki.event-b.org/index.php/Well-definedness_in_Event-B
[13] T. Nipkow, L. C. Paulson, and M. Wenzel. Isabelle/HOL - a proof assistant for higher-order logic. LNCS 2283, 2002.
[14] http://wiki.event-b.org/index.php/Improved_WD_Lemma_Generation
[15] S. Schulz. E - a brainiac theorem prover. AI Commun. 15(2-3);11-126, 2002.
[16] SPASS: an automated theorem prover for first-order logic with equality. (http://www.spass-prover.org)
[17] A. Riazanov and A. Voronkov. The design and implementation of VAMPIRE. AI Commun. 15(2-3);91-110, 2002.
[18] L. M. de Moura and N. Bjorner. Z3: an efficient SMT solver. TACAS, pages 337-340, 2008.

5 UML-B Improvements

5.1. Overview

Progress on UML-B consists of three parallel developments.

1. Enhancement and maintenance of the current and existing UML-B plug-in with new functionality and usability features.
2. Development of a new plug-in to provide animation of UML-B state-machine diagrams.
3. Development of a new plug-in (called iUML-B) that provides an alternative to UML-B which is more closely integrated with Event-B.

5.1.1. Enhancement and Maintenance of Existing UML-B

The main progress on UML-B has been to implement new features, improve usability and fix bugs. As in the previous years of DEPLOY, these bugs and features are reported either by email or through dedicated SourceForge trackers. The list below gives an overwiew of the noteworthy features added in UML-B during the past year:

1. Functional enhancements to modelling
   - State-machine transitions emanating from multiple states. It is often the case that a transition may occur from several (possibly all) states within a state-machine. Such models were impossible to represent in UML-B. Two pseudo-states were added to represent this. Firstly an 'ANY' pseudo-state can be used as a transition source to represent that the transition can occur from ANY state of the state-machine. Secondly a disjunctive pseudo-state can be used to combine several transitions from different source states into a single transition.
   - Conceptual Singleton classes - provides a conceptual grouping of associated modelling elements without generating the lifting mechanisms of a class.
   - Super-type arrows to target ExtendedClassTypes and RefinedClasses - this functionality was missing in previous versions.
   - Event convergence property on state-machine transitions - convergence was previously available only on events
2. Enhancements to improve usability
   - Report to user if translation didn't proceed due to model validator - previously, it was not clear when the model had failed validation and the translation had not been executed.
   - Improve refresh of diagrams - in some situations the diagram graphics did not update error marking and property changes unless some other event caused a refresh.
   - Improvements and additions to model validations - some model validations were inconsistent or incomplete.
   - Preference for line routing style for each diagram type - allows the user to choose whether to use rectilinear or oblique line routing for each diagram type.
3. Corrections
   - Correct and improve missing default labelling in diagrams.
   - Corrections and improvements to automatic diagram deletion.
   - Improved management of diagram files when model changes.
   - Add missing comment fields in properties view.

5.1.2. UML-B State-machine Animation Plug-in

The UML-B State-machine Animation Plug-in is a new feature developed by University of Southampton as a response to a request from industrial partners to support the animation of UML-B state-machine diagrams. The essence of the request was to provide a means of visualising the animation and model-checking process of Event-B machines modelled in UML-B tool, in particular state-machines, thus to simplify this process. The tool integrates the capabilities of ProB animation and UML-B State-machine notation.

5.1.3. iUML-B - Integrated UML-B

The prototype iUML-B plug-in (not yet released) is an extension to the Event-B EMF framework. It will consist of a collection of independent plug-ins that provide support for diagrammatic modelling integrated with Event-B textual modelling. At this stage a plug-in to show the project structure (in terms of machines and contexts and their relationships) has been released. A plug-in to support state-machine diagrams integrated with textual Event-B is at a prototype stage and nearing release. Plug-ins to support other kinds of diagram are in the early stages of development.

5.2. Motivations

5.2.1. Enhancement and Maintenance of Existing UML-B

The aim is to continually improve the usability of UML-B in the light of experience. Although the motivation for a more integrated version of UML-B that would be attractive to experienced Event-B users has been recognised for some time, the original aim of UML-B was to make formal modelling more accessible to new users. This aim remains valid and therefore the current non-integrated UML-B should be developed and enhanced wherever possible. Therefore, some issues concerning its usability have been identified and rectified. Some of these issues required enhancements to the modelling language. In particular, several case studies have highlighted the need for transitions that may be taken from any sub-state or a range of sub-states.

5.2.2. UML-B State-machine Animation Plug-in

The motivation for the State-machine Animation plug-in was to extend UML-B with animation capabilities similar to those that the ProB tool provides for Event-B models. With the aid of such a plug-in, animation and model checking would be possible on UML-B diagrams instead of translated and less obvious Event-B code. The State-machine Animation plug-in uses the ProB tool to animate the translated Event-B models and provides an animation interface based on the UML-B State-machine models.

5.2.3. iUML-B - Integrated UML-B

The motivations for an alternative version of UML-B, that is tightly integrated with Event-B, was recognised at the beginning of the Deploy project. The aim of UML-B was to provide an easier modelling environment for users that were familiar with UML-like modelling but not with textual mathematical notations. Experienced Event-B users would also benefit from the visualisations provided by UML-B but they do not wish to be

distanced from the generated Event-B and would prefer to be able to create parts of their models in the Event-B notation alongside the diagrammatic parts.

5.3. Choices / Decisions

5.3.1. Enhancement and Maintenance of Existing UML-B

As UML-B is already a relatively mature product attracting strong interest from various institutions, the main goal is currently to improve usability and stability of the tool rather than develop major new features. Consequently development has concentrated primarily on improving the basic tooling and user interface. However, if users are limited by the modelling language, which can be interpreted as an usability issue, new features to the language may be added.

5.3.2. UML-B State-machine Animation

The initial design decision was to extend the UML-B metamodel with the animation components. Due to difficulties with UML-B diagram extensibility an alternative option was determined to create a separate model, derived from UML-B state-machine subset, with incorporated animation support. This design was successfully implemented together with ProB and Rodin UI extensions into Animation plug-in, which supports such UML-B concepts as classes and different state-machine translation kinds, as well as Event-B refinement.

5.3.3. iUML-B - Integrated UML-B

A precursor stage to developing iUML-B was to develop an EMF representation of Event-B. This was completed last year and is now used successfully by several plug-ins. A Records plug-in was developed in response to user requests. The Records plug-in was implemented as an extension to the Event-B EMF framework. This was seen as a 'practice run' before attempting a similar extension to support UML-B. However, the Records plug-in took longer than expected and this has delayed work on iUML-B. Some progress on iUML-B has recently been made with the release of a project level diagram tool for Event-B. A plug-in for State-Machine diagram models as an extension to the Event-B EMF models is currently being developed and is nearing release.

5.4. Available Documentation

The following pages give useful information about UML-B:

- Lectures[1].
- Tutorials[1].
- Worked Examples[1].

UML-B State-machine Animation Plug-in:

- General information[2]
- Tutorial[3]

5.5. Planning

The current version of UML-B will continue to be enhanced. This may include some new modelling features such as better support for synchronisation of state-machines and support for more UML modelling details. However, usability of the current features is seen as the main objective. This will include,

- Support for copy, cut and paste of diagram elements so that they can be moved and/or replicated more easily,
- Support for re-attaching links (e.g. transitions) to different source/target elements,
- Facilities for refactoring/renaming elements,
- Support for the event extension mechanism of Event-B,
- Integration of Context Diagram model elements on Class diagrams,
- Improve facilities for navigating between state-machines and visualising multiple state-machines.

The new iUML-B tools will continue to be developed and released including,

- Enhancement of the Project Diagram Plugin for Event-B to make it extensible and/or to automatically cater for future component types,
- Release of the State-machine diagram plug-in as an integrated part of Event-B modelling,
- Development of a new plug-in to support event refinement diagrams.

References

[1] http://wiki.event-b.org/index.php/UML-B
[2] http://wiki.event-b.org/index.php/UML-B_-_Statemachine_Animation
[3] http://wiki.event-b.org/index.php/Statemachine_Animation_Tutorial

6 Code generation

6.1. General Overview

The code generation activity has been undertaken at the University of Southampton. This has been a new line of work for DEPLOY that was not identified in the original Description of Work for the project. The development of the approach, and the tools to support, it involved a number of team members at Southampton; and also at other institutions. This work draws on our recent experience with technologies such as *Shared Event Decomposition* [1] , and the *EMF Framework for Event-B* [2] . There was collaboration at an early stage with Newcastle University, where we explored the commonalities between their flow plug-in [3] and the flow control structures used in our approach. Collaboration with the University of York was also established since we chose to use their *Epsilon* [4] model-to-model transformation technology.

6.2. Motivations

The decision was taken in 2009 to include code generation as a project goal [5] . It had been recognised that support for generation of code from refined Event-B models would be an important factor in ensuring eventual deployment of the DEPLOY approach within their organisations. This was especially true for Bosch and Space Systems Finland (SSF). After receiving more detailed requirements from Bosch and SSF, it became clear we should focus our efforts on supporting the generation of code for typical real-time embedded control software.

6.3. Choices / Decisions

6.3.1. Strategic Overview

During the last year we have focussed on supporting the generation of code for typical real-time embedded control software. To this end we have evolved a multi-tasking approach which is conceptually similar to that of the Ada tasking model. Tasks are modelled by an extension to Event-B, called *Tasking Machines*. Tasking Machines are an extension of the existing Event-B Machine component. In implementations such as Ada, tasks share the resources and have mutually exclusive access to shared state through the use of a protection mechanism. An Event-B machine can also be viewed as an abstraction of a shared resource, and the mechanism protecting it. We use existing Event-B machines with minimal extensions (called Shared Machines) to represent shared resources.

For real-time control, periodic and one-shot activation is currently supported; and it is planned to support triggered tasks in the near future. Tasks have priorities to ensure appropriate responsiveness of the control software. For the DEPLOY project, it was regarded as sufficient to support construction of programs with a fixed number of tasks and a fixed number of shared variables – no dynamic creation of processes or objects has been accommodated.

Our main goal this year has been to devise an approach for, and provide tool support for, code generation (initially to Ada). In accord with the resources available during the year it was decided to limit the provision of tool support to that of a demonstrator tool. The tool is a proof-of-concept only, and lacks the productivity enhancements expected in a more mature tool. Nevertheless much insight has been gained in undertaking this work; it lays a foundation for future research,

and will be useful since it will allow interested parties to explore the approach.

6.3.2. The Tasking Extension for Event-B

The following text can be read in conjunction with the slides[6] from the Deploy Plenary Meeting - Zurich 2010.

Tasking Event-B can be viewed as an extension of the existing Event-B language. We use the existing approaches of refinement and decomposition to structure a development that is suitable for construction of a Tasking Development. At some point during the modelling phase parameters may have to be introduced to facilitate decomposition. This constitutes a natural part of the refinement process as it moves towards decomposition and on to the implementation level. During decomposition parameters form part of the interface that enables event synchronization. We make use of this interface and add information (see Events[7]) to facilitate code generation.

A Tasking Development is generated programmatically, at the direction of the user; the Tasking Development consists of a number of machines (and perhaps associated contexts). In our approach we make use of the Event-B EMF extension mechanism which allows addition of new constructs to a model. The tasking extension consists of the constructs in the following table.

Construct	Options
Machine Type	DeclaredTask, AutoTask, SharedMachine
Control	Sequence, Loop, Branch, EventSynch
Task Type	Periodic(n), Triggered, Repeating, OneShot
Priority	-
Event Type	Branch, Loop, ProcedureDef, ProcedureSynch
Parameter Type	ActualIn, ActualOut, FormalIn, FormalOut

The machines in the Tasking Development are extended with the constructs shown in the table, and may be viewed as keywords in a textual representation of the language. With extensions added, a Tasking Development can be translated to a common language model for mapping to implementation source code. There is also a translator that constructs new machines/contexts modelling the implementation, and these should refine/extend the existing elements of the Event-B project.

6.3.3. Tasking Machines

The following constructs relate only to Tasking Machines, and provide implementation details. Timing of periodic tasks is not modelled formally. Tasking Machines are related to the concept of an Ada task. These can be implemented in Ada using tasks, in C using the pthread library C, or in Java using threads.

- Tasking Machines may be characterised by the following types:
  - AutoTasks - Singleton Tasks.
  - Declared tasks - (Not currently used) A task template relating to an Ada *tasktype* declaration.
  - TaskType - Defines the scheduling, cycle and lifetime of a task. i.e. one-shot periodic or triggered.

- Priority - An integer value is supplied, the task with the highest value priority takes precedence when being scheduled.

6.3.4. Shared Machines

A Shared Machine corresponds to the concept of a protected resource, such as a monitor. They may be implemented in Ada as a Protected Object, in C using mutex locking, or in Java as a monitor.

- Applied to the Shared Machine we have:
  - A SharedMachine *keyword* that identifies a machine as a Shared Machine.

6.3.5. Tasks and Events

6.3.5.1. Control Constructs

Each Tasking Machine has a *task body* which contains the flow control, or algorithmic, constructs.

- We have the following constructs available in the Tasking Machine body:
  - Sequence - for imposing an order on events.
  - Branch - choice between a number of mutually exclusive events.
  - Loop - event repetition while it's guard remains true.
  - Event Synchronisation - synchronization between an event in a Tasking Machine and an event in a Shared Machine. Synchronization corresponds to an subroutine call with atomic (with respect to an external viewer) updates. The updates in the protected resource are implemented by a procedure call to a protected object, and tasks do not share state. The synchronization construct also provides the means to specify parameter passing, both in and out of the task.
  - Event wrappers - The event synchronization construct is contained in an event wrapper. The wrapper may also contain a single event (we re-use the synchronization construct, but do not use it for synchronizing). The event may belong to the Tasking Machine, or to a Shared Machine that is visible to the task. Single events in a wrapper correspond to a subroutine call in an implementation.

6.3.5.2. Implementing Events

An event's role in the implementation is identified using the following extensions which are added to the event. Events used in task bodies are 'references' that make use of existing event definitions from the abstract development. The events are extended. to assist with translation, with a keyword indicating their role in the implementation.

- Event implementation.
  - Branch - In essence a task's event is split in the implementation; guards are mapped to branch conditions and actions are mapped to the branch body. If the branch refers to a Shared Machine event (procedureDef) then this is mapped to a simple procedure call.
  - Loop - The task's event guard maps to the loop condition and actions to to loop body. If the loop refers to a Shared Machine event then it is mapped to a simple procedure call.
  - ProcedureSych - This usually indicates to the translator that the event maps to a subroutine, but an event in a task may not require a subroutine implementation if its role is simply to provide parameters for a procedure call.

- ProcedureDef - Identifies an event that maps to a (potentially blocking) subroutine definition. Event guards are implemented as a conditional wait; in Ada this is an entry barrier, and in C may use a pthread condition variable .

In an implementation, when an subroutine is defined, its formal parameters are replaced by actual parameter values at run-time. To assist the code generator we extend the Event-B parameters. We identify formal and actual parameters in the implementation, and add the following keywords to the event parameters, as follows:

- Event parameter types
  - FormalIn or FormalOut - event parameters are extended with the ParameterType construct. Extension with formal parameters indicates a mapping to formal parameters in the implementation.
  - ActualIn or ActualOut - Extension with an actual parameter indicates a mapping to an actual parameter in the implementation.

6.3.6. Other Technical Issues

6.3.6.1. Translation Technology

In order to provide a structured extensible code generation tool it was decided to use a multi-stage translation approach. The Event-B EMF model provided by the Event-B EMF Framework is extended to accommodate the tasking constructs as described above. The Tasking model is then translated to an intermediate model, the Common Language Model. The Common Language Meta-model is an abstraction of some useful generic programming constructs such as sequence, loop, branch and subroutine call/definition and so on. The translation of the Common Language Model to programme source code is then a relatively small step. The main translation activity takes place in the step between Tasking and Common Language models.

The decision was made to use Epsilon [4] to facilitate model to model translation for this stage. It was felt that an extensible, easily maintainable solution was required for this. Various model-to-model technologies (Java code, ATL, Epsilon) were appraised and it was judged that the Epsilon tool best matched our requirements. It proved to be a good choice initially for the specification of translations, especially in simpler areas of the project where the correspondence between models were simple. However the lack of debugging facilities, and productivity enhancements that are found in more mature tools, somewhat hindered rapid development as the project increased in complexity.

6.3.6.2. Implementation - Source Code

Early in the current phase of work we identified the possibility of translating the Common Language Model to EMF models of programming languages such as Ada and C, in addition to producing textual source. While the EMF route still remains an option, it was decided that we would produce a PrettyPrinter for the Ada code. This allows a user to cut and paste the Ada source code from the PrettyPrinter window to an Ada editor, and was the optimal route to code for this phase of the code generation activity in DEPLOY.

6.3.6.3. Editing the Tasking Model

The editor for the Tasking Development is based on a EMF tree-editor. The tree editor provides a facility for adding the extensions to Event-B constructs. The readability of the editor is enhanced by a PrettyPrinter, which provides a textual version of the Tasking Development, which is easier to read. It is envisaged that the textual notation will be fully integrated as a Camille extension when the facility/resources become available.

6.3.7. The Tool Deliverable

The demonstrator tool was released on 30 November 2010, and is available as an update site, or bundled Rodin package from:

```
https://sourceforge.net/projects/codegenerationd/files
```

Sources are available from:

```
https://codegenerationd.svn.sourceforge.net/svnroot/codegenerationd
```

The tool is based on a build of Rodin 1.3.1 (not Rodin 2.0.0 due to dependency conflicts).

- The Code Generation tool consists of,
  - a Tasking Development Generator.
  - a Tasking Development Editor (Based on an EMF Tree Editor).
  - a translator, from Tasking Development to Common Language Model (IL1).
  - a translator, from the Tasking Development to Event-B model of the implementation.
  - a pretty-printer for the Tasking Development.
  - a pretty-printer for Common Language Model, which generates Ada Source Code.

6.4. Available Documentation

6.4.1. Technical Background

Much insight was gained during the work on code generation reported in the thesis *Providing Concurrent Implementations for Event-B Developments* [8]

Tooling issues were reported in a paper *Tool Support for Event-B Code Generation* [9] which was presented at *Workshop on Tool Building in Formal Methods*, http://abzconference.org/

There are technical notes available [10] , that give more precise details of the approach and the mapping between Event-B and the common language meta-model, and its corresponding Event-B model.

6.4.2. For Users

There is an overview at Tasking Event-B Overview for D32[11]

There is a wiki page at Code Generation Activity[12]

There is a tutorial at Code Generation Tutorial[13]

6.5. Planning

During 2011 we plan to develop the code generation tools further, taking on board any feedback from interested parties. The tool support should advance to the prototype stage, with improvements in the tool's usability in terms of features and user experience.

References

[1] http://wiki.event-b.org/index.php/Event_Model_Decomposition
[2] http://wiki.event-b.org/index.php/EMF_framework_for_Event-B
[3] http://wiki.event-b.org/index.php/Flows
[4] http://www.eclipse.org/gmt/epsilon/
[5] http://wiki.event-b.org/index.php/D23_Code_Generation
[6] http://deploy-eprints.ecs.soton.ac.uk/260/2/
CGSlidesAndy%2520Edmunds%2520-%2520Code%2520Generation%2520Slides.pdf
[7] http://wiki.event-b.org/index.php/D32_Code_generation#Implementing
[8] http://eprints.ecs.soton.ac.uk/20826/
[9] http://eprints.ecs.soton.ac.uk/20824/
[10] http://wiki.event-b.org/images/Translation.pdf
[11] http://wiki.event-b.org/index.php/Tasking_Event-B_Overview_for_D32
[12] http://wiki.event-b.org/index.php/Code_Generation_Activity
[13] http://wiki.event-b.org/index.php/Code_Generation_Tutorial

7 Teamwork

7.1. Overview

Teamwork consists of

- **Team-working Plug-in** is a new feature developed by University of Southampton in request to industrial partners who required support of Rodin project management and team development using Subversion system. Having this support would bring the benefit of centralised model storage and versioning, as well as parallel development. Despite a few functional limitations, derived by specific nature of the Rodin projects, the implemented plug-in gives support for Subversion-based project sharing and collaborative development.
- **Decomposition Plug-in** was developed by Renato Silva (University of Southampton), Carine Pascal (Systerel) based on the initial prototype developed by T.S. Hoang (ETH Zurich). This plug-in was developed as an answer to models that became to big to be handled with a large number of events, a large number of variables and consequently a large number of proof obligations over several levels of refinements. There are two kinds of decomposition available: *shared event* (studied initially by Michael Butler[1] [2]) and *shared variable* (studied initially by Jean-Raymond Abrial et al [3] [4]). Both decomposition styles allow the partition of the original model into (smaller) sub-models. The sub-models are expected to be easier to handle, with less variables and less events and less proof obligations. This partition is done in a way that the sub-models (also referred as sub-components) are independent of each other and therefore can be refined individually. As a consequence, each sub-model can be further developed by different people allowing teamwork development.

7.2. Motivations

Main reasons for implementing teamwork are:

- SVN Teamwork

  The reason to support compatibility of Rodin projects with Subversion was to allow Rodin users to share their projects and work on them together, as well as have the benefits of versioning and revision control, provided by the SVN system. It was difficult to work on models in parallel and manage changes made by different parties, especially for big and complex models. Other users expressed a concern on safety aspect of collaborative development, thus pointing out the benefits of centralised repository storage of the models under development on SVN.

- Decomposition

  Difficulties in managing complex models (in particular for a large number of proof obligations) fed the idea of decomposing a model in a way that the resulting sub-models could be developed by different individuals. The decomposition process should be seen as a refinement step where the original properties and respective proof obligations should remain valid. With shared event and shared variable decomposition, these requirements are preserved, with the advantage of simplifying the overall development by dealing with sub-parts of the model at once in each sub-model.

7.3. Choices / Decisions

- SVN Teamwork

  The desired objective of a plug-in that would bring support for Subversion in Rodin was to make a Rodin project compatible with standard SVN interface. Due to nature of the Rodin resource management, in particular the use of Rodin database and non-XMI serialisation, it turned out a hard task. A solution to this difficulty was to provide an alternative serialisation method, that would be compatible with Subversion interface. XMI serialisation has been chosen in the final plug-in, which together with Event-B EMF framework provides a shareable copy of the resources of a Rodin project and takes care of synchronisation between two.

- Decomposition

  The two styles of decomposition use as criteria of partition two of the most important elements of an Event-B model: variables and events. The plug-in supports the two styles and allows the decomposition through a stepwise wizard or through a *decomposition file* (with extension *.dcp*) that can be stored are re-run whenever necessary. For the shared event decomposition, the user needs to selects which variables are allocated to which sub-component. For the share variable decomposition, the user selects which events with be part of which sub-component. The rest of the sub-component (which is no more than an ordinary machine) is built automatically (after some validations).

7.4. Available Documentation

- SVN Team-based development documentation[5]
- Decomposition plug-in user guide [6]
- Event Model decomposition for shared variable approach [7]
- Decomposition tool for Event-B [8]

7.5. Planning

This paragraph shall give a timeline and current status (as of 28 Jan 2011).

- Decomposition
  - Solve compatibility problems with other plug-ins: Records, Modularisation
  - Introduction of a graphical interface for decomposition, where the user *drags and drop* the elements to the respective sub-component.

References

[1] http://eprints.ecs.soton.ac.uk/16965/
[2] http://eprints.ecs.soton.ac.uk/16910/
[3] http://iospress.metapress.com/content/c74274t385t6r72r/
[4] http://www.inf.ethz.ch/research/disstechreps/techreports/
[5] http://wiki.event-b.org/index.php/Team-based_development
[6] http://wiki.event-b.org/index.php/Decomposition_Plug-in_User_Guide
[7] http://wiki.event-b.org/index.php/Event_Model_Decomposition
[8] http://eprints.ecs.soton.ac.uk/21714/

8 Scalability

8.1. Overview

Regarding scalability of the Rodin platform, the following contributions has been made:

- Flows plug-in development. The main purpose of the Flows plug-in [1] is to give a modeller a clean and concise view of the control flow aspects of a model without cluttering the model with program counter variables and associated guards and actions. The plug-in is essentially a graphical notation for formulating certain kind of theorems. The new version is to be released in February 2011 and this will include facilities for expressing a wider range of enabledness properties as well as tools for describing event refinement. It is going to address one of the critical shortcomings of the previous version: generation of unwieldy proof obligations (a large disjunction in a goal comprising several hundreds of terms). The tool is developed by Newcastle University with the requirements and suggestions from Bosch and SAP.
- Group Refinement. Group refinement plug-in is a tool realising an alternative set of Event-B refinement laws in the Rodin platform. It lets a modeller to switch to differing style of event atomicity refinement for the scope of a single refinement steps. For a certain case of atomicity refinement the alternative laws result in a more natural and compact model with fewer and simpler proof obligations. The method and the tool were development by Newcastle University, the pattern of refinement was discovered by the Bosch formal modelling team.
- Modes. The Mode and Fault Tolerance Views plug-in is a modelling environment for constructing modal and fault tolerance features in a concise manner and formally linking them to Event-B models. As many systems are developed using the notion of operation modes, the tool allows to separate the modal modelling as a refinement chain from the main Event-B models. This makes possible to model explicitly modal behaviour and certain fault tolerance aspects of systems and formally show the consistency between the models and their views. The theory and plug-in were developed by Newcastle University.
- Emre Yilmaz and Thai Son Hoang (ETH Zurich) has developed a plug-in supporting an extension of Event-B with qualitative reasoning. The extension allows developer to declare an event converges probabilistically (with probability 1). This is in contrast with standard certain termination, where convergent events *must decrease* the declared variant. Provided that the variant is bounded above, an event probabilistically converges if it *might decrease* the variant. The extension enables Event-B to model systems with *almost certain termination* properties, e.g. IEEE 1394 Firewire protocol[2] or Rabin's Choice Coordination algorithm [3] .

8.2. Motivations

8.2.1. Flow plug-in

The flows tool was applied by Bosch in the development of the cruise control model to verify deadlock-freedom and liveness properties of the model. Being a sizable case-study, this was an important test for the ideas and techniques behind the plug-in. The general conclusion was that a tool of this kind is essential and the current version should be improved in many directions. This experience has uncovered a rather fundamental issues with the size and complexity of the generated proof obligations. These were the largest theorems ever generated in Rodin and the only positive aspect is that this helped to stress-test and debug the proof handling facilities of the

Rodin. It was clear that such proof obligations can never be comfortably handled by Rodin tools (although there were some encouraging results in the application of ProB as a disprover for these kind of proofs) and it was decided that the approach to proof generation requires a complete redesign.

8.2.2. Group Refinement

One of the project industrial partners (Bosch) has identified a recurring refinement pattern that did not fit well the existing laws of refinement. It is a case of atomicity refinement where a previously atomic action (event) is split into a number of steps which combined effect achieves the effect of the abstract atomic event. The Event-B approach is to introduce new variables in a refinement machine and thus have a hidden concrete state on which the steps are defined. There is a further event summarising the effect of the computations accomplished on the hidden state and explicitly relating it to the abstract state. This is the event for which the refinement relation is demonstrated while the events defining the actual computation steps would have no formal link to the abstract event.

When the hidden state does not naturally follow as a part of the modelling process this refinement style leads to a contrived model. There appear auxiliary variables and auxiliary events that play no part in the characterisation of the system behaviour but are a codification of the refinement relation to an abstract model. Since such elements accumulate during refinement this has a profound effect on the development of a large model.

8.2.3. Modes

We have conducted a study of approaches to complex critical systems development, and the requirements documents within DEPLOY, and arrived at the following:

- Separation of concerns is a major approach to tackle the complexity of the systems development.
- A large amount of critical systems are developed using a notion of operation modes.
- All critical systems involve operations with important aspects of human activity (e.g. lives, finance) hence critical. And all of them inevitably have faults due to changing environmental conditions, hardware failures etc. A high percentage of requirements to such systems (up to 40% according to our study within DEPLOY) include fault tolerance as a way to mitigate the consequences of errors.
- Requirements evolve, including FT.

There were neither mode nor fault tolerance viewpoints in the state of the art Event-B development. The UML-B approach of statecharts is closely related to modal views. However, the statecharts drive the development by generating Event-B models as opposed to the mode views which facilitate the development by leaving the Event-B modelling activity with the user. On the fault tolerance side, we are aware of the work on ProR framework for tracing requirements, and we plan to integrate the tracing framework with our modelling approach.

The Mode/FT Views approach is to assist the main Event-B development by an additional set of abstractions and a toolset to facilitate modal and fault tolerance development. We were motivated by the following stimuli:

- Facilitate the modal and fault tolerance development in Event-B with a comprehensible modelling approach.
- Stimulate the consideration of fault tolerance at the very first phases of development.

- Explicitly covering mode and fault tolerance concerns, we wanted to improve the requirement traceability and fulfilment.
- Help make planning decisions. Focusing the developer attention on specific aspects of development leads to better understanding of the problem and planning of the solution.
- Provide consistent way of stepwise development of mode and FT aspects by a notion of views refinement.

8.2.4. Qualitative Reasoning

Probability is used in many distributed systems for breaking the symmetry between different components/processes, e.g. IEEE 1394 Firewire protocol [2] , Rabin Choice Coordination algorithm [3] . For such systems, termination cannot be guaranteed for certain. Instead, a slightly weaker property is mostly appropriate: *termination with probability one*. As an example for this type of systems is to consider tossing a coin until it comes up tail. Provided that the coin is fair (in that sense that no face is ignored forever), eventually, the coin will eventually come up head.

Qualitative probabilistic reasoning has been integrated into Event-B [4] : a new kind of actions is added, namely *probabilistic actions* with an assumption that the probability for each possible alternative is bounded away from 0 or 1. Most of the time, probabilistic actions behave the same as (standard) non-deterministic actions (e.g. invariant preservation). The difference between probabilistic and non-deterministic actions is with convergence proof obligation: probabilistic actions are interpreted *angelically*, whereas non-deterministic actions are interpreted *demonically*. The result is a practical method for handling qualitative reasoning that generates proof obligations in the standard first-order logic of Event-B.

The plug-in allows developers to declare an event to be probabilistic convergent and generate appropriate proof obligations. Since the obligations are in standard first-order logic supported by the Rodin platform, we do not need to make any extension for the provers to handle the new proof obligations.

8.3. Choices/Decisions

8.3.1. Flow plug-in

The single most important feature of the new version of the Flows plug-in is the introduction of a new form of diagram structuring to prevent the appearance of large proof obligations. The typical source of such proof obligations is demonstrating that an event enables one or more events from a long list of events. Technically, the proof would state that the after-state of the event implies the disjunction of the guards of the next events. To prevent the appearance of such a disjunction a modeller is encouraged to split the list into a set of sub-models. Each sub-model has a pre- and post-conditions and there are proof obligations demonstrating that all the entry events of the sub-model are enabled when the precondition is satisfied and, symmetrically, every exit event satisfies the sub-model post-condition. Externally, a sub-model appears to be a simple atomic event.

8.3.2. Group Refinement

In this work one obvious source of inspiration is the Classical B method [5] where an abstract atomic statement may be refined into an operation which body is made of a sequence of assignments. However, the introduction of the semicolon operator in Event-B is a substantial change affecting most aspects of the method. This would also reintroduce one of the problems of the Classical B that Event-B tries to address: proof scalability. Accumulation of sequential composition through refinement steps may result in unmanageable proof obligations. It is also more difficult to conduct subsequent refinement of events with sequential actions.

Actions Systems [6] has an atomicity refinement technique where one can refine an atomic action into a loop of new actions[7] . This is general enough to address the problem but, seemingly, the associated proof cost is prohibitively high and there is no evidence that such proofs may be efficiently mechanised.

The challenge was addressed by offering a method that lets a modeller to select an alternative set of refinement laws whenever the identified pattern of refinement is encountered. The new refinement laws are based on a different interpretation of a model: split refinement (a case of event refinement when an abstract event is refined into two or more concrete alternatives) is understood as a refinement into a composite event made of the concrete events arranged in some way. One simple arrangement case is when the concrete events are understood to execute sequentially. Then the refinement relation is demonstrated for the after-state produced by executing one event after another.

The method is not limited to sequential composition and there is also a form of parallel composition. An essential property of the method is that the group refinement relation is demonstrated not just for a single arrangement of concrete events but for a whole set of traces of concrete events. There is a simple notation for the removal of undesired traces and constraining the model to specific traces. For each trace there appears one instance of action simulation proof obligation (and possibly other refinement and consistency proof obligations). Thus, for practical reasons, it is necessary to keep number of traces low. This is best accomplished by doing small refinement steps with few concrete events.

8.3.3. Modes

The main objective in the tool design was to make a simple to use environment that can be used by a non-Event-B user (e.g. requirements engineer, fault tolerance specialist), yet provides the necessary functionality for an Event-B modeller. The tool was designed to be as much an external environment to Event-B models as possible.

- We decided not to extend the Rodin database with modal and fault tolerance elements and to keep them as separate models. This led to less platform dependencies and easier maintenance.
- The static check is separated from the Rodin SC, and realized by the GMF validation since it does not logically belong to Rodin / Event-B. However, since the proofs are a part of the modelling process, we properly extended the Rodin proof obligation generator.
- A modal/FT documents form a refinement chain that mimics the Event-B refinement. This allows our tool to be used with the existing types of decomposition / modularisation.
- During the initial experiments we have identified a possible need for multiple views on a single model. The tool supports this by keeping the references to the models in the views and not the opposite.

8.3.4. Qualitative Reasoning

- Ideally, we would like to have a new value for the **convergence** attribute of Event-B events. However, this is not currently supported by the Rodin platform. Instead, a new **probabilistic** attribute is defined for events, with the value is either *standard* or *probabilistic*.
- Since standard refinement does not maintain probabilistic convergent property, we put a restriction on the development method for *almost-certain termination systems* in two steps as follows.
  1. Establish the model of the systems with various *anticipated* events
  2. While proving convergence properties of events (either standard or probabilistic), we keep the variable and event system the same (i.e. no refinement is allowed).
- For probabilistic convergence, the variant need to be bounded above. A combination of standard and probabilistic convergence events results in a probabilistic convergence system. Moreover, the use of anticipated events and refinement allowed us to construct an lexicographic variant by combining the sub-variant at different level of refinements. For probabilistic convergence system, this lexicographic variant need to be bounded above, which require all the sub-variants to be bounded above (not only those variant concern with probabilistic events).

More details of our approach is in our report at AVoCS'10 [8] .

8.4. Available Documentation

8.4.1. Flow plug-in

- An extensive example of the application of the flows in the modelling (the old version) is available to the DEPLOY Project members as a part of the Bosch cruise control case study.
- There is a wiki page describing the core proof obligations generated by the tool - Flows
- Tutorial slides available from the project file share

8.4.2. Group Refinement

- There is a wiki page[9] briefly introducing the approach and the tool
- Slides

8.4.3. Modes

- There is a wiki page[10] with details of the plug-in functionality, installation guide, and a simple example.
- Papers on modal specifications [11] and [12] , and fault tolerance [13] .
- Also, we are working on a medium-scale case study

8.4.4. Qualitative Reasoning

- Master thesis of Emre Yilmaz on developing tool support for qualitative reasoning in Event-B [14] .
- The development of Rabin's Choice Coordination Algorithm is available at the DEPLOY Repository [15] .

- A paper describing the development of Rabin Choice Coordination algorithm and tool support in the Proceedings of AVoCS'10 [8] .

8.5. Planning

8.5.1. Flow plug-in

The plan is to gather feedback from the users within the Project and encourage the external users to try out the plug-in and provide feedback and feature requests. There are many ideas on the tool extension, in particular using the tool to document event refinement (split, merge, group, refinement diagrams) and provide mechanised patterns for some of the more important cases.

8.5.2. Group Refinement

The immediate plan is to produce a technical report on the semantics of group refinement during the first quarter of 2011. Long term plans are the tool maintenance and the investigation of the possibility of more expressive form of group refinement permitting branches and loops.

8.5.3. Modes

Current status is version 1.0.0 for Rodin 2.0 In a long term we plan:

- A few usability improvements
- Integration with ProR requirements tracing framework
- Event-B model generation and editing driven by FT patterns

8.5.4. Qualitative Reasoning

In DEPLOY's fourth year, we plan to implement the missing proof obligations. More importantly, we will investigate the interaction between refinement and almost certain termination. This allows us to prove convergence properties early in the development and guarantee that refinement will maintain these convergent properties.

References

[1] A. Iliasov. Augmenting Event-B Specifications with Control Flow Information. Nodes 2010. Copenhagen June 3-4 2010, Technical University of Denmark

[2] J.-R. Abrial, D. Cansell, D. Mery. A Mechanically Proved and Incremental Development of IEEE 1394 Tree Identify Protocol. *Formal Asp. of Comput.* 14(3):215-227, 2003

[3] M. Rabin. The Choice Coordination Problem. *Acta Informatica, 17:121-134*, 1982.

[4] S. Hallerstede, T.S. Hoang. Qualitative Probabilistic Modelling in Event-B. *iFM 2007: Integrated Formal Methods*, Oxford, U.K. July 2007

[5] J. R. Abrial. The B-Book: Assigning Programs to Meanings

[6] R.J.R. Back and R.Kurki-Suonio, Decentralization of process nets with centralized control. 2nd annual symposium on principles of distributed computing, Montreal 1983

[7] R.J.R. Back. Atomicity Refinement in a Refinement Calculus Framework (Back92:atomicity)

[8] E. Yilmaz and T.S. Hoang, Development of Rabin's Choice Coordination Algorithm in Event-B. *Proceedings of AVoCS'10*, Dusseldorf, Germany, 2010 (http://deploy-eprints.ecs.soton.ac.uk/258/)

[9] http://wiki.event-b.org/index.php/Group_refinement_plugin

[10] http://wiki.event-b.org/index.php/Mode/FT_Views

[11] A.Iliasov, A.Romanovsky, F.L.Dotti. Structuring Specifications with Modes. Fourth Latin-American Symposium on Dependable Computing - LADC 09, September 1-4, 2009 (http://citeseerx.ist.psu.edu/

viewdoc/download?doi=10.1.1.148.6369&rep=rep1&type=pdf)
[12] F.L.Dotti, A.Iliasov, L.Riberiro, A.Romanovsky. Modal Systems: Specification, Refinement and Realisation. In: International Conference on Formal Engineering Methods - ICFEM 09 , December 9 -12, 2009, Rio de Janeiro, Brazil (http://deploy-eprints.ecs.soton.ac.uk/153/1/paper105-ICFEM09.pdf)
[13] I.Lopatkin, A.Iliasov, A.Romanovsky. On Fault Tolerance Reuse during Refinement. In: 2nd International Workshop on Software Engineering for Resilient Systems - SERENE 2010, April 13-16, 2010, London (http://www.cs.ncl.ac.uk/publications/trs/papers/1188.pdf)
[14] E. Yilmaz. Tool support for qualitative reasoning in Event-B. Master's thesis, Department of Computer Science, ETH Zurich, Switzerland, Aug. 2010 (http://e-collection.ethbib.ethz.ch/view/eth:1677)
[15] E. Yilmaz. Rabin's Choice Coordination Development. (http://deploy-eprints.ecs.soton.ac.uk/232/)

9 Model Animation

9.1. Overview

Most of the improvements made to model animation and model checking are related to the industrial deployment workpackages, in particular deployment within Siemens, SAP and Bosch.

9.1.1. Siemens Data Validation

STS are successfully using the B-method and have over the years acquired considerable expertise in its application. One aspect of the current development process, which is unfortunately still problematic, is the validation of properties on configuration parameters. These parameters, such as the rail network topology parameters, are only known at deployment time. Up to now, Siemens was using a custom version of Atelier B to check properties on these parameters. However, the data parameters are nowadays becoming so large (relations with thousands of tuples) that Atelier B quite often runs out of memory, even with the dedicated proof rules and with maximum memory allocated. In some of the bigger and more recent models, just substituting values for variables fails with out-of-memory conditions. The alternative consisting in the manual checking of these properties (e.g., by creating huge spreadsheets on paper for the compatibility constraints of all possible itineraries), is very costly and arguably less reliable than automated checking.

Previously, we had established the feasibility of using the ProB tool on a case study (metro of San Juan) to validate runtime data for the industrial B models, replacing a month worth of effort by a few minutes of calculation. The next step was to apply ProB to ongoing developments of Siemens and to allow Siemens to use ProB in their SIL4 development chain.

The most important contributions of the last 12 months are:

- Application of ProB in three active deployments, namely the upgrading of the Paris Metro Line 1 for driverless trains, line 4 of the São Paulo metro and line 9 of the Barcelona metro. We also briefly report on experiments on the models of the CDGVAL shuttle. The paper [1] only contained the initial San Juan case study, which was used to evaluate the potential of our approach. In this article we describe the previous method adopted by Siemens in much more detail, as well as explaining the performance issues with Atelier B.
- Comparisons and empirical evaluations with other potential approaches and alternate tools (Brama, AnimB, BZ-TT and TLC) have been conducted.
- Reports about the ongoing validation process of ProB, which is required by Siemens to allow them to use ProB instead of the existing method. The validation also lead to the discovery of errors in the English version of the Atelier B reference manual.

Also, since [1], ProB itself has been further improved inspired by the application, resulting in new optimisations in the kernel (see below).

More details about this work can be found in the following works: [2] and [3].

9.1.2. Multi-level Animation

Prior versions of ProB only supported the animation of a single refinement level. Abstract variables and predicates referring to them were ignored. To support validation of Event-B models such as the ones generated by Bosch, this short coming had to be addressed. In [4] and [5] we extended ProB in a way that all refinement levels of a model can be animated simultaneously.

First, this can give the user a deeper insight into how the model behaves and how the refinement levels are related to each other.

Second, we can now find errors in context of refinement. This include violation of the gluing invariant or not satisfiable witnesses for abstract variables. If such errors are present in a model, the corresponding proof obligation cannot be discharged. But without an animator it is not always easy to see for an user if this is caused by the complexity of the proof or by an error.

In the articles we summarized Event-B's current refinement methodology and showed for each proof obligation how the algorithm would find a counter-example. We presented empirical results and discussed how the algorithm can be combined with symmetry reduction.

9.1.3. Constraint-Based Deadlock Checking

Ensuring the absence of deadlocks is important for certain applications, in particular for Bosch's Cruise Control. We are tackling the problem of finding deadlocks via constraint solving rather than by model checking. Indeed, model checking is problematic when the out-degree is very large. In particular, quite often there can be a practically infinite number of ways to instantiate the constants of a B model. In this case, model checking will only find deadlocks for the given constants chosen.

The basic idea is to generate deadlocks by solving a constraint consisting of the axioms Ax, the invariants Inv together with a constraint D specifying a deadlock. More formally, D is the negation of the disjunction of all the guards.

The following tool developments were required to meet the challenges raised by the industrial application:

- generation of the deadlock freedom proof obligation by ProB (to avoid dependence on other plug-ins and being able to control whether theorems are to be used or not; currently they are not used)
- implementation of a constraint-based deadlock checking algorithm:
  - with the possibility to specify an additional goal predicate to restrict the deadlock search to certain scenarios: in Bosch's case due to the flow plugin, one wants to restrict deadlock checking e.g. to states with the variable Counter set to 10
  - with semantic relevance filtering (to be able to filter out guards which are always false given the goal predicate).
  - with partitioning of the constraint predicate into components and optionally reordering according to usage (basic predicates which occur in most guards are listed first)
- Improvements to ProB's constraint solving engine: (reification of constraints, detection of common sub-predicates, more precise information propagation for membership constraints, performance improvments in the typchecker and other parts of the kernel).

ProB has been applied successfully to two models of the adaptive cruise control by Bosch. The more complicated model is CrCtrl_Comb2Final. To give an idea, here are some statistics of the deadlock freedom proof obligation for CrCtrl_Comb2Final:

- when printed in 9-point Courier ASCII the formula takes 32 A4 pages (the disjunction of the guards starts at page 6)
- the model contains 59 events with 837 guards (19 of them disjunctions, some of which themselves nested)
- Bosch are interested in deadlocks that are possible according to a flow specified using the flow plugin; these can be found with ProB by specifying a goal predicate (such as "Counter=10")
- the proof obligation (as generated by the flow plugin) initially could not be loaded in Rodin due to "Java Heap Space Error".
- Counter examples are found by ProB for various versions of the model in 9-24 seconds (including loading, typechecking and deadlock PO generation; the constraint solving time is 1.03 to 12.86 seconds).

9.1.4. BMotionStudio for Industrial Models

Previously, we presented BMotion Studio, a visual editor which enables the developer of a formal model to set-up easily a domain specific visualization for discussing it with the domain expert. However, BMotion Studio had not yet reached the status of an Industrial strength tool due to the lack of important features known from modern editors.

In this work we present the improvements to BMotion Studio mainly aimed at upgrading it to an industrial strength tool and to show that we can apply the benefits of BMotion Studio for visualizing more complex models which are on the level of industrial applications. In order to reach this level the contribution of this work consists of three parts:

- We added a lot of new features to the graphical editor known from modern editors like: Copy-paste support, undo-redo support, rulers, guides and error reporting. One step towards was the redesign of the graphical editor with GEF.
- Since extensibility is a very important design principle for reaching the level of an industrial strength tool we pointed up the extensibility options of BMotion Studio.
- We introduced the visualization for two models which are on the level of industrial applications in order to demonstrate that we can apply the benefits of BMotion Studio for visualizing more complex models. The first model is a mechanical press controller and the second model is a train system which manages the crossing of trains in a certain track network.

More details can be found in [6] and [7]

9.1.5. Evaluation of the ProB Constraint Solver

Various industrial applications have shown the need for improved constraint-solving capabilities (see CBC Deadlock, Test-Case Generation). In order to evaluate ProB, and detect areas for improvement, we have studied to what extent classical constraint satisfaction problems can be conveniently expressed as B predicates, and then solved by ProB. In particular, we have studied problems such as the n-Queens problem, graph colouring, graph isomorphism detection, time tabling, Sudoku, Hanoi, magic squares, Alphametic puzzles, and several more. We have then compared the performance with respect to other tools, such as the model checker TLC for TLA+, AnimB for Event-B, and Alloy.

The experiments show that some constraint satisfaction problems can be expressed very conveniently in B and solved very effectively with ProB. For example, TLC takes 8747 seconds (2 hours 25 minuts) to solve the 9-queens problem expressed as a logical predicate; Alloy 4.1.10 with minisat takes 0.406 seconds, ProB 1.3.3 takes 0.01 seconds. For 32 queens, ProB 1.3.3 takes

0.28 seconds, while Alloy 4.1.10 with minisat takes over 4 minutes (TLC was only able to solve the n-queens problem up until n=9, or n=14 when reformulating the problem as a model checking problem rather than a constraint-solving problem). In another small experiment, we checked whether two graphs with 9 nodes of out-degree exactly one are isomorphic by checking for the existence of a permutation which preserved the graph structure. TLC finds a permutation after 2 hours 6 minutes and 28 seconds; ProB 1.3.3 takes 0.01 seconds to find the same solution, while Alloy takes 0.11 seconds with SAT4J and 0.05 seconds with minisat. For some other examples (in particular time-tabling) involving operators such as the relational image, the performance of ProB is still sub-optimal with respect to, e.g., Alloy; we plan to overcome this shortcoming in the future. Our long term goal is that B can not only be used to as a formal method for developing safety critical software, but also as a high-level constraint programming language.

9.1.6. Various other improvements

Mainly inspired by the Siemens and Bosch applications mentioned above, various improvements to the ProB kernel were undertaken.

- Improved algorithms for large sets and relations (such computing the domain of a relation), optimised support for more B operators on relations, functions, and sequences.
- Record support: automatic detection of records described by a bijection between a cartesian product and a carrier set. These axioms can either be entered manually, such as in the Bosch models of the Cruise Control, or generated automatically by the Records plug-in. In both cases, ProB detects that a record is being used, and sets the carrier set to the cartesian product and sets the bijection to the identity function.
- Detection and treatment of certain infinite sets, in particular complement sets such as INTEGER \ {x}. Such sets are being used in some of the Siemens models. Similarly, infinite identity functions are also never expanded and always treated symbolically.
- Partitioning of predicates into connected sub-components (was useful for Siemens application, to be able to pinpoint location of an inconsistency in the axioms; it turned out useful for constraint-based deadlock checking as well)
- Improved constraint solving in particular:
  - better use of Prolog's CLP(FD) constraint solver for arithmetic constraints as well as for elements of carrier sets. For example, x:{a,b,d} will attach a finite-domain constraint to x, even without enumeration.
  - improved boolean constraint solver; deals with well-definedness and propagates known boolean values through complicated predicates. Can also solve SAT problems expressed in B (up to around 600 variables and 2000 clauses).
  - Reification of constraints inside the boolean constraint solver. In particular, given x:1..10, the ProB boolean constraint solver will know that, e.g., x:{12,13,14} must be false.
  - detection of common sub-predicates inside larger formulas. This is to improve performance and overcome possible precision problems of the constraint solver. The main motivation here is deadlock checking (where the same predicate often appear multiple times, sometimes in negated form).
- General performance improvments, such as in the typchecker and other parts of the kernel when loading larger B models.

9.2. Motivations

The above works were motivated mainly to support the following three industrial deployments:

- Siemens: enable Siemens to use ProB in their SIL4 development chain, replacing Atelier B for data validation (see above).
- Bosch: provide animation and constraint-based deadlock detection for the Cruise Control. Indeed, proving absence of deadlocks is important to Bosch, as it means that the modelers have thought of every possible scenario. Currently, the proof obligation is so big (see above) that it is difficult to apply the provers and the feedback obtained during a failed proof attempt is not very useful. Using ProB to find concrete deadlock counterexample helps Bosch to find scenarios they have not yet thought about, and enables them to adapt the model. Once all cases have been covered, the proof of deadlock freedom can be done with Rodin's provers (at least that was the case for the smaller of the two models; the bigger one is still contains deadlocks and is being improved).
- SAP: provide a way to generate test cases using constraint-based animation; for more details see the description of the Model-based testing work[8] .

9.3. Choices / Decisions

For constraint-based deadlock checking we had the choice of either generating the deadlock freedom proof obligation with ProB or using ProB as a disprover on a generated proof obligation. Currently, the core of Rodin does not generate the deadlock freedom proof obligation. The flow plugin can be used to generate deadlock freedom proof obligations. The advantage, however, of generating them within ProB are the following:

- ProB knows which parts of the axioms are theorems (and can thus be ignored; they are often added for simplifying proof but can make constraint solving more difficult)
- the techniques can also be applied to classical B

For record detection we decided not to use any potential "hints" provided by the records plugin, but infer the information from the axioms. In this way, the improvement can also be applied to records generated manually (as was the case in the Bosch case study) or in a classical B setting.

9.4. Available Documentation

See the references below.

The validation document is being prepared and will probably be made available in spring 2011.

9.5. Planning

- Finish Validation Report
- Write up Constraint-Based Deadlock Checking and integrate fully into Rodin Platform
- Support mathematical extensions in ProB
- Further improvements in the constraint-solving kernel of ProB; in particular for relations and operators. A Kodkod translator is being developed.

References

[1] M. Leuschel, J. Falampin, F. Fritz, D. Plagge, Automated Property Verification for Large Scale B Models, FM'2009, LNCS 5850, Springer-Verlag, 2009

[2] M. Leuschel, J. Falampin, F. Fritz, D. Plagge, Automated Property Verification for Large Scale B Models, to appear in a special issue of FM'2009 in Formal Aspects of Computing, Springer-Verlag

[3] Leuschel et al. Draft of Validation Report

[4] S. Hallerstede, M. Leuschel, D. Plagge, Refinement-Animation for Event-B - Towards a Method of Validation, ASM'2010, LNCS 5977, Springer-Verlag, 2010

[5] S. Hallerstede, M. Leuschel, D. Plagge, Validation of Formal Models by Refinement Animation, to appear in Science of Computer Programming, Elsevier

[6] Lukas Ladenberger, Industrial Applications of BMotionStudio, Master's thesis. University of Düsseldorf. 2010

[7] Lukas Ladenberger, Jens Bendisposto, Michael Leuschel, Visualising Event-B models with B-Motion Studio. Proceedings FMICS'2009. LNCS 5825, p.202-204. 2009.

[8] http://wiki.event-b.org/index.php/D32_Model-based_testing

10 Model-based testing

10.1. Overview

Model-based testing (MBT) is an approach from software engineering that uses formal models as basis for automatic generation of test cases. A test case is defined as a sequence of actions (or events, or triggers) together with corresponding test data that can be executed against a System Under Test (SUT). There are different types of test models that can be used for MBT, many of them being state-based models (e.g. UML state diagrams). In DEPLOY, we investigate a version of MBT using Event-B models as test models. This research work provides a new feature in the Rodin platform, complementing the existing theorem proving and model-checking capabilities.

The main purpose of MBT track in DEPLOY is to provide the deployment partners an MBT method together with a Rodin plug-in that allows generation of test cases satisfying different coverage criteria (e.g. covering of all events in a model or covering paths to a set of target global states). This includes the generation of appropriate test data that satisfies the guards of the single test steps.

University of Duesseldorf (Michael Leuschel, Daniel Plagge, Jens Bendisposto) started developing first tool support for MBT for Event-B in 2009 and continuously improved the prototype based on the feedback from SAP, the main deployment partner interested in MBT. Starting with June 2010, the team of University of Pitesti (led by Florentin Ipate) joined the DEPLOY consortium when DEPLOY-Enlarged-EU officially begun. Moreover, Alin Stefanescu, who moved from the SAP team to Pitesti team in September 2010, added MBT and SAP experience to the Pitesti team.

10.2. Motivations

The interest in MBT is to get the opportunity, by using the Event-B models, not only to formally validate specifications, but also to verify using test cases, that an existing implementation behaves as expected. Along with code generation, MBT (using Event-B) operates at the lower level of the envisaged rigorous engineering chain. In DEPLOY, this chain goes from high-level requirements down to software implementations via specification, architecture and refined designs.

Deployment partners (DP), especially SAP (WP4), showed interest into having tool support for MBT. As a consequence, this topic was introduced in the refocus exercise (in the middle of the project [M24]) and was documented in the updated version DoW signed in August 2010 (see Task 9.10 there). The deployment partner SSF (WP3) had recently also shown interest in the MBT task.

For the SAP use case, MBT is applied in the area of integration and system testing for service-oriented applications. First, a method for integration testing using SAP's message choreography models was developed using ProB. In the reported period (Feb. 2010 - Jan. 2011), SAP focused on UI system testing using high-level business processes. This required an adaptation of the first MBT approach to the new model types. In these new models, the associated test data constraints played a more prominent position which required also more effort from the tooling point of view.

10.3. Choices / Decisions

For MBT using state-based models, test generation algorithms usually traverse the state space starting in an initial state and being guided by a certain coverage criteria (e.g. state coverage) collecting the execution paths in a test suite. Event-B models do not have an explicit state space, but its state space is given by value of the variables and the state is changed by the execution of events that are enabled in that state. ProB tool has a good grip of the state space, being able to explore it, visualize it, and verify various properties using model checking algorithms. Such model checking algorithms can be used to explore the state space of Event-B models using certain coverage criteria (e.g. event coverage) and thus generating test cases along the traversal. Moreover, the input data that allows to trigger the different events provides the test data associated with the test cases.

Given the above considerations, the following choices and decisions have been made:

- Using explicit model-checking: First, model-checking algorithms described in the previous paragraph were implemented and applied to message choreography models from SAP. They work fine for models with data with a small finite range. However, in case of variables with a large range (e.g. integers), the known state space explosion problem creates difficulties, since the model-checker explores the state enumerating the many possible values of the variables. This required to consider different approaches as described below.
- Using constraint solving: To avoid the state space explosion due to the large bounds of the variables, another approach ignores these values in the first step and uses the model-checker only to generate abstract test cases satisfying the coverage criteria. However, these paths may be infeasible in the concrete model due to the data constraints along the path. The solution is to represent the intermediate states of the path as existentially quantified variables. The whole path is then represented as a single predicate consisting of the guards and before-after predicates of its events. ProB's improved constraint solver (see Model Animation[1]) is then used to validate the path feasibility and find appropriate data satisfying the constraints.
- Using meta-heuristic search algorithms: As an alternative to the above constraint solving approach, we investigated also a recent approach to test data generation using meta-heuristic search algorithms (e.g. evolutionary and genetic algorithms). The idea is to solve the test data problem by starting with an initial set of data and improving it using guidance from the constraints to be satisfied (using certain fitness functions that describe "how far" is the current data from a solution). Meta-heuristic search algorithms can be applied not only to the test data generation for one path but also to obtain a set of test cases with the required coverage.

10.4. Available Documentation

Papers describing previous work:

- M. Satpathy, M. Butler, M. Leuschel, S. Ramesh. Automatic Testing from Formal Specifications. In Proc. of TAP'07, pp. 95-113, LNCS 4454, Springer, 2007.
- S. Wieczorek, V. Kozyura, A. Roth, M. Leuschel, J. Bendisposto, D. Plagge, I. Schieferdecker. Applying Model Checking to Generate Model-Based Integration Tests from Choreography Models. In Proc. of TestCom/FATES 2009, pp. 179-194, IEEE Computer Society, 2009.
- R. Lefticaru, F. Ipate. Functional Search-based Testing from State Machines. In Proc. of ICST 2008, pp. 525-528, IEEE Computer Society, 2008.
- S. Wieczorek, A. Stefanescu. Improving Testing of Enterprise Systems by Model-Based Testing on Graphical User Interfaces. In Proc. of the Satellite Workshops of ECBS'10. pp.

352-357, IEEE Computer Society, 2010.

See also: DEPLOY Deliverable D53 (August 2010).

10.5. Planning

In the fourth year of the project, the work on MBT will continue in a sustained pace in order to support the work of the deployment partners. SAP for instance has MBT as one of its main focus deployment area in the last year of the project. The tool provider Duesseldorf will continue to improve the MBT tooling by improving its constraint solver to solve the state space explosion problem. University of Pitesti will experiment with different fitness functions adapted to the concrete coverage requirements requested by the DPs. Such approaches proved to work good for numerical problems, but need adaptations to also be fully applied to data domains using sets. Finally, the tool providers may investigate the MBT requirements of SSF use case (i.e., how feasible is it to generate test cases from the Event-B models for their Bepi Colombo use case?).

References

[1] http://wiki.event-b.org/index.php/D32_Model_Animation