Alexander Romanovsky
Martyn Thomas *Editors*

Industrial Deployment of System Engineering Methods

Industrial Deployment of System Engineering Methods

Alexander Romanovsky • Martyn Thomas
Editors

Industrial Deployment of System Engineering Methods

Editors
Alexander Romanovsky
School of Computing Science
Newcastle University
Newcastle upon Tyne, UK

Martyn Thomas
Martyn Thomas Associates Ltd.
Bath, UK

ISBN 978-3-642-33169-5 ISBN 978-3-642-33170-1 (eBook)
DOI 10.1007/978-3-642-33170-1
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2013944049

ACM Computing Classification (1998): D.2, J.7, K.4

© Springer-Verlag Berlin Heidelberg 2013
This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Foreword

This book is a splendid condensed record of the DEPLOY project, epitomising a reflective and informed approach to the industrial use of formal methods in software development. The project plan was to introduce a formal method—Event-B—into a handful of industrial organisations working in different application domains; its larger aim was to learn lessons from the experience. There would be technical and managerial lessons: lessons for the industrial partners, for the academic and research partners, and for the builders of tools to support the selected method. In this larger aim the project succeeded brilliantly, and the book contains an honest and insightful account of what has been learned.

For the academic and research partners and the tool vendors, necessary improvements in the formal method and its supporting tools were identified; some have already been implemented in the course of the project. For the industrial partners, much has been learned about the value of formal methods in general and of Event-B in particular, and the match—or mismatch—with specific complexities of the systems they build and with their established development techniques. All the participants have acquired a stronger understanding of the roles of formal methods in developing dependable systems.

The industrial applications included automotive, space, railway and business systems, and even the development of an instruction set architecture for an industrial microprocessor. The varied complexities of their system functionalities, and the heterogeneous nature of their problem worlds demanded a rich variety of development processes and of concepts and techniques to be used at each development stage. A formal method cannot be the main engine of the development process. It must be applied more locally within an essentially non-formal structure. Its vital contribution is to improve system dependability by motivating formalisation where it is useful, and then by mathematically rigorous analysis and proof. Many development artifacts, formalised in a sufficiently expressive language, can be analysed to detect inconsistency, failure to satisfy known formal specifications, and other errors. Other development artifacts, most conspicuously those closely associated with the discovery, design and expression of system requirements, spring from inherently non-formal investigations and decisions, and may not admit of useful formalisation.

Their importance lies in the conceptual infrastructure they provide for other more formal artifacts.

The contributors to the book have provided an impressive wealth of detail. They report efforts on pilot projects, the results obtained, and their considered judgement of their experiences. They describe their difficulties and failures with honesty, and offer sober evaluations of their successes. The tool builders give a careful account of their responses to requests for new and improved features. Deficiencies in the method and its supporting tools are frankly discussed, as is the work that was put in hand to remedy them. Quantitative evaluations are given where appropriate, and qualitative evaluation is not spurned where it is more suitable. Managerial and organisational challenges are discussed. The software tools supporting Event-B are described, along with enhancements and extensions motivated by the needs of the industrial partners. There is a concise introduction to Event-B and its conceptual basis.

In short, this book describes a project that has made a major contribution towards bridging the gap between formalists and practitioners in software development for dependable systems. The detailed substance of the contribution lies in the specifics of what has been done; but the full value lies even more in the cooperative way in which the project has been carried out and the open-minded acknowledgement of challenges. This book will amply repay a careful and thoughtful reading by researchers and practitioners alike.

London, UK Michael Jackson
June 2012

Preface

This book is about experience gained and lessons learnt in the course of a major European project on industrial deployment of formal methods. The DEPLOY Integrated Project ran for 4 years and involved 15 partners from academia and industry. The editors came to the project from different backgrounds and with different motivations. Sascha (Alexander) Romanovsky has been working on system dependability and fault tolerance for many years and has always stressed the importance of reasoning about faults and fault tolerance at the earlier phases of system development. He coordinated the RODIN project, preceding DEPLOY, and became involved in writing the DEPLOY proposal and coordination of DEPLOY to see the tools and methods originated in RODIN further advanced and applied in wide industrial settings. Martyn Thomas is an industrialist who has been concerned with safety-critical and other high-dependence computer systems since the 1980s, and who came to DEPLOY eager to understand the barriers to greater use of science-based software engineering in industry. The reader will see in Chapter 15 what we have learnt through DEPLOY and through editing this book, and where we believe the field now stands. It is enough to say here that we have both learnt a lot, and to acknowledge that the success of DEPLOY and the insight that this book contains are the result of the talents, good-humoured collaboration and very hard work of the whole project. We therefore thank Zoe Andrews (Newcastle University), Frédéric Badeau (Systerel), Iulia Banu, Tudor Balanescu (University of Pitesti), David Basin (ETH Zurich), Nicolas Beauger (Systerel), Jens Bendiposto (University of Düsseldorf), Karim Berkani (Siemens), Emmanuel Billaud (Systerel), Pontus Bostrom (Åbo Akademi University), Jeremy Bryans (Newcastle University), Lilian Burdy (Inria), Michael Butler (University of Southampton), Mathieu Clabaut (Systerel), Kriangsak Damchoom (University of Southampton), Renaud De Landtsheer (CETIC), Fredrik Degerlund (Åbo Akademi University), Jean-Christophe Deprez (CETIC), Denisa Diaconescu, Ionut Dinca (University of Pitesti), Nicolas Dubois, Andy Edmunds (University of Southampton), Nadine Elbeshausen, Jérôme Falampin (Siemens), Yoann Fages-Tafanelli (Systerel), John Fitzgerald (Newcastle University), Fabian Fritz (University of Düsseldorf), Andreas Fuerst (ETH Zurich), Aurélien Gilles (Consultant), Rainer Gmehlich (Bosch), Radu Gramatovici (Uni-

versity of Bucharest), Katrin Grau (Bosch), Stefan Hallerstede (Aarhus University), Natasha Hebdige (Newcastle University), Thai Son Hoang (ETH Zurich), Jodi Hossbach (Newcastle University), Alexei Iliasov (Newcastle University), Florentin Ipate (University of Pitesti), Michael Jackson (Consultant), Michael Jastram (University of Düsseldorf), Cliff Jones (Newcastle University), Minh-Thang Khuu, Linas Laibinis (Åbo Akademi Univeristy), Gwenaël Le Cointre (ClearSy), Hung Le Dang (Siemens), Raluca Lefticaru (University of Pitesti), Thierry Lecomte (ClearSy), Eric Lelay (Siemens), Michael Leuschel (University of Düsseldorf), Ioana Leustean (University of Bucharest), Christophe Logerot, Ilya Lopatkin (Newcastle University), Felix Lösch (Bosch), Li Luo, Benoît Lucet (Systerel), Manuel Mazzara (Newcastle University), Larissa Meinicke, Christophe Métayer (Systerel), Arnaud Michot (CETIC), Mikael Mokrani (Siemens), Thomas Muller (Systerel), Louis Mussat (ClearSy), Mats Neovius (Åbo Akademi University), Carine Pascal (Systerel), Luigia Petre (Åbo Akademi University), Daniel Plagge (University of Düsseldorf), Marta Plaska (Åbo Akademi University), Christophe Ponsard (CETIC), Mike Poppleton (University of Southampton), Antoine Requet (ClearSy), Abdolbaghi Rezazadeh (University of Southampton), Sanae Saadaoui (CETIC), Denis Sabatier (ClearSy), Yah Said Mar (University of Southampton), Peter Sandvik, Kaisa Sere (Åbo Akademi University), Matthias Schmalz (ETH Zurich), Renato Silva (University of Southampton), Colin Snook (University of Southampton), Corinna Spermann (University of Düsseldorf), Alin Stefanescu (University of Bucharest), Anton Tarasyuk (Åbo Akademi University), Monica Tataram (University of Bucharest), Elena Troubitsyna (Åbo Akademi University), Cristina Tudose (University of Pitesti), Adrian Turcanu, Laurent Voisin (Systerel), Marina Walden (Åbo Akademi University), Jon Warwick (Newcastle University), Ingo Weigelt (University of Düsseldorf) and Sebastian Wieczorek (SAP).

Newcastle upon Tyne, UK — Alexander Romanovsky
London, UK — Martyn Thomas
May 2012

Contents

1 **Introduction** 1
Alexander Romanovsky and Martyn Thomas

2 **Integrated Project DEPLOY** 5
Alexander Romanovsky

3 **Experience of Deployment in the Automotive Industry** 13
Rainer Gmehlich and Cliff Jones

4 **Improving Railway Data Validation with ProB** 27
Jérôme Falampin, Hung Le-Dang, Michael Leuschel, Mikael Mokrani, and Daniel Plagge

5 **Deployment in the Space Sector** 45
Dubravka Ilić, Linas Laibinis, Timo Latvala, Elena Troubitsyna, and Kimmo Varpaaniemi

6 **Business Information Sector** 63
Sebastian Wieczorek, Vitaly Kozyura, Wei Wei, Andreas Roth, and Alin Stefanescu

7 **Formal Methods as an Improvement Tool** 81
Aryldo G. Russo Jr.

8 **Critical Software Technologies' Experience with Formal Methods** . . 97
Alex Hill, Jose Reis, and Paulo Carvalho

9 **Experience of Deploying Event-B in Industrial Microprocessor Development** 107
Stephen Wright and Kerstin Eder

10 **Industrial Deployment of Formal Methods: Trends and Challenges** . 123
John Fitzgerald, Juan Bicarregui, Peter Gorm Larsen, and Jim Woodcock

11 **Introducing Formal Methods into Existing Industrial Practices** . . . 145
Martyn Thomas and Alexander Romanovsky

12 Tooling 157
Michael Butler, Laurent Voisin, and Thomas Muller

13 Technology Transfer 187
David Basin and Thai Son Hoang

14 After and Outside DEPLOY: The DEPLOY Ecosystem 197
Alexander Romanovsky

15 Industrial Software Engineering and Formal Methods 203
Martyn Thomas and Alexander Romanovsky

Appendix A An Introduction to the Event-B Modelling Method 211
Thai Son Hoang

Appendix B Evidence-Based Assistance for the Adoption of Formal Methods in Industry 237
Jean-Christophe Deprez, Christophe Ponsard, and Renaud De Landtsheer

Chapter 1
Introduction

Alexander Romanovsky and Martyn Thomas

Abstract The aims of the book are explained in the context of current demands for cost-effective and dependable software and the methods that are typically employed in the software industry. The target audiences are identified and the contribution that the book could make to each audience is described. The structure of the book is described in outline.

1.1 Deployment of Formal Methods

The commercial and industrial uses of computer-based systems are growing in complexity every year, and this is causing managers and developers to look for new ways to improve productivity and dependability. Software is already pervasive in products and services that are vital to the safe, secure and efficient working of most parts of industry, commerce, health, transport and leisure. For more and more systems, it is essential that they can be developed cost-effectively and that their users can have high confidence that they will work safely and reliably. This is an engineering task.

All engineers use mathematically formal methods. They use methods that exploit well-established scientific results, embedded in mature engineering processes, because such methods allow a rigour of expression and analysis that is essential in tackling projects of industrial scale reliably and cost-effectively, and in gaining confidence that what is being built will be fit for its intended purpose. Formal methods, in this sense, are what distinguishes engineering disciplines from less professional ways of working.

Professional engineers are rightly conservative; they do not rush to adopt new methods in place of their traditional, trusted ways of working: quite reasonably they experiment on pilot projects first, and if they can, they learn from other en-

A. Romanovsky (✉)
Newcastle University, Newcastle upon Tyne, UK
e-mail: alexander.romanovsky@ncl.ac.uk

M. Thomas
Martyn Thomas Associates, London, UK
e-mail: martyn@thomas-associates.co.uk

A. Romanovsky, M. Thomas (eds.), *Industrial Deployment of System Engineering Methods*, DOI 10.1007/978-3-642-33170-1_1,
© Springer-Verlag Berlin Heidelberg 2013

gineers. For civil engineers this process has been going on for centuries, at least since Archimedes, and mechanical, electrical, and chemical engineers have all built science into their methods gradually and over more than a century.

The software industry worldwide is still immature compared with other engineering industries. The most widely applied methods and tools use little of the computer science of the past 40 years, and software contains many unnecessary errors as a result. Most of these errors cannot be corrected by testing the software and fixing the failures in the way that mechanical systems and structures can be tested and fixed, because digital systems are so complex that testing every state that could contain an error would take an impractical amount of time and resources. As the computer scientist Edsger Dijkstra remarked forty years ago, testing software can reveal the presence of bugs but never their absence.

For these reasons, new, science-based methods are increasingly important for engineers building computer-based systems. These methods offer high productivity *and* high dependability by reducing the opportunity for introducing errors and by automating most of the task of finding the residual errors and showing that the design is correct. The *DEPLOY* project on *Industrial Deployment of System Engineering Methods Providing High Dependability and Productivity* (http://www.deploy-project.eu/) set out to collect the experience of introducing formal methods into several very different application domains and to make that experience available as widely as possible.

This book is the result. It is a book of experience, written for

- technical leaders in industry who may be thinking about the possible introduction of formal methods;
- early and mid-career professionals who may need to assess the importance of these methods for their future careers, and
- system and software engineers developing important systems.

We hope that the book will also prove valuable to

- standards makers and regulators;
- academics who can make use of the examples and experience for teaching purposes;
- undergraduate and postgraduate students, for understanding the industrial context for the methods they are studying, and
- the developers of tools and methods who may not have experience in the practical issues that determine whether their work will be usable in a real commercial or industrial environment.

1.2 Book Structure

This is not a tutorial on any particular method, although Event-B was widely used in the DEPLOY project and we have included a description of and introduction to it in Appendix A. Inevitably, much of the experience that we report comes from the

use of Event-B and the Rodin toolset, but we have sought to make the conclusions as independent of particular methods as possible so that the lessons from DEPLOY are widely applicable. We see this as the start of a process that will continue to accumulate and disseminate experience; how the reader can contribute and where s/he can find further evidence is explained in Appendix B.

The structure of the book is as follows:

Chapter 2 describes the aims and approach of DEPLOY, to show the scale of the work on which our experience and conclusions are based and to provide the context for later chapters;

Chapters 3–9 describe the experiences of introducing or extending the use of formal methods in particular application domains (electronic commerce, satellite systems, rail transportation, automotive, microprocessor design and other safety-related domains);

Chapter 10 describes the results of a large survey of the use of formal methods in industry and compares our industrial deployment projects with experience acquired elsewhere;

Chapter 11 explores the issues that arise when increasingly formal methods are introduced into industrial companies operating within their own context of existing methods and tool chains, regulatory requirements, policies for reuse and intellectual property, and other practical considerations;

Chapter 12 shows the sorts of issues that arise when new methods are first used on real, industrial projects from the perspective of the tool developers. It describes the enhancements that have to be made to the methods and the supporting toolset so that they have the power to support teams of engineers with very particular needs and constraints;

Chapter 13 explains how we addressed training and technology transfer more generally, and explains how and why we would do things differently now;

Chapter 14 looks at the ecosystem that is required before an industrial company can adopt new methods as the basis for product development and support, possibly over decades of service lifetime, and describes what has been created to provide continuing support for the growing community of companies using Event-B and Rodin;

Chapter 15 summarises what we have learnt and draws some conclusions;

Appendix A provides a description of and a brief tutorial on Event-B to facilitate understanding of the examples that occur in several of the chapters;

Appendix B describes our online evidence repository, the way in which evidence was collected and how readers can contribute their own experiences. It provides examples of Case Studies and Frequently Asked Questions.

Acknowledgements We are grateful to Ronan Nugent from Springer for supporting the idea of this book and helping us with its publication, and to Alexei Iliasov for helping us in dealing with LaTeX formatting.

Chapter 2
Integrated Project DEPLOY

Alexander Romanovsky

Abstract This chapter introduces the four-year DEPLOY (Industrial Deployment of Advanced System Engineering Methods for High Productivity and Dependability) project to overview the context in which industrial deployment was conducted and explain how it was organised and managed.

2.1 DEPLOY Objectives, Consortium and Outcomes

The overall aim of the FP7 (EC Seventh Framework Programme) Integrated Project DEPLOY, run between February 2008 and April 2012 [6], was to make major advances in engineering methods for dependable systems through the deployment of formal engineering methods. The work was driven by the tasks of achieving and evaluating the industrial take-up of the DEPLOY methods and tools, initially in the four sectors which are key to European industry and society.

Four leading European companies representing four major sectors: transportation (Siemens Transportation Systems), automotive (Bosch), space (Space Systems Finland) and business information (SAP AG) worked in DEPLOY on deploying advanced engineering approaches to further strengthen their development processes and thus improve competitiveness.

The overall aim of DEPLOY was achieved with a coherent integration of scientific research, technology development and industrial deployment of the technology. The complementary expertise and technological bases of the industrial deployment partners and the technology provider partners were combined to achieve a set of challenging scientific and technological objectives.

DEPLOY offered a balanced interplay between industrial deployment, scientific research and tool development, where companies in four sectors joined their forces with ten technology providers to meet the goal.

The industrial sectors (transportation, automotive, space and business information) comprised a palette of important European base industries of today. Before entering the project the companies possessed different maturity levels when it came to deploying formal approaches.

A. Romanovsky (✉)
Newcastle University, Newcastle upon Tyne, UK
e-mail: alexander.romanovsky@ncl.ac.uk

A. Romanovsky, M. Thomas (eds.), *Industrial Deployment of System Engineering Methods*, DOI 10.1007/978-3-642-33170-1_2,
© Springer-Verlag Berlin Heidelberg 2013

The seven academic partners (Newcastle University, Åbo Akademi University, ETH Zurich, University of Düsseldorf, University of Southampton, University of Bucharest and University of Pitesti) are world leaders in formal methods research, with considerable experience in developing and applying dependability methods as well as a wide range of formal approaches.

The tool vendors, Systerel and ClearSy, have long-standing experience in developing tool support for formal engineering methods. CETIC has considerable experience in industrial quality measurement and was in charge of the assessment and evidence collection and analysis activities.

DEPLOY was set to deliver methods and tools that

- support the rigorous engineering of complex resilient systems from high-level requirements down to software implementations via specification, architecture and detailed design;
- support the systematic reuse and adaptation of models and software, thus addressing the industry's requirement for high productivity and requirements evolution;
- have been field-tested in and adapted for a range of industrial engineering processes;
- are accompanied by deployment strategies for a range of industrial sectors;
- are based on an open platform (Eclipse) and are themselves open.

By the end of DEPLOY, each industrial partner planned to achieve real deployment of formal engineering methods and tools in the development of products and to become self-sufficient in the use of formal engineering methods. The project plan focused on deployment that would enable the consortium to provide scientifically valuable artefacts (including formally developed dependable systems) and results of systems analysis (including a rich repository of models, proofs and other analysis results).

The description of work aimed to extend the mathematical foundations of formal methods in order to deliver research advances in complex systems engineering methods that enable high degrees of reuse and dependability and ensure effective systems evolution that maintains dependability. The DEPLOY work was planned with the aim of developing a professional open development platform based on Eclipse [7] that provides powerful modelling and analysis capabilities, is highly usable by practising engineers and is tailored to sector-specific engineering needs. It was in the project plan to use the experience and insights gained in the industrial deployments of DEPLOY to identify and report on the strategies that enable the integration of formal methods and tools with existing sector-specific development processes.

The consortium planned to put in place an organisation which would be the home of the open platform and which would serve as a body of industrial users and technology providers whose role would be to coordinate technical decisions on the platform and deliver training material covering general and sector-specific formal engineering methods. Åbo Akademi (Kaisa Sere, Elena Troubitsyna), ETH Zurich (Jean-Raymond Abrial) and the University of Southampton (Michael Butler) came together to work on FP6 STREP (Strategic Targeted Research Project) RODIN, Rigorous Open Development Environment for Complex Systems (2004–2007) [12], with the goal of creating a methodology and supporting open tool plat-

form for the cost-effective rigorous development of dependable complex software systems and services. This project mainly focused on building a methodology and tool support for the Event-B Method [2], which was created at about the same time by Jean-Raymond Abrial.

While the B Method [1], developed by Abrial in the early 1990s, is focused on supporting formal development of software, Event-B broadens the perspective to cover systems. Rather than just modelling software components, Event-B is intended for modelling, analysing and reasoning about systems that may consist of physical components, electronics and software. An essential difference between Event-B and the B Method is that Event-B allows a richer notion of refinement in which new observables may be introduced in refinement steps. This means that complex interactions between subcomponents may be abstracted away in modelling at an early stage and then incrementally introduced through refinement.

The RODIN project was extremely successful as it researched and developed advanced engineering methods and tools that were extensively validated and assessed through industrial case studies from various domains, which paved the way for the technology to be deployed. In particular, RODIN delivered an extensible open source platform (called Rodin), based on Eclipse, for refinement-based formal methods along with a body of work on formal methods for dependable systems. DEPLOY has exploited and built on these results.

DEPLOY used the Event-B formal method as a basis but also explored various extensions and other formal approaches where appropriate. There were several reasons for choosing Event-B, and the success of the RODIN project was one of them. This project produced promising research results well received by the scientific community, and an open tool environment built on innovative principles and appreciated by the RODIN industrial partners and the project industry interest group. RODIN demonstrated the need for formal modelling at the system level. The RODIN team worked very well together and formed the core of the DEPLOY project consortium. It was very important for the consortium to have the creator of Event-B, Jean-Raymond Abrial, on board. It should be noted, however, that the consortium did not plan or have the resources to use multiple formal methods as a basis for its work.

The results of RODIN prompted several companies, which later became part of the DEPLOY consortium, to become interested in formal modelling at the system level, and in particular in Event-B and the Rodin platform. In addition to the academic partners well known for their research in formal methods, two French companies developing tools for B and Event-B joined DEPLOY. These formed the core of the DEPLOY consortium.

2.2 Event-B

The B Method [1] is a state-based formal approach that promotes the correct-by-construction development paradigm and formal verification by theorem proving.

The Event-B formalism [2] is a specialisation of the B Method. It enables modelling of event-based (reactive) systems by incorporating the ideas of the Action Systems formalism [3] into the B Method. In Event-B, a system specification (model) is defined using the notion of an abstract state machine. An abstract state machine encapsulates the model state, represented as a collection of model variables, and defines operations on it. Therefore, it describes the dynamic part (behaviour) of the modelled system. Usually a machine also has an accompanying component, called context, which contains the static part of the model. In particular, a context can include user-defined carrier sets, constants and their properties, which are given as a list of model axioms. Event-B employs a top-down refinement-based approach to system development. Development starts from an abstract system specification that models the most essential functional requirements. While capturing more detailed requirements, each refinement step typically introduces new events and variables into the abstract specification. These new events correspond to the steps that are not visible at the abstract level. The variables of a more abstract model in the refinement chain are called the abstract variables, whereas the variables of the next refined model are called the concrete variables. Event-B formal development supports data refinement, allowing us to replace some abstract variables with their concrete counterparts. In that case, the invariant of the refined machine formally defines the relationship between the abstract and concrete variables. To verify correctness of a refinement step, one needs to prove a number of proof obligations for the refined model. Intuitively, those proof obligations allow us to demonstrate that the refined machine does not introduce new observable behaviour, or more specifically, that concrete states are linked to the abstract ones via the given (gluing) invariant of the refined model. In general, these proofs guarantee that the concrete model adheres to the abstract one, and thus all proved properties of the abstract model are automatically inherited by the refined one. Appendix A provides a full introduction to Event-B.

2.3 DEPLOY Implementation

The work on DEPLOY had a cyclic nature and was conducted in two phases: pilot deployment and full deployment. The first phase started with an initial transfer of the technology developed during the RODIN project and intensive training of the deployment partners' engineers (in particular, we ran a three-day block course for all deployment partners' engineers). The pilot deployment was started in parallel; it consisted in the formal development of small- to medium-size systems typical of the application domains of the deployment partners. This allowed the consortium to assess the domain-specific deployment issues and to feed them back to the methodological and tooling work. The first phase was successfully completed after 1.5 years. The implementation plan included a project refocus at this point.

During the refocusing stage, the project Board analysed the major methodological and tooling needs identified in the pilot phase. In addition to this, an improved

understanding of the DEPLOY methods and tools prompted the deployment partners to adjust their priorities and to clearly identify new needs. This resulted, in particular, in the creation of three new strands of work: code generation, model-based testing, and modelling and analysis of real-time systems.

During the second phase a full deployment was conducted in parallel with the improvement of the DEPLOY methods and tools. Regular meetings were held to insure that the feedback from deployment quickly came to the attention of the technology developers. The project assigned an experienced academic, who worked very closely with the deployment engineers, to each deployment partner; in some situations even becoming part of their team, s/he was responsible for reporting all deployment-related issues identified in the partner's work to the project method and tool development teams.

At the core of the DEPLOY project implementation was a triangle with industrial deployment, methodological and tooling work as the three corners. These three elements were always closely connected to and influenced each other, so that the needs of deployment drove the development of methods and tools, which in their turn were fed into the industrial deployment. The importance of the link (and the tension) between tools and methods was realised very early in the project; this allowed the consortium to find the right balance between advancing methods that could be supported with tools and those that could not, focusing more on advancing the tool-supported ones.

Building on the success of project dissemination and exploitation at phase one, the Board identified new opportunities for working with external users and developers. During the refocusing stage, we introduced the mechanism of DEPLOY Associates in order to allow selected companies interested in applying DEPLOY tools and methods in their settings to work together with the project partners. The aim was to gain more experience in deploying DEPLOY results in new application domains and for new types of applications. As the project supported only training and exchange visits, DEPLOY Associates significantly contributed to this work. The three companies (XMOS, Grupo AeS, Critical Software Technologies) that became DEPLOY Associates provided valuable feedback to the project work on methods and tools, and helped the project demonstrate wider applicability of its results. The work of the DEPLOY Associates is reported in Chaps. 7–9.

During the third year of the project, two new partners (University of Pitesti and University of Bucharest) joined the consortium, supported by a special grant available as part of FP7. This was intended to allow the ongoing FP7 projects to be extended by integrating new partners from the enlarged EC. The project Board added the two new academic partners in recognition of their concrete plans to closely work on methods and tool deployment with one of the deployment partners (SAP), and their excellence in research.

The initial plan was to work on measuring the impact of formal methods deployment, by focusing on defining metrics and collecting data showing quantitative improvements. During the first phase of the project, the consortium realised the importance (especially for the industrial partners) of qualitative measurement and collection of evidence. At the refocusing stage, a shift was made to gathering evidence

that would help industrial organisations decide whether to adopt formal engineering methodologies, and to what extent. We created a methodology for collecting and structuring the evidence collected during the work of the deployment partners and DEPLOY Associates. Appendix B describes how CETIC created an evidence repository as the main mechanism for collecting and structuring evidence in order to make this information available to the various stakeholders (top-level managers of the industrial organisations, industrial managers working on specific projects and products, industrial engineers, academics, etc.).

As part of the project, we worked on building an ecosystem of people and organisations engaged with the tools and methods developed in the project by advancing and extending these tools and methods, deploying them in industrial settings and training engineers in using them. This ecosystem included universities that incorporated the methodological and tooling materials developed in the project in their courses. Over the lifetime of DEPLOY, it consisted of the project team (about 100 people from 14 organisations were involved in the project during its four-year work), the DEPLOY Associates and the DEPLOY Interest Group. The Interest Group consisted of more than 70 organisations and individuals that received regular updates on our work and took part in our public events, such as Rodin developer and user workshops and Industry days.

In the final year, the project worked not only on full deployment, finalisation of the tools, preparation of the documentations, and summarising of the project results and the lessons learnt, but also on building DEPLOY legacy and ensuring that its ecosystem will be operational and active after the project ends.

2.4 Legacy and Results

The DEPLOY project successfully achieved its main goal: it made major, substantial advances in developing advanced engineering methods for constructing dependable systems. This was done in response to feedback from industrial deployment of formal engineering methods on realistic problems.

The project legacy includes the main DEPLOY web site [6]; this will be maintained after the project end, but no new information will be added. The site includes all public project deliverables and newsletters [5].

The home of Event-B and the Rodin platform at [11] will be actively used after the project end. All participants involved in tool development will use it for dissemination of their results; this includes activities conducted in various public (both national and European) and industrial projects. This site allows free downloads of all tools and plug-ins [13] and the up-to-date Event-B and Rodin documentation wiki [10]. The Rodin handbook developed in DEPLOY is made publicly available at [14]. The Rodin platform development will continue at SourceForge [9].

All DEPLOY publications, reports, tutorials, training materials, presentations, papers, and models can be freely downloaded from [4]. This site will be used and maintained by the follow-up FP7 ADVANCE STREP on Advanced Design and Verification Environment for Cyber-physical System Engineering [8].

The open repository of evidence for adopting formal methods in industry created by the DEPLOY team is made available at http://www.fm4industry.org. It will be maintained in the foreseeable future, to be used by a wider community for collecting evidence on formal method application in and impact on industry.

As part of the project we created a not-for-profit company called Rodin Tools Ltd., which will take over the responsibility for the Rodin toolset at the end of DEPLOY (see http://www.rodintools.org). The company consists of

- a Strategy Committee of external advisers responsible for the development strategy,
- a Platform Development and Maintenance partner to carry out the wishes of the Strategy Committee and Company members, and
- a Coordination partner to manage the Company, run workshops and training, etc.

The main outcomes of the project include industrial deployment of advanced systems engineering methods at SAP, Bosch, Siemens Transportation and Space Systems Finland. In the course of the project, each deployment partner became self-sufficient in using these methods. DEPLOY developed an extensive set of scientifically valuable artefacts, including models, theories, methods, architectures, patterns and tools, which were thoroughly assessed during industrial deployment. The DEPLOY team made substantial research advances in complex systems engineering methods. We developed a professional industry-strength open development platform based on Eclipse (the Rodin platform), as well as a large number of high-quality training materials, courses and tutorials. We developed strategies for integration of formal methods and tools with existing sector-specific development processes. One of our outcomes is an organisation which will be the home of the open platform (Rodin Tools Ltd.). We ensured that the results of the project would be widely used and extended after the project end by building an ecosystem that comprises a substantial number of industrial users and technology providers (including the members of the project consortium, DEPLOY Associates and members of the DEPLOY Interest Group).

Acknowledgements I would like to express my deepest gratitude to everybody who has been involved in the preparation and implementation of the project since its idea was first discussed with Cliff Jones in April 2006. Cliff helped me enormously in negotiating and establishing DEPLOY. Martyn Thomas, who has chaired the project Board, was instrumental in steering the consortium towards its successful completion. Jon Warwick, the project Manager, has been with the team since the very first day of our work on the proposal, shielding me from dealings with various financial, legal and funding authorities.

References

1. Abrial, J.-R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, Cambridge (1996)
2. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)

3. Back, R., Sere, K.: Superposition refinement of reactive systems. Form. Asp. Comput. **8**(3), 1–23 (1996)
4. DEPLOY e-prints. http://deploy-eprints.ecs.soton.ac.uk/
5. DEPLOY project deliverables and newsletters. http://www.deploy-project.eu/html/deliverables.html
6. DEPLOY project web site. http://www.deploy-project.eu/
7. Eclipse modeling framework. http://www.eclipse.org/emf
8. FP7 ADVANCE project. http://www.advance-ict.eu/
9. Rodin development repository. http://sourceforge.net/projects/rodin-b-sharp/
10. Rodin documentation wiki. http://wiki.event-b.org/index.php/Main_Page
11. Rodin platform. http://www.event-b.org/
12. RODIN project web site. http://rodin.cs.ncl.ac.uk/
13. Rodin tools and plug-ins. http://wiki.event-b.org/index.php/Rodin_Plug-ins
14. The Rodin Handbook. http://handbook.event-b.org/

Chapter 3
Experience of Deployment in the Automotive Industry

Rainer Gmehlich and Cliff Jones

Abstract This chapter sets out the experience of deployment in the automotive components company Bosch (Robert Bosch GmbH). An analysis of the typical challenges and practices is followed by a detailed description of the process used to experiment with the adoption of more formal methods by Bosch Research. One conclusion is that there is a need for semi-formal methods for bridging the gap between the initial (natural language) requirements and the creation of a formal model in Event-B. It is also important to note that the process of development reveals differences between refinement as used in the Problem Frames Approach and that envisaged in Event-B. Finally, the experience gained by the main support contact (Newcastle University) is analysed in the hope that these lessons will assist future projects.

3.1 Automotive Software and Typical Development Practice

It is important to understand that automotive applications are rarely "green field" developments; in fact, most are designed as embedded systems to implement a small number (possibly only one) of dedicated functions. Also, in this domain one often needs to deal with a large range of system variants, so it is necessary to find a way of describing and proving system variants effectively in any formal notation that might be helpful. Furthermore, the coordination and prioritisation of requests is an important task in such applications.

It should also come as no surprise that typical software applications within the automotive environment must never behave in a way that can endanger the driver, the vehicle or other vehicles nearby. For this reason, safety requirements are defined [4] and, up to now, the practice has been to test rigorously (with a considerable expendi-

R. Gmehlich (✉)
Robert Bosch GmbH, Schwieberdingen, Germany
e-mail: Rainer.Gmehlich@de.bosch.com

C. Jones
Newcastle University, Newcastle upon Tyne, UK
e-mail: cliff.jones@newcastle.ac.uk

A. Romanovsky, M. Thomas (eds.), *Industrial Deployment of System Engineering Methods*, DOI 10.1007/978-3-642-33170-1_3,
© Springer-Verlag Berlin Heidelberg 2013

ture of effort). Reacting as quickly as possible to either driver requests or changing environmental conditions is another important requirement to most applications implemented in the automotive environment. Thus, an appropriate way of modelling time is important when applying formal methods to such systems.

The current development process broadly follows a tailored version of the "v-model" [9]. This needs to be supplemented by the ability to adjust system parameters which is a large and time-consuming task.

3.2 Plan

It was never the expectation that the DEPLOY project would result in a complete overhaul of development processes at Bosch. More specifically, our objectives in this work [8] were to

- provide evidence that refinement-based formal engineering methods are applicable to Bosch systems. The key priorities for Bosch are to
  - structure development of system requirements and support systematic construction and validation of formal models from requirements,
  - make effective reuse and evolution of formal models and analysis,
  - provide evidence of the applicability of formal methods to the development of automotive systems;
- develop a specific methodology for automotive systems and provide evidence for applicability by close-to-production implementation of relevant parts of the pilot application;
- identify changes to the current development process as well as concepts for assimilation.

The Bosch project team consists of three full-time engineers, with some experience with formal methods in general but no experience with Event-B in particular.

In order to achieve the objectives above, the following deployment strategy was pursued:

- "Block course". This provided an initial exposure to the Event-B method and an introduction to Rodin tools.
- "Mini-pilot". The mini-pilot constructed a small Event-B model focused on specific aspects (in the case of Bosch, modelling of continuous behaviour and time).
- Pilot. The goal of the pilot was to develop a specific methodology for automotive systems, including an industrial process for formal development (necessary for large-scale deployment), as well as providing evidence for sector acceptance (by developing a close-to-production implementation of the relevant parts of the cruise control system).
- Enhanced deployment. This will result in the application of the methodology in the context of another domain which has different characteristics.

The following subsections elaborate the main phases.

3.2.1 *Mini-Pilot*

One objective of the mini-pilots was of course to increase confidence in the use of the formal notation and provide experience with the Rodin tools.

Applications realised in the automotive environment tend to be complex and distributed between hardware and software. Thus, the idea of the mini-pilot was to capture a manageable, yet typical, element of the pilot application to demonstrate the concepts and functional range of Event-B. With switches and buttons being typical elements of the interface between the cruise control system and the driver, we started with a simple on/off switch as well as a button. Later, a three-way and an n-way switch were added. Apart from being simple, yet typical, elements of a cruise control system, the modelling of switches and buttons requires a simple time model. This time aspect is important in virtually all automotive applications.

An on/off switch is characterised by its input, its output, and—in our case—four additional parameters: debounce time BT, cycle time CT, rising threshold RTH, and falling threshold FTH. From the rising and falling threshold, the state of the switch (on/off) is inferred. Debouncing is a mechanism applied to filter interfering signals or handle stress peaks. The input (in) is a continuous signal between 0 and 1. The output (out) is a digital signal with the values on and off. The input will be read cyclically with cycle time CT. Typical values are 100 ms for BT, 10 ms for CT, 0.9 for RTH, and 0.1 for FTH. The behaviour of the switch is specified as follows.

- Initially, the output is off. If the output is off and the switch on condition is true, the output is set to on.
- If the output is on and the switch off condition is true, the output is set to off.
- A rising edge is detected if at time t the input is higher than RTH and at time t–CT it was lower than RTH.
- A falling edge is detected if at time t the input is lower than FTH and at time t–CT it was higher than FTH.
- The switch on condition is true if a rising edge was detected and the input exceeds RTH for BT after the rising edge.
- The switch off condition is true if a falling edge was detected and the input is lower than FTH for BT after the falling edge.
- $BT > CT$.
- $0 \leq FTH < RTH \leq 1$.

Figures 3.1 and 3.2 give an idea of the behaviour of the on/off switch. Figure 3.1 shows an ideal signal flow with a stable input signal. In reality, however, the input signal needs a certain amount of time to stabilise after the switch has been pressed, as shown in the signal flow after the rising edge has been detected in Fig. 3.2. We also have to deal with interfering signals that can influence the input signal, as depicted after the output signal is set to "on" in Fig. 3.2.

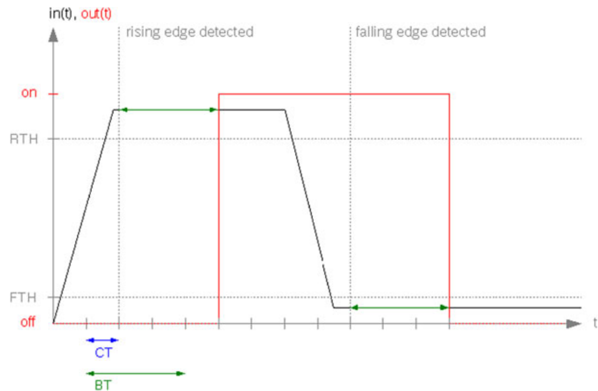


Fig. 3.1 Switch signal flow under ideal conditions

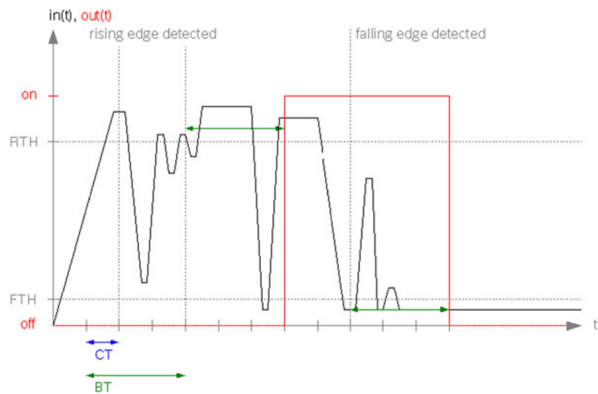


Fig. 3.2 Switch signal flow under realistic conditions

3.2.2 Pilot Deployment

For the pilot deployment [1], we chose a "cruise control" system, an automotive system implemented in software which can automatically control the speed of a car. The cruise control system is part of the engine control software which, in turn, controls engine actuators (e.g. injectors, fuel pumps, throttle valves) based on the values of specific sensors (e.g. the gas pedal position sensor, air flow sensor, lambda sensor). Since the cruise control system automatically controls car speed, there are safety aspects to be considered and it needs to fulfil a number of safety properties. For example, the cruise control system must be deactivated upon certain actions of the driver or in case of a system fault.

3.2.3 Enhanced Deployment

For the enhanced deployment [2] in the automotive sector we chose a "start-stop" system (SSE). The SSE is a system that helps save fuel and reduce emissions by

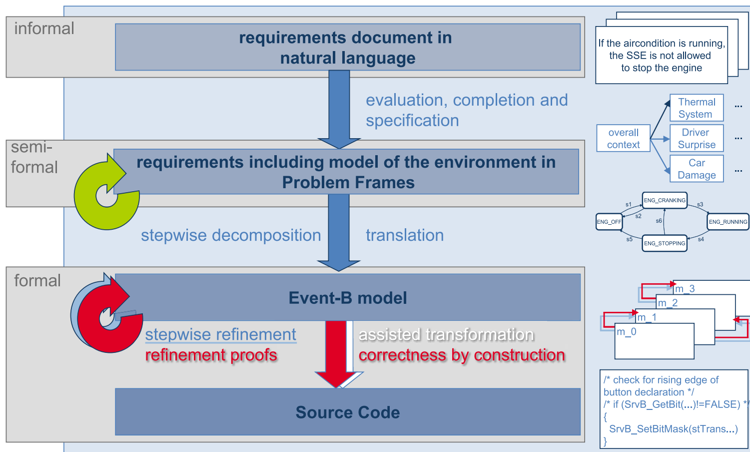


Fig. 3.3 Development process

turning the combustion engine off when it is not needed and turning it on again as soon as it is required. A typical scenario is when stopping at a red light: usually the engine would keep running (consuming fuel and producing emissions) although it is not needed until the light changes to green and the driver continues the journey. With the SSE, if the driver stops at a red light, changes the gear lever position to neutral and releases the clutch, the engine is turned off; as soon as the driver presses the clutch again to engage a gear, the SSE starts the engine and the driver can move forward as normal. It is, however, not only the driver who influences the SSE. In the situation described above for example, the SSE will not stop the engine if the charge in the battery is below a certain threshold. Furthermore, there are a number of crucial safety considerations involved in starting an engine automatically.

3.3 Pilot Deployment and Experiences

The original idea—before having experience with refinement-based development with Event-B—was to put some initial effort into amending the requirements before starting with an Event-B model. We decided very early that this approach was not feasible for several reasons. First of all, the requirements we obtained from the traditional requirements engineering process were not the optimal starting point for formal modelling. The gap between requirements in natural language and a formal model was simply too big. From our point of view, if this one-step approach is used, the validation problem does not have an efficient solution. Such a development process might work in small-scale applications but not in industrial-size ones.

We therefore developed an approach to overcome this problem. This uses a semi-formal requirements engineering method with several extensions to bridge the gap between informal specification and Event-B (see Fig. 3.3). Using this approach, we

were able to divide the big validation problem into two smaller ones. As a side effect, we also improved the quality of the requirements document.

3.3.1 Experience

We gathered considerable experience concerning methods and tools from the pilot deployment. This section will draw conclusions from this.

3.3.2 Requirements Engineering Using Problem Frames

A well-developed and structured requirements document which allows us to identify different abstraction levels is key to successful formal modelling with a refinement-based method such as Event-B. Furthermore, the method used for requirements engineering needs to provide means for handling large and complex systems by decomposing a problem into smaller parts and later recomposing them. Finally, it should support a semi- or pre-formal notation of requirements to make the step from informal requirements to a formal model manageable. By using Problem Frames [5], we achieved a clear separation of the requirements, the environment and the system to be built. Furthermore, the requirements are more precise than the informal ones, which is an important step forward towards a formal specification.

Figure 3.4 shows the quantitative results of applying our requirements engineering method to the cruise control system. The total effort in working hours is shown on the x-axis. Four curves are plotted on the y-axis. The first curve shows the total number of requirements as a reference. This curve changes over time due to the addition of new requirements (shown in a separate curve) and the rejection of original requirements (shown in the third curve).

In total we ended up with nearly half the number of text units (requirements) needed to describe the required functionality of the cruise control system. Thus, by applying our requirements engineering method, we were able to reduce the total number of text units (requirements) by more than 40 %. The total effort on the restructuring and improvement of the requirements amounts to approximately 300 working hours.

3.3.3 Mapping Problem Frames to Event-B

With semiformal requirements in Problem Frames, the mapping to Event-B is made very promising by the similarity between the notion of refinement in Event-B and elaboration in the Problem Frames Approach.

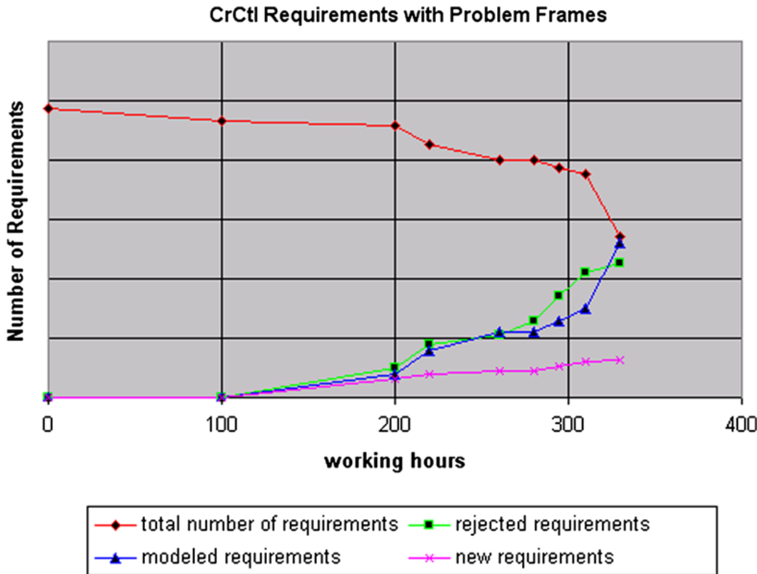


Fig. 3.4 Quantitative results—requirements

In Problem Frames the use of abstract functions is quite standard. It is easy to state that two or more phenomena are in some relationship or influence each other. In Event-B modelling, expressing this kind of function is cumbersome because it complicates refinement to more concrete models and makes proofs difficult. Therefore, we decided not to use abstract functions for modelling such relations. Instead, we used nondeterministic assignments, knowing that they have weaker expressiveness than abstract functions.

3.3.4 Requirements Traceability

With the translation of semiformal requirements into a formal model there is the challenge of traceability between requirements in Problem Frames and their implementation in Event-B. Some of the requirements in Problem Frames might be modelled as invariants, others might be guards of events or actions. Some of them might occur not only at a specific point (such as an event) but could also spread throughout the entire model. A requirement in Problem Frames describes how under certain conditions a domain should be influenced by the machine which makes the mentioned possibility of requirements spreading in the Event-B model more likely.

However, rather than using completely informal requirements, ordering and structuring requirements at different levels of abstraction in a semiformal notation helps achieve traceability.

3.3.5 Modelling of Control Systems in Event-B

Although modelling in Event-B is usually straightforward, there are some challenges we encountered when modelling a control system. There is a general idea that a change of the signals in the environment should lead to a reaction by the controller, which will affect the environment with actors. For this we need some kind of ordering of events. A simple nondeterministic choice of events is not sufficient for this kind of problem.

In order to generate C-Code from the Event-B model, it is required that we have some kind of support for scheduling of events as well as a way to distinguish between the part of the Event-B model which represents the environment and that representing the actual machine.

3.3.6 Refinement Strategy and Decomposition

The refinement strategy in Problem Frames is not necessarily the same as in Event-B, but the Problem Frames refinements do give some indication of what needs to be done. This can be quite helpful. It is difficult to decide on a refinement strategy before the actual work on modelling is performed.

The possibility of decomposing the model in Event-B is especially important if the model is large and more than one person has to work on it. The two decomposition styles (shared variable style and shared event style) are not suitable for our model. We chose shared variable style decomposition, but if a variable is shared, it cannot be refined further. Although there are obvious difficulties in providing support that synchronises the work of engineers working on parts of a decomposed model, we do require support for it in order to handle the design of an application such as the cruise control system.

3.4 Enhanced Deployment and Changes Prompted by the Lessons Learned

To develop an enhanced deployment strategy which included the choice of the pilot itself, the development process to be used, the modelling style and verification goals, we meticulously analysed the experience of the pilot deployment and adjusted our strategy. Below are described the various changes and experiences involved in application, method/process and tools.

3.4.1 Application

The application for the pilot deployment was the cruise control, which is an embedded real-time system with a closed loop controller as an essential part. Having

learnt important lessons from the cruise control deployment, we chose the SSE as our second pilot application for the following reasons.

First of all, the SSE does not include a closed loop controller. During the pilot deployment we decided to separate the development and modelling of the closed loop controller from the Event-B modelling/development process. This was done because it is obvious that Event-B does not support the development and/or modelling of closed loop controllers. The separation between the discrete and the continuous parts (closed loop controller) was successful; nevertheless, it complicated the requirement engineering (in which the separation was done) and, later on, the synthesis of the two development strands. This separation was done in an *ad hoc* manner, which suggests that, if deployed in industry, a deeper investigation is needed into how this could or should be done and thereafter a more mature method and tool support developed.

The second difference between the cruise control and the SSE is the size. We underestimated the effort required at the requirement engineering, modelling and proving stages of cruise control system deployment. For this reason, and due to the limited time for the enhanced deployment, we chose a smaller application. In addition, apart from the limited resources, the size itself introduces problems into Event-B modelling and verification (see the tools and method Sects. 3.4.2–3.4.4).

To sum up, there are similarities between the two applications, as both

- have an extensive interface with other parts of the engine control
- calculate the driver demands from direct and indirect information
- contain a state machine with a moderate number of states but complicated conditions
- display information to the driver about the current state,

but there are also differences:

- SSE is smaller than the cruise control
- the main output of the cruise control is a torque demand (calculated by a closed loop controller), unlike two boolean variables output by SSE
- SSE contains no closed loop controller.

3.4.2 Method/Development Process

In the pilot deployment, extended Problem Frames [1] were used for analysing the system to produce detailed requirements and a design specification. Following the requirements engineering phase, we started modelling the system in Event-B, preserving the structure we introduced during the requirements engineering step. This was done to achieve direct traceability between the requirements document and the Event-B model.

For the enhanced deployment, we changed this development process slightly, taking into account the lessons learnt. Figure 3.5 shows the general approach to the enhanced deployment. The biggest change is that we introduced an additional step between requirement engineering with Problem Frames and modelling in Event-B:

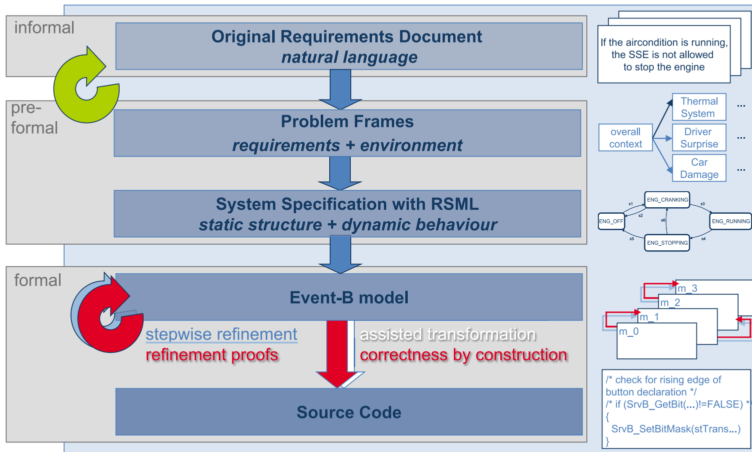


Fig. 3.5 Detailed development process

the specification document. With the approach taken during the pilot deployment, our major issue was to achieve direct traceability between the natural language requirements and the final Event-B model. We were successful in doing this, but we had to pay a price. There is a trade-off between traceability and architecture. With the pilot deployment approach, the end result will be a model which is traceable but—from a modelling, proving and maintainability point of view—poor. Our decision for the enhanced deployment was to use a development process which is more balanced with regard to these different qualities. By introducing the specification document as an intermediate step between the Problem Frames analysis and the Event-B model, we aimed to produce a more solution-oriented structure of the system. With the experience of the pilot deployment in mind, we chose RSML [6] (with some adjustments) as the language for the specification document. The reason for this is that a natural language description of finite state machines is cumbersome.

3.4.3 Quantitative Results Enhanced Deployment—Proof Obligations

In the second pilot, we were most interested in gathering evidence that the proof obligations, which arise when using Event-B, are manageable.

In Fig. 3.6, the absolute number of proof obligations (4,215) for the start-stop system model is shown. Of these 4,215 POs (proof obligations) the majority were proven automatically; 425 had to be proven manually. Figure 3.7 shows a more detailed view of the 425 manual proofs: 400 of them were very easy (needed time <1 min), 12 medium (needed time <1 h), and 13 of the manual proofs were difficult (needed time >1 h). This evaluation shows that it is in principle feasible to prove a system of the start-stop system size.

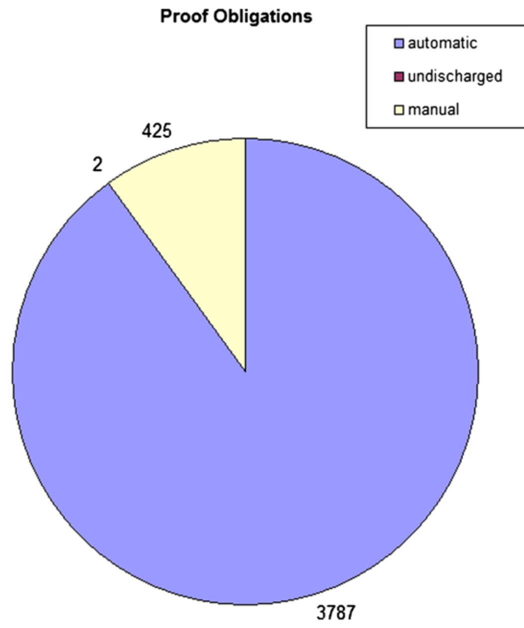


Fig. 3.6 Quantitative results—automatic vs. manual proof obligations

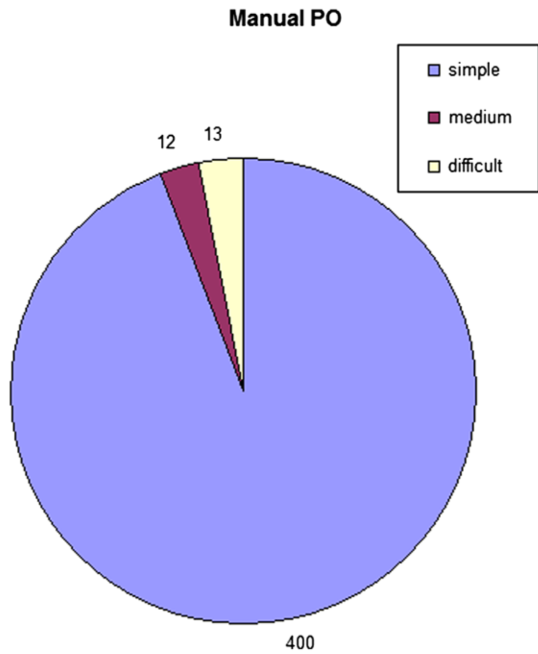


Fig. 3.7 Quantitative results—distribution of manual proof obligations

3.4.4 Tools

As mentioned in Sect. 3.4.1, the size of the application itself led to problems. During the pilot deployment the performance of Rodin was not satisfactory in dealing which such big models. There were performance issues related to the editing process itself (very slow reaction time in editing) as well as the proving interface. Significant progress in supporting big models was achieved as a result of this feedback. Nevertheless, some features essential for parallel team development of big models (e.g., version control) are still missing, which would discourage us from trying to model an application of comparable size again.

3.5 Lessons Learnt from Enhanced Deployment

During the pilot deployment we focused on finding a solution for bridging the gap between the informal and formal worlds. We used (an extended version of) the Problem Frames method for requirements engineering to solve this problem. Our main issue was preserving traceability between natural language requirements and the final formal model in Event-B. By doing this we also improved the quality of our requirements. Although the approach was generally successful, we also identified some drawbacks or open issues. During the enhanced deployment, we did not aim to reproduce the method developed previously (this would have been an option if everything had worked perfectly). Our goal was to reuse the steps which worked well and try out alternatives where we had identified problems.

In the enhanced deployment, we changed our development process by introducing a dedicated specification document written in a tailored version of RSML. We also introduced the identification and (Event-B) specification of invariants. This resulted in better support for good design and systematically deduced invariants which are independent of the concrete model. To achieve this, we moved our focus away from traceability towards design. There are several overall conclusions (taking into account both the pilot and enhanced deployments) about Event-B and the work described in this chapter.

Closed loop controller. There is no practicable way of modelling closed loop controllers in Event-B and it is therefore not possible to prove properties about continuous behaviour. This is an open area that needs more research. One of the main questions is whether it is useful to try this and, if yes, how this could be done.

Time. Time was from the very beginning an open issue and still is. In the embedded systems market there is a strong need to include the notion of time at some point during development. The inclusion of events ordering (e.g., the flow plug-in developed at Newcastle University) is a first step in addressing this problem.

Gap between informal and formal worlds. We have made considerable progress, which gives us confidence that this problem has been (or can be) solved. What is needed here is more experience of applying the developed methods and strong

tool support. We were able to improve the quality of requirements by applying Problem Frames as well as by formally proving invariants in Event-B.

Formal modelling. We have made significant progress in formal modelling of industrial size applications. At the same time, the cruise control application in particular exposed the limitations of Rodin in formal modelling. There is room for improvement.

Flexibility. The Rodin tool is flexible enough to be adjusted for different needs via the development of plug-ins. For example, the flow plug-in helped us to graphically specify a desired order of events, which was necessary to prove important properties of our system.

Industrial development process. There is an open issue in providing support for state-of-the-art industrial development processes: configuration management, variant management, team development, version management are not supported by Rodin (or not supported well enough).

Scalability. Rodin does not scale to large applications. There is work to be done related to decomposition (architecture), performance and stability as well as to supporting team operation.

3.6 Lessons Learnt from the Providers' Side

We recognised from the beginning that each deployment scenario would be different (they came from different industries and types of application), and we even anticipated to some extent the need to integrate the DEPLOY technologies with a variety of in-house engineering tools and standards. In spite of this, the reality was even harder than we expected, leading to frustrations and misunderstandings for both academic and deployment partners.

Furthermore, we had reason to claim that the Rodin toolset was moderately mature because we had learnt the lesson that tool support was essential for (most) technology transfer situations; Rodin tools reflected the outcome of a previous project that included industrial partners; and because the tools were well designed by an outstanding team. Again, despite our preparation and an awareness of the need to support the tools and expect evolution, there were problems on the tool front. Some lessons here include:

- Any *initial* omissions that are quickly recognised and found to be important by deployment partners must be quickly resolved. In our case, records were an example and the developed "workaround" proved to be rather confusing as time progressed.
- Developers must test their tools on problems at least as large as those likely to be used by deployment partners. Failure to do this can result in extreme frustration when developers face exponential performance problems.
- Do not assume that advice that works for classroom examples will wish away all serious problems (it is, for example, unrealistic to expect to be able to use large numbers of layers in developing industrial systems).

- Where we did develop tailored tools such as [3], they were of considerable benefit and were well appreciated.
- Connections to other methods might include, for example, work on real time system modelling, such as the "Duration Calculus" [10] or the related work by Hayes and Mahony [7].

More generally,

- One area where we would claim success is the flexibility in fitting the Problem Frames Approach into our interaction with Bosch colleagues. Any developer who thinks his or her single idea will solve the world's problems is mistaken.
- But we might have done better if we had accepted earlier just how valuable state diagrams were to the Bosch engineers: there is a strong case for providing interfaces from this ubiquitous notation directly to tools such as Pro-B; a translation of state diagrams to, for example, Event-B brings nothing.
- We were extremely fortunate that the Bosch team remained constant and that the members were all highly motivated; we would have achieved far less without this.
- *With hindsight* it is easy to see that the initial "block course" was only a first step in the technology transfer task and we should have had a clearer follow-up strategy.
- Not only did we make many trips to Bosch, we had an appointed "anchor man" as their prime contact. This helped generate trust and tolerance and provided a way of linking users with tool developers.

References

1. DEPLOY. Deliverable D19: D1.1 pilot deployment in the automotive sector WP1. http://www.deploy-project.eu/html/deliverables.html (2009)
2. DEPLOY. Deliverable D38: D1.2 report on enhanced deployment in the automotive sector WP1. http://www.deploy-project.eu/html/deliverables.html (2010)
3. Iliasov, A.: Use case scenarios as verification conditions: Event-B/flow approach. In: Troubitsyna, E. (ed.) SERENE. Lecture Notes in Computer Science, vol. 6968, pp. 9–23. Springer, Berlin (2011)
4. ISO 26262 Road vehicles—Functional safety. http://www.iso.org/iso/home.html
5. Jackson, M.: Problem Frames: Analyzing and Structuring Software Development Problems. Addison-Wesley Longman Publishing, Boston (2001)
6. Leveson, N.G., Heimdahl, M.P.E., Hildreth, H., Reese, J.D.: Requirements specification for process-control systems. IEEE Trans. Softw. Eng. **20**, 684–707 (1994)
7. Mahony, B., Hayes, I.: Using continuous real functions to model timed histories. In: Bailes, P. (ed.) Engineering Safe Software, pp. 257–270. Australian Computer Society, Sydney (1991)
8. Newcastle University. DEPLOY—Industrial deployment of system engineering methods providing high dependability and productivity. http://www.deploy-project.eu/ (May 2009)
9. V-modell XT. http://www.bit.bund.de/
10. Zhou, C., Hoare, C.A.R., Ravn, A.P.: A calculus of durations. Inf. Process. Lett. **40**, 269–271 (1991)

Chapter 4
Improving Railway Data Validation with ProB

Jérôme Falampin, Hung Le-Dang, Michael Leuschel, Mikael Mokrani, and Daniel Plagge

Abstract In this chapter, we describe the successful application of ProB in industrial projects realised by Siemens. Siemens is successfully using the B-method to develop software components for the zone and carborne controllers of CBTC systems. However, the development relies on certain assumptions about the actual rail network topology which have to be validated separately in order to ensure safe operation. For this purpose, Siemens has developed custom proof rules for Atelier B. Atelier B was, however, unable to deal with properties related to large constants (relations with thousands of tuples). These properties thus have, until now, had to be validated by hand at great expense (and revalidated whenever the rail network infrastructure changes). In this chapter we show how we have used ProB to overcome this challenge. We describe the deployment and current use of ProB in the SIL4 development chain at Siemens. To achieve this, it has been necessary to extend the ProB kernel for large sets and improve the constraint propagation phase. We also outline some of the effort and features involved in moving from a tool capable of dealing with medium-sized examples to one able to cope with actual industrial specifications. Notably, a new parser and type checker have had to be developed. We also touch upon the issue of validating ProB.

J. Falampin (✉) · H. Le-Dang · M. Mokrani
Siemens SAS IC-MOL, Paris, France
e-mail: jerome.falampin@gmail.com

H. Le-Dang
e-mail: hung.ledang@siemens.com

M. Mokrani
e-mail: mikael.mokrani@siemens.com

M. Leuschel · D. Plagge
University of Düsseldorf, Düsseldorf, Germany

M. Leuschel
e-mail: leuschel@cs.uni-duesseldorf.de

D. Plagge
e-mail: plagge@cs.uni-duesseldorf.de

A. Romanovsky, M. Thomas (eds.), *Industrial Deployment of System Engineering Methods*, DOI 10.1007/978-3-642-33170-1_4,
© Springer-Verlag Berlin Heidelberg 2013

4.1 Introduction

Siemens has been developing Communication-based Train Control (CBTC) products using the B-method [1] since 1998 and has over the years acquired considerable expertise in its applications. Siemens uses Atelier B [11], together with automatic refinement tools developed in-house, to successfully develop critical control software components in CBTC systems. Starting from a high-level model of the control software, refinement is used to make the model more concrete. Each refinement step is formally proven correct. When the model is concrete enough, an Ada code generator is used. This results in the system ensuring the highest Safety Integrity Level (SIL4). This railway software development process fully complies with current railway standards (EN50126, EN50128, EN50129) and significantly reduces the test cost at the validation step. In fact, the unit test is no longer needed in this process as it is replaced by proof activities during the development.

The first and best known example is obviously the controller software component for the fully automatic driverless Line 14 of the Paris Métro, also called Météor (Metro Est-Ouest Rapide). To quote [10], “Since the commissioning of line 14 in Paris on 1998, not a single malfunction has been noted in the software developed using this principle”. Since then, many other train control systems have been developed and installed worldwide by Siemens [2, 3]: San Juan metro (Puerto Rico, commissioned in 2003), New York metro Canarsie line (USA, commissioned in 2009), Paris metro line 1 (France, commissioned in 2011), Barcelona metro line 9 (Spain, commissioned in 2009), Algiers metro line 1 (Algeria, commissioned in 2011), Sao Paolo metro line 4 (Brazil, commissioned in 2010) and Charles de Gaulle Airport Shuttle line 1 (France, commissioned in 2005), among others.

Relationship to DEPLOY While the work described in this chapter was carried out within the context of DEPLOY, Siemens uses the classical B rather than Event-B in their current development process. Indeed, neither Event-B nor the Rodin tool is at the time of writing ready for software development at the industrial level. However, the classical B and Event-B clearly share a common basis, and the ProB tool used in this chapter works equally well for both formalisms.

The Property Verification Problem One aspect of the current development process which unfortunately is still problematic is the validation of properties of parameters only known at deployment time, such as the rail network topology parameters. In CBTC systems, the track is divided into several subsections, each of which is controlled by safety-critical software called ZC (Zone Controller). A Zone Controller contributes to the realisation of the ATP (Automatic Train Protection) and ATO (Automatic Train Operation) functions of CBTC systems for a portion of the track; these include train location, train anticollision, overspeed prevention, train movement regulation and train and platform door management, among others. In order to avoid multiple development, each ZC is made from a generic B-model and data parameters that are specific to a subsection and a particular deployment (cf. Fig. 4.1).

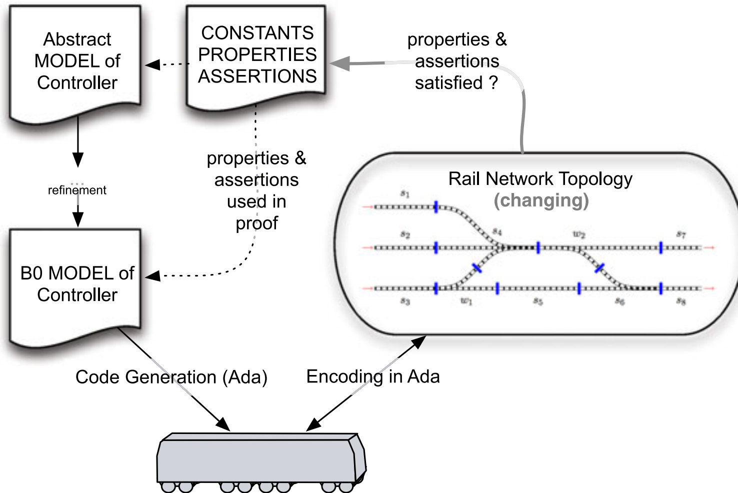


Fig. 4.1 A need for ZC data validation

These data parameters take the form of B functions describing, for example, tracks, switches, traffic lights, electrical connections and possible routes. These B functions are typically regrouped in basic invariant machines in which assumptions about topological properties are defined in the PROPERTIES clause. The proofs of the generic B-model rely on assumptions about data parameters, for instance, those about the topological properties of the track. We therefore have to make sure that parameters used for each subsection and deployment actually satisfy formal assumptions.

A trivial example is the slope of the track: the calculation of the stopping distance is made with an algorithm that requires the slope to be within $+/-5$ %. In the generic software model, we make the assumption that the slope is between -5 % and $+5$ %, and this enables to complete the proof of the algorithm. Then, we also have to make sure that the assumption is correct, that is to say the slope is indeed between -5 % and $+5$ % in the track data (and, of course, that the track data is correct regarding the actual track). More generally, data validation is needed when generic software is used with several sets of specific data.

4.2 An Approach to Validating ZC Data Using Atelier B

Siemens has developed the following approach to validating ZC data:

1. The parameters and topology are extracted from concrete Ada programs (each of which corresponds to an invariant basic machine in the B model; in B, a basic machine does not have a refinement or an implementation but has instead an implementation in Ada) and encoded in B0 (i.e., B-executable) syntax, written

into Atelier B definition files. Definition files contain only B DEFINITIONS, i.e., B macros.

This is done with the aid of a tool written in lex. Note that Siemens wants to check not only that the assumptions about the data parameters hold, but also that these have been correctly encoded in the Ada code. Hence, the data is extracted from Ada programs rather than directly from a higher-level description (which was used to generate the Ada code).

2. The definition files containing the topology and the other parameters are merged with the relevant invariant basic machines to create assertion machines. The properties for the concrete topology and parameters represented in basic machines are translated into B assertions in the relevant assertion machines. The macros derived from the actual data extracted from Ada programs are used to define the constant values and these definitions are realised as properties in the assertion machines.

   In B, assertions are predicates that should follow from the properties or invariants, and that have to be proved. Properties themselves do not have to be proved, but can be used by the prover. By translating the topology properties into B assertions, we create proof obligations which stipulate that the topology and parameter properties must follow from the concrete values of the constants.
3. The machines with assertions obtained from Step 2 are then grouped in a B project called IVP (Invariant Validation Project). Siemens tries to prove the assertions with Atelier B, using custom proof rules and tactics dedicated to dealing with explicit data values.
4. The assertions for which the proof is unsuccessful are investigated manually.

Problems with the Existing Process This approach initially worked quite well for ZC data, but then it ran into considerable problems:

- First, if the proof of a property fails, the feedback from the prover is not very useful in locating the problem (and it may be unclear whether there is in fact a problem with the topology or "simply" with the power of the prover).
- Second, and more importantly, the constants are currently 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 the maximum memory possible allocated. In some of the bigger, more recent models, even simply substituting values for variables fails with out-of-memory conditions.

  This is especially difficult as some of the properties are very large and complicated, and the prover typically fails on these. For example, for the San Juan development, 80 properties (out of 300) could not be checked by Atelier B either automatically or interactively (within reasonable time; sometimes just loading the proof obligation results in failure with an out-of-memory condition).

The second point means that these properties have to be checked by hand (e.g., by creating huge spreadsheets for the compatibility constraints of all possible itineraries on paper), which is very costly and arguably less reliable than automated checking. For the San Juan development, this meant about one man-month of effort; this is likely to be more for larger developments, such as the Canarsie line project [3].

4.3 First Experiments with ProB

The starting point of the experiment was to try to automate the proof of the remaining proof obligations by using an alternate technology. Indeed, the ProB tool [5, 6] has capabilities for dealing with B properties in order to animate and model-check B models. The big question was whether the technology would scale to handle the industrial models and the large constants in this case study.

In order to evaluate the feasibility of using ProB for checking the topology properties, on 8 July 2008 Siemens sent the STUPS team at the University of Düsseldorf the models for the San Juan case study. There were 23,000 lines of B spread over 79 files. Of these 79 files, two were to be analysed, containing a simpler and a more elaborate model. It then took STUPS some time to understand the models and get them through the new parser, whose development was being finalised at that time.

On 14 November 2008 STUPS were able to animate and analyse the first model. It uncovered one error in the assertions. However, at that point it became apparent that a new data structure would be needed in ProB to validate bigger models.

On 8 December 2008 STUPS were finally able to animate and validate the more elaborate model. It revealed four errors. Note that the STUPS team was not told about the presence of errors in the models (they were not even hinted at by Siemens), and initially STUPS believed that there was still a bug in ProB. In fact, the errors were genuine and identical to those previously uncovered by Siemens using manual inspection. The manual inspection of the properties took Siemens several weeks (about a man-month of effort). With ProB 1.3.0, checking the properties takes 4.15 seconds, and the assertions 1017.7 seconds (i.e., roughly 17 minutes) on a MacBook Pro with 2.33 GHz Core2 Duo. More information on this was included in the presentation at FM 2009 [7] and in the ensuing journal paper [8].

4.4 RDV: Railway Data Validator

In the first experiments, ProB was used with great success on the same IVP instead of Atelier B. But the creation of the IVP was still problematic, with little atomisation. As described in Sect. 4.1, each IVP is an encoding of specific wayside configuration data in B; this is required for validating the configuration data against the formal properties in the generic B project.

The goal was to create a tool that could automatically generate B projects (containing assertion machines), run ProB on the created B projects and compile the result in a synthesis report. In addition, this tool should work not only on ZC data, but also on CC (carborne controller) data, which were not formally validated before the use of ProB. Indeed, in CC software development, as shown in Fig. 4.2, the topology data are contained in text files and loaded "on the fly" by the CC software component when needed. Therefore, in order to enable CC data validation, the macros in definition files are derived from topology text files instead of Ada programs. Such macros are then merged with variables defined in the basic invariant machines to create assertion machines for the segment in question.

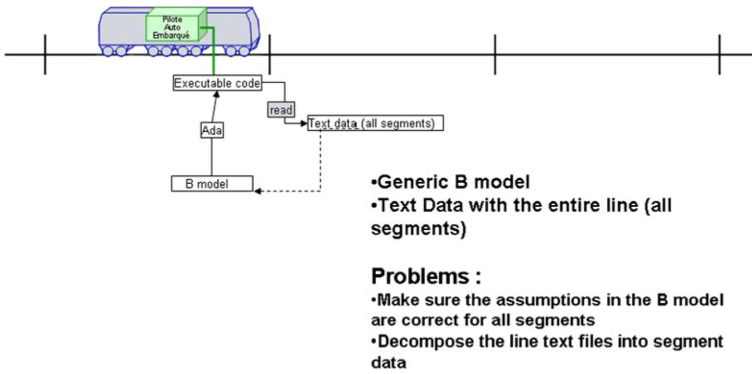


Fig. 4.2 CC data validation

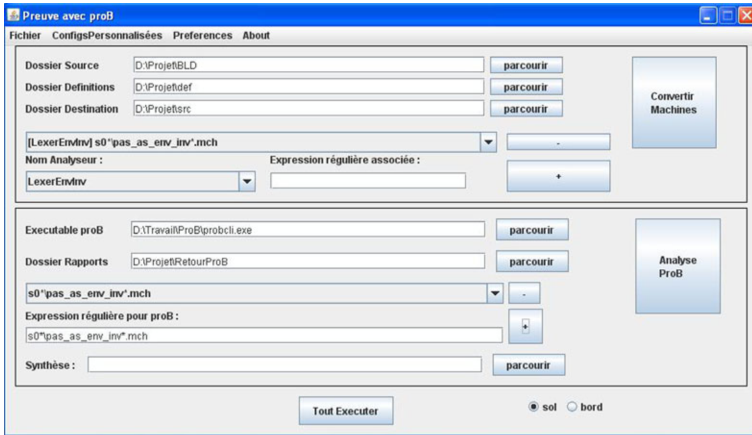


Fig. 4.3 RDV interface

The RDV is a new tool developed by Siemens. Via a graphical interface (cf. Fig. 4.3), the RDV provides the following services:

IVP Generation This function generates an IVP for a subsection in the case of ZC or a segment in the case of CC. The generated IVP is almost ready to be used by ProB or Atelier B. True, we still need some manual modification related to properties of non-function constants; however, in comparison with the tool used previously, manual modification on generated B machines is significantly reduced. In addition, with the file selection function based on regular expressions, the RDV enables the generation of a subset of assertion machines (i.e., only machines with modifications). Moreover, the automation of CC IVP creation is of great help to safety engineers as there are about several hundred IVP to be created (one project per segment).

ProB Launch The RDV enables users to parametrise ProB before launching it. ProB is called on each assertion machine in order to analyse the assertions contained in each machine. There is no need to have B experts for data validation to be carried out. Indeed, B experts are required only in case of a problem, whereas in the previously used process, B experts were required in any case, to carry out long and laborious tasks. In addition, ProB significantly reduces the time for checking data properties: from two or three days with the old tool to two or three hours per project with the RDV. Analysing CC assertion machines with ProB significantly reduces interaction with users as one does not need to launch Atelier B several hundred times on the created CC IVP.

Assertion-Proof Graph Generation This function provides a graphical way of investigating the proof of an assertion. It is based on a service provided by ProB for computing values of B expressions and truth values of B predicates, as well as all sub-expressions and sub-predicates. This is very useful because users often want to know exactly why an assertion fails. This was a problem with the Atelier B approach: when a proof fails, it is very difficult to find out why it did so, especially when large and complicated constants are involved.

Validation Synthesis Report The results of analyses realised by ProB on assertion machines are recorded in a set of .rp and .err files (one per B component and one per sector or segment). Each .rp (report) file contains the normal results of the analysis with ProB:

- Values used when initialising machines;
- Details of proof for each assertion checked;
- Result of the analysis (true, false or time-out).

Each .err (error) file contains the abnormal results of the analysis:

- Variables/constants with incorrect type;
- Variables/constants with multiple values, redefinition of value or missing value;
- Errors occurring when executing ProB (e.g., missing file).

When all report and error files have been generated, an HTML synthesis report is issued. This report gives the result (the number of false/true assertions, time-outs, unknown results) for each sector or segment. For each B component, a hyper-link to the detailed error and report files provides access to the results of the component (to show which assertion is false, for instance).

4.5 ZC Data Validation

As CC data validation differs from ZC data validation only in the way in which assertion machines are created, we present here an example of the latter to show how data validation is carried out with the RDV.

As shown in Fig. 4.1, the track is decomposed into several sectors (from two to ten); there is one zone controller dedicated to each sector. Ada data is linked with

Ada code transcoded from the generic B model. Below is a B machine with some properties of data inv_zaum_ quai_i and inv_zaum_troncon_i:

```
MACHINE
 pas_as_env_inv_zaum
SEES
 pas_as_env_typ_quai,
 pas_as_env_typ_zaum,
 pas_as_env_typ_troncon
CONCRETE_CONSTANTS
 inv_zaum_quai_i,
 inv_zaum_troncon_i
DEFINITIONS
 typage_tab == (
   inv_zaum_quai_i : t_nb_zaum_par_pas --> t_nb_quai_par_pas &
   inv_zaum_troncon_i : t_nb_zaum_par_pas --> t_nb_troncon_par_pas )
PROPERTIES
 typage_tab &
   /* Proprietes */
  !xx.(xx : t_zaum_pas
         =>
         inv_zaum_quai_i(xx) : t_quai_pas \/ {c_quai_indet}) &
  !xx.(xx : t_zaum_pas
         =>
         inv_zaum_troncon_i(xx) : t_troncon_pas \/ {c_troncon_indet})
END
```

An example of the relevant Ada program with configuration data follows:

```
with SEC_OPEL_FONC;
with PAS_AS_ENV_TYP_FONC;

use SEC_OPEL_FONC;

package pas_as_env_inv_zaum_val is
   subtype T_INV_ZAUM_TRONCON_I
     is SEC_OPEL_FONC.FINT_ARRAY_1(PAS_AS_ENV_TYP_FONC.T_NB_ZAUM_PAR_PAS);
  subtype T_INV_ZAUM_QUAI_I
     is SEC_OPEL_FONC.FINT_ARRAY_1(PAS_AS_ENV_TYP_FONC.T_NB_ZAUM_PAR_PAS);
INV_ZAUM_TRONCON_I : constant T_INV_ZAUM_TRONCON_I := T_INV_ZAUM_TRONCON_I'(
      1 => 3,      2 => 3,      3 => 3,      4 => 3,
      5 => 4,      6 => 4,      7 => 3,      8 => 3,
      9 => 3,      10 => 3,      11 => 4,      12 => 4,
      13 => 4,      14 => 4,      15 => 4,     16 => 4,
      17 => 3,      18 => 3,      19 => 3,      21 => 3,
      22 => 1,      23 => 1,      24 => 1,      25 => 1,
      26 => 1,      27 => 2,      28 => 2,      29 => 2,
      30 => 1,      31 => 2,      32 => 2,      33 => 2,
      34 => 2,      35 => 2,      36 => 3,
      OTHERS=>0
    );
INV_ZAUM_QUAI_I : constant T_INV_ZAUM_QUAI_I := T_INV_ZAUM_QUAI_I'(
      23 => 1,      25 => 2,
      27 => 3,      31 => 4,
      33 => 5,
      OTHERS=>0
    );
end pas_as_env_inv_zaum_val;
```

By using the RDV we obtained the following assertion machine:

```
MACHINE
 pas_as_env_inv_zaum
SEES
 pas_as_env_typ_quai,
 pas_as_env_typ_zaum,
 pas_as_env_typ_troncon
```

```
CONCRETE_CONSTANTS
 inv_zaum_quai_i,
 inv_zaum_troncon_i
DEFINITIONS
 "pas_as_env_inv_zaum_val.1.def";
 typage_tab == (
    inv_zaum_quai_i    : t_nb_zaum_par_pas --> t_nb_quai_par_pas  &
    inv_zaum_troncon_i : t_nb_zaum_par_pas --> t_nb_troncon_par_pas )
PROPERTIES
 inv_zaum_quai_i = inv_zaum_quai_i_indetermine <+
              inv_zaum_quai_i_surcharge &
 inv_zaum_troncon_i = inv_zaum_troncon_i_indetermine <+
              inv_zaum_troncon_i_surcharge
ASSERTIONS
 typage_tab ;
 !xx.(xx : t_zaum_pas => inv_zaum_quai_i(xx): t_quai_pas \/ {c_quai_indet});
 !xx.(xx : t_zaum_pas => inv_zaum_troncon_i(xx) :
             t_troncon_pas \/ {c_troncon_indet})
END
```

The machine name is not changed for traceability purposes. The ASSERTIONS clause in the new machine contains predicates which were in the PROPERTIES clause of the original machine. The DEFINITIONS clause includes the definition file pas_as_env_inv_zaum_val.1.def, which contains macros derived from data defined in pas_as_env_inv_zaum_val.1.ada:

- inv_zaum_quai_i_indetermine and inv_zaum_quai_i_ surcharge, which were derived from the configuration data INV_ZAUM_QUAI_I.
- inv_zaum_troncon_i_indetermine and inv_zaum_troncon_i_surcharge, which were derived from the configuration data INV_ZAUM_QUAI_I.

```
DEFINITIONS
        inv_zaum_quai_i_indetermine == (t_nb_zaum_par_pas)*{0};
        inv_zaum_quai_i_surcharge =={
                23|->1, 25|->2, 27|->3, 31|->4, 33|->5
        };

        inv_zaum_troncon_i_indetermine == (t_nb_zaum_par_pas)*{0};
        inv_zaum_troncon_i_surcharge =={
                1|->3, 2|->3, 3|->3, 4|->3, 5|->4, 6|->4, 7|->3,
                8|->3, 9|->3, 10|->3, 11|->4, 12|->4, 13|->4,
                14|->4, 15|->4, 16|->4, 17|->3, 18|->3, 19|->3,
                21|->3, 22|->1, 23|->1, 24|->1, 25|->1, 26|->1,
                27|->2, 28|->2, 29|->2, 30|->1, 31|->2, 32|->2,
                33|->2, 34|->2, 35|->2, 36|->3
        }
```

Macros inv_zaum_quai_i_indetermine and inv_zaum_quai_i_ surcharge are then used to define inv_zaum_quai_i as shown in the PROPERTIES clause. As the modified PROPERTIES clause is instantiated using the definition file, inv_zaum_quai_i is replaced by the actual data:

```
 inv_zaum_quai_i = (t_nb_zaum_par_pas)*{0}
                     <+ {23|->1,25|->2,27|->3,31|->4,33|->5}
```

Macros inv_zaum_troncon_i_indetermine and inv_zaum_troncon_i_surcharge are used to define and instantiate inv_zaum_troncon_i.

The goal of the modification presented above is therefore to include the actual data (in the PROPERTIES clause) and the definition files (in the DEFINITION clause), and displace the assumptions on data in the ASSERTIONS clause. This

modification will lead to some proof obligations “data” = “assumptions” being generated. For example, with inv_zaum_quai_i, we will have to prove the following (simplified) proof obligation:

```
inv_zaum_quai_i = (t_nb_zaum_par_pas)*{0} <+ {23|->1,25|->2,27|->3,
31|->4,33|->5} => inv_zaum_quai_i : t_nb_zaum_par_pas --> t_nb_quai_par_pas
```

4.6 Industrial Projects Validated Using the RDV

In addition to the San Juan case study, ProB has been used in all ongoing projects, with Atelier B being used in parallel for data validation; the results show that ProB is more effective and less restrictive than Atelier B for railway data validation. Below is an account of experiences from typical projects.

ALGIERS line 1 (ZC). This is the first driverless metro line in Africa. This line is composed of ten stations over 10 km. The line is divided into two sections, each controlled by a ZC. There were 25,000 lines of B spread over 130 files. ProB was used for the last three versions of this project railway data. Table 4.1 shows the results obtained with ProB on the last version: Each line represents the summary result for one sector. The Predicates column shows the number of assertions to be analysed; the TRUE column represents the number of assertions verified by ProB (no counterexample found by ProB). The FALSE column represents the number of assertions failed by ProB (with a counterexample). The UNKNOWN column represents the number of assertions that ProB dotes not know how to verify. The TIMEOUT column shows the number of assertions that ProB encountered a time-out problem during the analysis.
For each sector, there were two assertions that could not be proved with Atelier B due to their complexity. One is shown below. This property has been proven wrong with ProB using the railway data for both sectors.

```
ran(inv_quai_variants_nord_troncon >< ((((((((((t_quai_pas <|
inv_quai_adh_red_nord_rg_variant_bf_i) |> t_rg_variant_bf) \/ ((t_quai_pas <|
inv_quai_ato_inhibe_nord_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <|
inv_quai_mto_inhibe_nord_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <|
inv_quai_atp_inhibe_nord_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <|
inv_quai_arret_tete_nord_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <|
inv_quai_arret_centre_nord_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <|
inv_quai_arret_queue_nord_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <|
inv_quai_tete_ape_nord_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <|
inv_quai_centre_ape_nord_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <|
inv_quai_queue_ape_nord_rg_variant_bf_i) |> t_rg_variant_bf))) /\
ran(inv_quai_variants_sud_troncon >< ((((((((((t_quai_pas <|
inv_quai_adh_red_sud_rg_variant_bf_i) |> t_rg_variant_bf) \/ ((t_quai_pas <|
inv_quai_ato_inhibe_sud_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <|
inv_quai_mto_inhibe_sud_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <|
inv_quai_atp_inhibe_sud_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <|
inv_quai_arret_tete_sud_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <|
inv_quai_arret_centre_sud_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <|
inv_quai_arret_queue_sud_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <|
inv_quai_tete_ape_sud_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <|
inv_quai_centre_ape_sud_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <|
inv_quai_queue_ape_sud_rg_variant_bf_i) |> t_rg_variant_bf))) = {}
```

Sao Paolo line 4 (ZC). This is the first driverless metro line in South America. It has 11 stations over 12.8 km and is divided into three sections. The model consisted of 210 files with over 30,000 lines of B. ProB has been used for the last six versions of this project railway data. For the last version, the results are presented in Table 4.2.

Table 4.1 ALGIERS line 1 (ZC)

Sector	Predicates	TRUE	FALSE	UNKNOWN	TIMEOUT
pas_as_inv_s01.html	1174	1164	10	0	0
pas_as_inv_s02.html	1174	1162	12	0	0

Table 4.2 Sao Paolo line 4 (ZC)

Sector	Predicates	TRUE	FALSE	UNKNOWN	TIMEOUT
pas_as_inv_s036.html	1465	1459	6	0	0
pas_as_inv_s037.html	1465	1460	5	0	0
pas_as_inv_s038.html	1465	1457	8	0	0

ProB has detected issues with a group of properties which had to be put in commentary in machines used with Atelier B because they were crashing the predicate prover. Here is an example of one of them:

```
!(cv_o,cv_d).(((cv_d : t_cv_pas & cv_o : t_cv_pas) & cv_o :
inv_lien_cv_cv_orig_i[inv_chainage_cv_liste_i[inv_chainage_cv_deb(cv_d) ..
inv_chainage_cv_fin(cv_d)]])
& not(inv_lien_cv_cv_dest_i((t_cv_pas <| inv_lien_cv_cv_orig_i~)|>
inv_chainage_cv_liste_i[inv_chainage_cv_deb(cv_d) ..
inv_chainage_cv_fin(cv_d)](cv_o)) = cv_d)
    => inv_lien_cv_cv_dest_i((t_cv_pas <| inv_lien_cv_cv_orig_i~) |>
inv_chainage_cv_liste_i[inv_chainage_cv_deb(cv_d) ..
inv_chainage_cv_fin(cv_d)](cv_o)) : inv_cv_pas_modifiable_i~[{TRUE}]))
```

After analysis, we concluded that the problems were not, fortunately, critical. Nonetheless, without ProB, it would have been a lot harder to find them. The assertion-proof graphs helped us better understand the source of the problems.

Paris line 1 (ZC). This is a project to automate line 1 (25 stations over 16.6 km) of the Paris Métro. The line has been gradually upgraded to driverless train while in operation. The first driverless train was commissioned in November 2011. In February 2012, out of 49 trains 17 were driverless, operating in conjunction with manually driven ones. The line is going to be fully driverless by early 2013. It is divided into six sections. The model to be checked is the same as the SPL4 one. ProB has been used for the last seven versions of railway data. The results for the last version are presented in Table 4.3.

Paris line 1 (PAL). PAL (Pilote Automatique Ligne) is a controller line that implements the Automatic Train Supervision (ATS) function of CBTC systems. The B models of PAL consisted of 74 files with over 10,000 lines of B. Overall, 2,024 assertions about concrete data of the PAL needed to be checked. ProB found 12 problems in under five minutes. They were later examined and confirmed by manual inspection at Siemens.

CDGVAL LISA (Charles de Gaulle Véhicle Automatique Léger). This is an extension of CDVVAL which was commissioned in 2005. This line is going to be operational in early 2012. The LISA model consisted of 10,000 line of B over

Table 4.3 Paris line 1 (ZC)

Sector	Predicates	TRUE	FALSE	UNKNOWN	TIMEOUT
pas_as_inv_s011.html	1503	1501	2	0	0
pas_as_inv_s012.html	1503	1498	5	0	0
pas_as_inv_s013.html	1503	1496	7	0	0
pas_as_inv_s014.html	1503	1499	4	0	0
pas_as_inv_s015.html	1503	1498	5	0	0
pas_as_inv_s016.html	1503	1498	5	0	0

Table 4.4 Roissy LISA (ZC)

Sector	Predicates	TRUE	FALSE	UNKNOWN	TIMEOUT
ry_pads_as_inv_pa31.html	1038	1038	0	0	0
ry_pads_as_inv_pa32.html	957	957	0	0	0
ry_pads_as_inv_pagat.html	1038	1038	0	0	0

30 files. This project has three sections. ProB has been used for the last 3 versions of this project railway data. The results for the last version are presented in Table 4.4.

4.7 Tool Issues

A crucial consideration when trying to successfully deploy a tool like PROB in an industrial project is that the changes made to the existing models must be minimal because these are usually regarded as cost- and time-intensive. We cannot expect industrial users to adapt their models to an academic tool. Thus it is important to support the full language used in industry.

To make PROB fit for use in industry, we had to make significant changes to the parser and type checker even to be able to load the models. Then several changes to PROB's core had to be made to make the large data sets in the models tractable.

The Parser Previous versions of PROB used a freely available parser that lacked some features of Atelier B's syntax. In particular, it had no support for parameterised DEFINITIONS, for tuples that use commas instead of `|->` or for definition files. We realised that the code base of the existing parser was very difficult to extend and maintain and decided to completely rewrite the parser. We decided to use the parser generator SableCC [4], because it allowed us to write a clean and readable grammar. The following will briefly describe some of the issues we encountered in developing the parser and point out where our parser's behaviour deviates from Atelier B's behaviour.

- Atelier B's definitions provide a macro-like functionality with parameters, i.e., a call to a definition somewhere in the specification is syntactically replaced by the definition. The use of such macros can, however, lead to subtle problems. For example, consider the definition `sm(x,y)==x+y`:. The expression `2*sm(1,4)` is not equal to 10 as we might expect. Instead, it is replaced by `2*1+4`, which equals 6.

  We believed that this could easily lead to errors in the specification and decided to deviate from Atelier B's behaviour in this respect. Thus, we treat `2*sm(1,4)` as `2*(1+4)`, which equals 10 as expected.
- Another problem was ambiguity in the grammar when dealing with definitions and overloaded operators like `;` and ||. `;` can be used to express composition of relations as well as sequential composition of substitutions. When such an operator is used in a definition like `d(x)==x;x`, it is not clear which meaning is expected. It could be substitution when calling it with `d(y:=y+1)` or, in another context, it could be an expression when calling it with two variables a, b, as in `d(a,b)`. We resolved the problem by requiring the user to use parentheses for expressions (as in `(x;x)`). Substitutions are never put in parentheses because they are bracketed by `BEGIN` and `END`.
- The parser generator SableCC does not support source code positions. Thus we are not able to trace expressions in the abstract syntax tree back to the source code, which is important for identifying the source of any errors in specification. To circumvent this problem, we modified the parser generator itself to support source code positions.

The resulting parser deviates from Atelier B's in just some respects. It is relatively rare for an Atelier B model to need to be rewritten to work with PROB.

The Type Checker We re-implemented our type checker for B specifications because the previous version often needed additional predicates to provide typing information. We implemented a type inference similar to the Hindley-Milner algorithm [9], which is more powerful than Atelier B's type checker and thus accepts all specifications that Atelier B would accept and more. A nice side-effect of the new type checker is that by using the information provided by the parser, we can highlight an erroneous expression in the source code. This improves user experience, especially for new users.

Improved Data Structures and Algorithms Originally, PROB represented all sets internally as lists. But with thousands of operations on lists to carry out, it performed very badly. We introduced an alternative representation based on self-balancing binary search trees (AVL trees), which provides much faster access to its elements. In this context we examined the bottlenecks of the kernel in light of the industrial models given to us and implemented specialised versions of various operations. For example, an expression like `dom(r)<:1..10` can be checked very efficiently when `r` is represented by an AVL tree. We can exploit the feature that sorts the tree elements and just check if the smallest and largest elements are in the interval.

Infinite Sets Several industrial specifications make use of definitions of infinite sets such as `INTEGER \ {x}`. ProB now detects certain patterns and keeps them symbolic, i.e., instead of trying to calculate all elements, ProB just stores the conditions that all elements fulfil and uses them for member checks and function applications. Examples of expressions that can be kept symbolic are

- integer sets and intervals,
- complementary sets (as in the example above),
- some lambda expressions and
- unions and intersections of symbolic sets.

4.8 Tool Validation

In order to be able to use ProB without resorting to Atelier B, Siemens asked the Düsseldorf team to validate ProB. There are no general requirements for using a tool within an SIL 4 development chain; the amount of validation depends on how critical the tool is in the development or validation chain. In this case, Siemens requires

- a list of all critical modules of the ProB tool, i.e., modules used by the data validation task, that can lead to a property being marked wrongly as fulfilled
- a complete coverage of these modules by tests
- a validation report, with a description of ProB functions, and a classification of functions into critical and non-critical, along with a detailed description of the various techniques used to ensure a proper functioning of ProB.

Two versions of the validation report have already been produced, and a third version is under development. The test coverage reports are now generated completely automatically using the continuous integration platform "Jenkins".[1] This platform also runs all unit, regression, integration and other tests.

Below, we briefly describe the various tests as well as additional validation safeguards. In addition, the successful case studies described earlier in this chapter also constitute validation by practical experience: in all cases the approach based on ProB has proven to be far superior to the existing one.

Unit Tests ProB contains over 1,000 manually entered unit tests at the Prolog level. These check, for example, the proper functioning of the various core predicates operating on B data structures. In addition, there is an automatic unit test generator, which tests the ProB kernel predicates in many different scenarios and with different set representations. For example, starting from the initial call,

```
union([int(1)],[int(2)],[int(1),int(2)]),
```

[1] See http://en.wikipedia.org/wiki/Jenkins_(software).

the test generator will derive 1,358 unit tests. The latter kind of testing is particularly important for the PROB kernel, which relies on co-routining: we want to check that the kernel predicates behave correctly irrespective of the order in which (partial) information is propagated.

Integration and Regression Tests PROB contains over 500 regression tests, which are made up of B models along with saved animation traces. The models are loaded, the saved animation traces replayed and the models run through the model checker. The tests have turned out to be extremely valuable in ensuring that a bug once fixed remains fixed. They are also very effective in uncovering errors in arbitrary parts of the system (e.g., the parser, type checker, the interpreter or the PROB kernel).

Self-model Check with Mathematical Laws This approach allows PROB model checker to check itself, in particular, the PROB kernel and the B interpreter. The idea is to formulate a wide variety of mathematical laws (e.g., taken from [1]) and then use the model checker to ensure that no counterexample to these laws can be found. The self-model check has been very effective in uncovering errors in the PROB kernel and interpreter. Furthermore, the self-model checking tests rely on every component of the entire PROB execution environment working perfectly; otherwise, a violation of a mathematical law could be found. In other words, in addition to the PROB main code, the parser, type checker, Prolog compiler, hardware and operating system all have to work perfectly. Indeed, we have identified a bug in our parser (`FIN` was treated as `FIN1`) using the self-model check. Furthermore, we have even uncovered two bugs in the underlying SICStus Prolog compiler using the self-model check.

Validation of the Parser We execute our parser on a large number of our regression test machines and pretty-print the internal representation. We then parse the internal representation and pretty-print it again, verifying (with `diff`) that we get exactly the same result. This type of validation can be easily applied to a large number of B machines and will detect if the parser omits, reorders or modifies expressions, provided the pretty printer does not compensate errors of the parser.

Validation of the Type Checker At the moment we also input a large number of our regression test machines and pretty-print the internal representation, this time with explicit typing information inserted. We now run this automatically generated file through the Atelier B parser and type checker. In this way we test whether the typing information inferred by our tool is compatible with the Atelier B type checker. (Obviously, we cannot use this approach in cases where our type checker detects a type error.) Also, as the pretty printer only prints the minimal number of parentheses, we also ensure to some extent that our parser is compatible with the Atelier B parser. Again, this validation can be easily applied to a large number of B machines. More importantly, it can be systematically applied to the machines that

ProB validates for Siemens: provided the parser and pretty printer are correct, this gives us a guarantee that the typing information for the machines is correct. The latest version of ProB has a command for cross-checking the typing of the internal representation with Atelier B in this manner.

With the help of testing we identified 26 errors in the B syntax as described in the Atelier B English reference manuals, upon which our pretty printer and parser were based (the French versions were correct; our parser is now based on the French reference manuals). We also detected that Atelier B reports a lexical error ("illegal token |-") if the vertical bar (|) of a lambda abstraction is followed directly by the minus sign.

Double Evaluation As an additional safeguard during data validation, all properties and assertions were checked twice, both positively and negatively. Indeed, ProB has two Prolog predicates to evaluate B predicates: one positive version, which will succeed and enumerate solutions if the predicate is true, and a negative version, which will succeed if the predicate is false and then enumerate solutions to the negation of the predicate. For an assertion to be classified as true, the positive Prolog predicate must succeed *and* the negative Prolog predicate must fail, introducing a certain amount of redundancy (admittedly with common error modes). In fact, if both the positive and negative Prolog predicates succeeded for a particular B predicate, then a bug in ProB would be uncovered. If both fail, then either the B predicate is undefined or we have a bug in ProB. This validation method can detect errors in the predicate evaluation parts of ProB, i.e., in the treatment of the Boolean connectives $\vee$, $\wedge$, $\Rightarrow$, $\neg$, $\Leftrightarrow$, quantification $\forall$, $\exists$, and the various predicate operators such as $\in$, $\notin$, $=$, $\neq$ and $<$. Redundancy cannot detect bugs inside expressions (e.g., $+$, $-$) or substitutions (but the other validation methods mentioned above can).

4.9 Conclusions

In this chapter we have described a successful application of the ProB tool for data validation in several industrial applications. This required the extension of the ProB kernel for large sets as well as an improved constraint propagation algorithm. We also outlined some of the effort and features that were required in moving from a tool capable of dealing with medium-sized examples toward a tool able to deal with actual industrial specifications.

Acknowledgements We would like to thank Jens Bendisposto, Fabian Fritz and Sebastian Krings for assisting us in various ways, both in writing the chapter and in applying ProB to the Siemens models. Most of this research has been funded by the EU FP7 project 214158: DEPLOY (Industrial deployment of advanced system engineering methods for high productivity and dependability). Parts of this chapter are taken from [7, 8].

References

1. Abrial, J.-R.: The B-Book. Cambridge University Press, Cambridge (1996)

2. Badeau, F., Amelot, A.: Using B as a high level programming language in an industrial project. In: Treharne, H., King, S., Henson, M.-C., Schneider, S.-A. (eds.) Proceedings of ZB 2005: Formal Specification and Development in Z and B, 4th International Conference of B and Z Users. Lecture Notes in Computer Science, vol. 3455, pp. 334–354. Springer, Berlin (2005)
3. Essamé, D., Dollé, D.: B in large-scale projects: The Canarsie line CBTC experience. In: Julliand, J., Kouchnarenko, O. (eds.) Proceedings of B 2007: 7th Int. Conference of B Users. Lecture Notes in Computer Science, vol. 4355, pp. 252–254. Springer, Berlin (2007)
4. Gagnon, E.: SableCC, an object-oriented compiler framework. Master thesis, McGill University, Montreal, Canada (1998). http://www.sablecc.org
5. Leuschel, M., Butler, M.-J.: ProB: A model checker for B. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) Proceedings FME 2003: Formal Methods. Lecture Notes in Computer Science, vol. 2805, pp. 855–874. Springer, Berlin (2003)
6. Leuschel, M., Butler, M.-J.: ProB: An automated analysis toolset for the B method. Int. J. Softw. Tools Technol. Transf. **10**(2), 185–203 (2008)
7. Leuschel, M., Falampin, J., Fritz, F., Plagge, D.: Automated property verification for large scale B models. In: Cavalcanti, A., Dams, D. (eds.) Proceedings FM 2009. Lecture Notes in Computer Science, vol. 5850, pp. 708–723. Springer, Berlin (2009)
8. Leuschel, M., Falampin, J., Fritz, F., Plagge, D.: Automated property verification for large scale B models with ProB. Form. Asp. Comput. **23**(6), 683–709 (2011)
9. Milner, R.: A theory of type polymorphism in programming. J. Comput. Syst. Sci. **17**, 348–375 (1978)
10. Siemens: B method—optimum safety guaranteed. Imagine **10**, 12–13 (2009)
11. Steria, Aix-en-Provence. France. Atelier B, user and reference manuals (2009). http://www.atelierb.eu/

Chapter 5
Deployment in the Space Sector

Dubravka Ilić, Linas Laibinis, Timo Latvala, Elena Troubitsyna, and Kimmo Varpaaniemi

Abstract The greatest challenges in space projects are ensuring traceability of system requirements throughout the development process and guaranteeing that they have been properly implemented, and that the overall system therefore complies with the standards adopted in the sector. In addition, the software development process is often influenced by a number of factors, such as constraints on the hardware platform, stringent performance requirements, and results of the RAMS (Reliability, Availability, Maintainability and Safety) analysis. To address the above challenges, Space Systems Finland Ltd. has used the DEPLOY project to explore ways of using formal modelling and verification for facilitating requirements engineering, deriving robust system architectures and increasing the degree of development automation.

5.1 Background

DEPLOY's work in the space sector has been carried out by SSF (Space Systems Finland Ltd.) in close cooperation with Åbo Akademi University and Newcastle University accompanied by considerable interaction with the University of Southampton, CETIC (Centre d'Excellence en Technologie de l'Information et de la

D. Ilić (✉) · T. Latvala · K. Varpaaniemi
Space Systems Finland Ltd., Kappelitie 6, FI-02200 Espoo, Finland
e-mail: dubravka.ilic@ssf.fi

T. Latvala
e-mail: timo.latvala@ssf.fi

K. Varpaaniemi
e-mail: kimmo.varpaaniemi@ssf.fi

L. Laibinis · E. Troubitsyna
Department of Information Technologies, Åbo Akademi University, Joukahaisenkatu 3-5 A, FI-20520 Turku, Finland

L. Laibinis
e-mail: linas.laibinis@abo.fi

E. Troubitsyna
e-mail: elena.troubitsyna@abo.fi

A. Romanovsky, M. Thomas (eds.), *Industrial Deployment of System Engineering Methods*, DOI 10.1007/978-3-642-33170-1_5,
© Springer-Verlag Berlin Heidelberg 2013

Communication), ETHZ (Eidgenössische Technische Hochschule Zürich), the University of Düsseldorf and Systerel.

Within the DEPLOY project, SSF has explored ways to integrate refinement-based methods and tools into the processes traditionally used to develop space applications. Space software development in Europe is governed by the guidelines provided by the European Space Agency (ESA). Specifically, the main software standards provided by the ESA are ECCS-E-ST-40C [5] and ECSS-Q-ST-80C [6]. However, a space mission typically has "self-contained documentation", i.e., certain requirements are essentially mission-specific tailorings of ESA's standards. Non-compliance of these mission-specific tailorings with the standards can be acceptable provided the ESA officially allows it, which happens quite often.

In DEPLOY, SSF has experimented with formal modelling and development of several typical space applications. These developments were based on (a subset of) requirements for

- BepiColombo Solar Intensity X-ray and Particle Spectrometer (SIXS)/Mercury Imaging X-ray Spectrometer (MIXS) on-board software (OBSW), and
- reusable Attitude and Orbit Control System software (AOCS).

Currently, SSF has a major responsibility for designing BepiColombo SIXS/MIXS OBSW. In contrast, the work on AOCS is to a large extent based on SSF's previous design experience from several projects.

5.1.1 SSF (Space Systems Finland Ltd.)

The main business of SSF is system and software design as well as safety critical system assessment in several chosen technological areas, such as nuclear industry, Global Positioning System (GPS) technology, machinery and space. In the DEPLOY project, SSF has focused on the space domain.

SSF is one of Europe's leading providers of space application software. Over the last 20 years, the company has developed high reliability software solutions for many ambitious European space missions. SSF focuses on mission-critical solutions for highly demanding environments, such as on-board software, robotics, Digital Signal Processing (DSP), data processing facilities, and processing control environments. The goal of SSF is to provide customers and partners with reliable and innovative solutions for their demanding applications and complex systems.

In its space sector activities, SSF concentrates on several key technology areas such as on-board software, embedded software (especially embedded software with strict real-time requirements), ground software, scientific computing, system studies and consulting, and the creation of software development tools and standards.

5.1.2 Space Software Development in Europe

The space sector in Europe mostly functions under the auspices of the European Space Agency (ESA). Although there are significant activities unrelated to ESA

in certain domains, such as telecom and military satellites, the ESA standards and practices also tend to dominate there.

To facilitate safe use and cost-effective development of space assets, ESA pursues various standardisation and technology development activities which can be roughly categorised as follows:

- Method and technology research on space applications;
- Standardisation of system and software engineering development, including software lifecycles, product assurance, project management and safety analysis.
- Standardisation of reference architectures for satellite on-board software in order to facilitate reuse of software components and reduce the cost of development.

Software development in ESA follows the general principles defined for avionics software in standards such as RTCA DO-178B [24]. Moreover, the European Cooperation for Space Standardisation (ECSS) series of standards also covers over 100 topics. In particular, standards ECSS-E-ST-40C [5] and ECSS-Q-ST-80C [6] together define most software engineering-related activities.

The software engineering activities can be broadly divided into the specification, design, implementation, integration and validation phases. We believe that formal modelling and verification can greatly facilitate the specification and design phases by helping clarify and structure the system requirements and define the system architecture.

Among the greatest challenges in space projects are:

- managing requirement changes, e.g., integrating requirements resulting from Reliability, Availability, Maintainability and Safety (RAMS) activities,
- achieving traceability of the requirements to different levels of specification, and
- validating that the requirements have been properly implemented.

Below we will describe the applications that were used as case studies in DEPLOY to investigate these issues.

5.1.3 BepiColombo

BepiColombo is the first European mission to Mercury, the innermost planet of the Solar System. The mission consists of two orbiters. The Mercury Planetary Orbiter (MPO), the responsibility of ESA, will study the surface and the internal composition of the planet at different wavelengths and with different techniques. The Mercury Magnetospheric Orbiter (MMO), which is the responsibility of the Japan Aerospace Exploration Agency (JAXA), will study the planet's magnetosphere.

According to the ESA schedule [7] at the time of writing, MPO and MMO will be launched in 2015, start orbiting Mercury in November 2022 and then conduct their scientific programme for one or two Earth years.

SSF has a major responsibility for designing On-Board Software (OBSW) for two MPO instruments, namely, the Solar Intensity X-ray and Particle Spectrometer

(SIXS) and the Mercury Imaging X-ray Spectrometer (MIXS). SIXS will perform measurements of X-rays and solar particles at a high time resolution and with a very wide field of view. MIXS will use the X-ray fluorescence analysis method to produce a global map of the surface atomic composition at a high spatial resolution.

SIXS consists of two sensor units, namely, SIXS-X (X-ray spectrometer) and SIXS-P (particle spectrometer); MIXS also consists of two sensor units, namely, MIXS-T (telescope) and MIXS-C (collimator). The Data Processing Unit (DPU) takes care of the four sensor units by controlling the power and operating states, monitoring unit operations, receiving and processing incoming Telecommands (TCs), and producing and sending outgoing Telemetry reports (TMs). The TC/TM communication is regulated by a mission-specific tailored version of the Packet Utilisation Standard (PUS) ECSS-E-70-41A [4].

BepiColombo SIXS/MIXS OBSW is run in the DPU and consists of a Core Software (CSW) and four Application Software (ASW) modules such that each sensor unit has its own ASW module. The execution of the whole OBSW is essentially sequential, ensuring that an ASW module is executed only when called by the CSW.

5.1.4 Attitude and Orbit Control

Any spacecraft, aircraft or submarine needs attitude control (i.e., control over three-dimensional angular movement) as well as altitude control (i.e., control over height or depth). For spacecrafts, orbit control usually comprises (though is not necessarily limited to) altitude control.

Every spacecraft has a kind of the Attitude and Orbit Control System (AOCS), which periodically evaluates the current situation and decides on the required actions (with a typical period of one second). Information about the current situation is obtained from sensor and measurement units. The required actions are performed by actuator units on the basis of what they are commanded to do. Typically, it is the AOCS itself that is responsible for gathering information from the sensor and measurement units and commanding the actuator units.

Although the functionality of AOCS is highly mission-dependent, any two AOCS systems are likely to be very similar to each other at a certain abstraction level. In particular, it is often the case that many parts of software designed for one AOCS can be reused (as they are or with marginal modifications) in another AOCS. It should be noted that in itself reuse does not imply any compromise in quality. Every space software project has precise procedures for qualification of reused software.

The AOCS case study in DEPLOY considers reusable features of the AOCS software. To facilitate quick AOCS-related knowledge transfer to DEPLOY partners, SSF has defined a relatively simple but realistic AOCS. To do so, SSF has relied on its experience in the design, implementation, validation and maintenance of several projects.

5.1.5 *DEPLOY People at SSF*

A significant number of people have been involved in the DEPLOY project. Below we mention those who have made the most significant contributions to DEPLOY. Their experience outside SSF (especially with respect to formal methods) is mentioned when considered relevant.

- Dr. Timo Latvala, the Technical Director, has worked in technical and management positions at SSF, especially those related to software verification and validation, since 2007. Before that, he worked as a computer science researcher at Helsinki University of Technology for several years, and at the University of Illinois in Urbana-Champaign, USA, for one year, focusing on formal methods by model checking;
- Dr. Dubravka Ilić, a formal methods expert, has worked in technical and management positions at SSF, especially those related to software verification and validation, since 2007. Before that, she worked as a researcher in computer science at Åbo Akademi University for several years, focusing on refinement-based formal methods;
- Dr. Thomas Långbacka, a software expert, has worked in technical and management positions at SSF, especially those related to software design, verification and validation, since 1998. Before that, he worked as a computer science researcher at Åbo Akademi University and the University of Helsinki for several years, focusing on refinement-based formal methods;
- Laura Nummila, a software expert, has worked in technical and management positions at SSF, especially those related to software design, verification and validation, since 2006;
- Tuomas Räsänen, a software expert, has worked in technical positions at SSF, especially those related to software design, since 1999;
- Dr. Pauli Väisänen, a software expert, has worked in technical positions at SSF, especially those related to software design, verification and validation, since 2003. Before that, for several years he worked as a researcher in mathematics at the University of Helsinki and as a researcher in computer science, focusing on formal logic, at the Helsinki University of Technology;
- Dr. Kimmo Varpaaniemi, a formal methods expert, has worked in technical positions at SSF, especially those related to software verification and validation, since 2006. Before that, for several years he worked as a researcher in computer science, focusing on formal methods by model checking, at the Helsinki University of Technology. He is also a Docent in Formal Verification Methods for Parallel and Distributed Systems at the Aalto University School of Science.

5.2 Approach and Achievements

Throughout the project, our approach was to experiment with the formal modelling and development of several typical space applications in order to understand how

to reap the benefits of rigorous engineering and to pave the path towards integrating it into the existing development process. As a result of the project, SSF has gained significant insights into both strengths and drawbacks of a formal refinement-based approach, thus acquiring rich expertise in this area. Although refinement-based development is currently not part of everyday development practice at SSF, we believe that within DEPLOY we have done a lot of useful technical work addressing both theoretical and tooling issues of the applied formal methods as well as space domain-specific problems. Most of this work is described below.

5.2.1 Modelling of BepiColombo SIXS/MIXS OBSW Requirements

In autumn 2008 and spring 2009, SSF created Event-B models for a considerable subset of requirements of BepiColombo SIXS/MIXS OBSW, with a focus on the handling of telecommands and production of telemetry packets. The unit management in the OBSW was modelled to some extent, too, with particular attention to the management of operational modes in the CSW and the ASW modules. The resulting models are available as the Event-B project BepiColombo_Models_v5.0 [17]. The modelling process is also informally described in the DEPLOY deliverable D20 [1].

The formal modelling of SIXS/MIXS OBSW helped us find some problems with its requirements. As a result, the problematic requirements were modified. Since formal modelling requires careful inspection of requirements, it might be hard to draw the line between the impact of the requirements inspection and the actual modelling based on the given requirements. However, it is fair to conclude that DEPLOY had an impact on the requirements. The Event-B modelling of OBSW has also resulted in useful feedback to the Event-B community on how to improve scalability of the approach and usability of the Rodin platform in particular.

In 2010, SSF and the University of Southampton continued the Event-B modelling of OBSW. These experiments focused on improving the modelling methodology per se and did not lead to further requirements changes. The work undertaken was aimed at constructing better structured models by utilising the modular (see Sect. 5.2.5) and decompositional extensions of the Event-B language. Modelling using the modular extension was done by SSF, whereas modelling using the decompositional extension was done by the University of Southampton. The Event-B projects BepiColombo_Models_v6.4 (by SSF) [29] and BepiColombo_Soton_v18.0 (by Southampton) [10] are the final public versions of these models. The work on the modular extension also resulted in useful tool feedback and prompted appropriate tool improvements. We shall return to these issues in Sect. 5.2.5. The work on the decompositional extension has been done independently of SSF, except for the use of the SSF's Event-B projects as de facto specifications for the behaviour of the OBSW. The paper [9] describes this work in detail.

The formal development of OBSW has also demonstrated that, for some systems, modelling their dynamic behaviour poses a major challenge. For instance,

OBSW did not have complex safety constraints that could have been represented by the corresponding Event-B invariants. This was not surprising for SSF engineers because a typical software requirement expresses what the software should do in a given situation. Event-B events alone suffice for modelling such requirements. Formal modelling of systems similar to OBSW should be focused more on reasoning about liveness-type properties. Indeed, as it became clear later, it proved to be more interesting to analyse the dynamic behaviour of OBSW using the UPPAAL model checker.

5.2.2 Modelling of Attitude and Orbit Control Systems

In spring 2009, SSF decided to investigate formal modelling and development of Attitude and Orbit Control systems (AOCS), as it was possible that SSF would be involved in the development of this type of system in several future projects.

To focus on reusable features rather than on mission-specific details, a simple but realistic AOCS was specified. Instead of producing a textual requirements description, SSF decided to compose a system description from generalised modules that had been previously implemented in the Ada programming language. In this way, the resulting requirements description was produced in less than one month, even though it had to be revised a few times thereafter. SSF's work on modelling AOCS in Event-B was carried out within a few months and completed in January 2010. The official result of this work is Event-B project DepSatSpec015Model000 [30], and the corresponding requirements document expressed in Ada is "DEPLOY Satellite (an Attitude and Orbit Control System) Specification, Version 15" [26]. The corresponding verbal specification [27] was completed much later in December 2010, and is still not necessarily better than its Ada counterpart. (The verbal specification is inherently more ambiguous, and there has been no review process for resolving these ambiguities.)

It was observed that the engineers were reluctant to follow the refinement approach when the system requirements were provided as code. This could be because it is hard to justify building abstractions of well-known behaviour. The approach taken by SSF was to model the given Ada representation in a statement-by-statement manner. This was based on the conviction that code modelling had been, for many years, a mainstream approach in computer-aided verification. However, in January 2010, SSF decided to discontinue the statement-by-statement modelling because it became clear that modelling and proving the "mirror" model of the Ada code would require a very large amount of work.

A careful requirements analysis required to construct formal models has resulted in several modifications of the Ada code. However, SSF felt that the Ada code provides a high degree of accuracy in describing the system behaviour and hence the mathematical modelling relying on invariants and gradual refinement is excessive (as distinct from the BepiColombo SIXS/MIXS OBSW requirements, which in a strict mathematical sense do not ensure the intended behaviour, while still giving sufficient guidelines for lower design levels).

As an alternative, Åbo Akademi University and Newcastle University have proposed a method for developing mode-rich systems by refinement (see [11] and the papers [14, 15] and [18]). The method was inspired by the AOCS requirements. It is based on the instantiation and refinement of a generic specification pattern for a mode manager. The pattern defined as a generic module interface captures the essential structure and behaviour of a component and can be instantiated by component-specific data to model a mode manager at any layer of the system hierarchy. The overall process can be seen as a stepwise unfolding of architectural layers. At every step of this unfolding, its correctness is proved, and the mode consistency between two adjacent layers is verified. Such incremental verification allows us to guarantee global mode consistency without having to check the property for the whole architecture at once.

AOCS is a centralised system, i.e., a high-level mode manager is responsible for monitoring states of components and changes to the global mode. A distributed AOCS [28] is a modification of a centralised AOCS that relies on a mode synchronisation protocol that several mode managers run to achieve consistent mode transitions. The engineers at SSF decided to design a mode synchronisation protocol from scratch rather than rely on any well-known consensus protocols.

Since the distributed AOCS has complex dynamic behaviour, SSF relied on model-checking facilities of ProB [21] to verify the system behaviour. Verification was performed for three modes and two mode managers. The case of three modes and three managers turned out to be beyond the capabilities of explicit-state model checkers such as ProB. To accomplish system verification, SSF decided to experiment with model checker NuSMV [20]. We shall consider this model checking work in Sect. 5.2.6.

5.2.3 Statistics of Major Event-B Models by SSF

Let us now consider some quantitative aspects of Event-B projects

– BepiColombo_Models_v5.0 [17],
– BepiColombo_Models_v6.4 [29] and
– DepSatSpec015Model000 [30]

(see Sects. 5.2.1 and 5.2.2), which can be considered major Event-B model developments by SSF in DEPLOY.

In each of these developments, Event-B models form a chain where in each pair of adjacent machines, the succeeding machine is a superposition refinement of the preceding one, with the consequence that the preceding one can be reproduced from the succeeding machine simply by removing some "details". Whenever the amount of activity in such models can be measured by counting the number of events, it in a sense suffices to count the number of events in the most refined machine.

BepiColombo_Models_v5.0 consists of nine Event-B machines and ten Event-B contexts. The number of proof obligations for these machines and contexts is exactly 1,000, of which 683 (=68.3 %) were discharged automatically and the remain-

ing 317 (=31.7 %) manually. The number of events in the most refined machine is 63.

BepiColombo_Models_v6.4 consists of two Event-B machines, ten Event-B contexts and three Event-B module interfaces. The number of proof obligations for all these machines, contexts and interfaces is 930, of which 560 (≈60.2 %) were discharged automatically and the remaining 370 (≈39.8 %) manually. (Note that, at the time of writing, the relevant statistics displayed by the latest version of the Rodin platform are somewhat misleading by not taking the module interfaces into account properly.) The number of events in the most refined machine is 26.

DepSatSpec015Model000 consists of 14 Event-B machines and two Event-B contexts. The number of proof obligations over all these machines and contexts is 1859, of which 1310 (≈70.5 %) were discharged automatically and the remaining 317 (≈29.5 %) manually. The number of events in the most refined machine is 230.

The above information on proof obligations is saved in archives [17, 29] and [30]. In general, any generated set of proof obligations depends on specific versions of the Rodin platform and on the plug-ins used in the generation, whereas the possibility of getting a proof obligation automatically discharged depends on the provers used and on the configurations of proof tactics. Even so, some experimental evidence suggests that, no matter which versions of the Rodin platform and plug-ins are used, it is difficult to radically increase the percentage of automatically discharged proof obligations in these models.

The number of manually discharged proof obligations is a somewhat poor measure of the amount of proof work because the difficulty of discharging a proof obligation is highly case-dependent. The total number of "user actions" in interactive proofs would be a considerably better measure, but SSF is not aware of any existing direct tool support for obtaining such a number. The work required to prepare such actions would still not be taken into account. The time used in interactive proofs is not an ideal measure either because it is affected by experience, concentration, motivation and so on.

Similarly, the number of events (or the number of event guards or event actions) is a poor measure of the amount of work because the impact of an event (or guard or action) is highly case-dependent. The number of names, operator symbols and numeric literals in event definitions would be a considerably better measure, but SSF is not aware of any existing direct tool support for obtaining such numbers. The details "delegated" to Event-B contexts or Event-B module interfaces would still get ignored, as would variations in their importance. The amount of work "hidden" by using Event-B contexts seems inherently difficult to measure.

5.2.4 Assessment of Event-B and the Rodin Platform

Event-B is essentially a guarded command-style language where events and guards can be specified using a simply typed first-order logic. Since the language is event-based, it is eminently suited for modelling behaviour that can be captured well with state machines. Capturing algorithmic computation requires more effort.

An Event-B model has two parts: the context and the machine. Roughly speaking, the context captures static definitions such as types, while the machine models behaviour. Refinement can be used to split the modelling into several successive steps, where each model is provably a behavioural refinement of the more abstract machines. This facilitates model management as well as proof complexity.

The Rodin platform is the main tool used for Event-B modelling and proof activities in DEPLOY. Though Event-B is in principle independent of tools, its implementation in the Rodin platform is dominant in the sense that there is no competing Event-B tool.

SSF has experienced a lack of scalability when using the Rodin platform. The platform has significantly improved in the course of DEPLOY but certain problems still remain. To a certain degree, the problems of large memory consumption and lack of responsiveness have been alleviated. However, the problem of "disappearing auto-provability" (i.e., failing to automatically discharge a proof obligation that in other circumstances was automatically discharged) still remains.

Selecting the right subset of hypotheses needed for interactive proving tends to require large amounts of time. Although sometimes the Rodin platform has certain difficulties in automatically proving rather simple goals, the work on linking it with more powerful external theorem provers, such as Isabelle, promises improvements in this direction.

The most significant problem is the lack of proofs verifying consistency of axioms. In cases of inconsistency, even a very careful user may produce arbitrarily many proofs without noticing the inconsistency, as it is not necessarily likely that an inconsistent combination of axioms and/or guards will get used in a single proof. The Rodin platform should definitely have proper support for proving consistency of axioms and guards.

Despite certain improvements made in the course of DEPLOY, the type system in Event-B is not convenient. For example, there is no proper support for abstract data types, i.e., types defined by means of inductive axioms, as it is surprisingly difficult to produce any inductive proof in the Rodin platform. The theory plug-in [8] is not an improvement in terms of inductive proving; rules defined using the plug-in do not provide more "computational power" than the corresponding ordinary theorems.

Without language extensions, the lack of modular or decompositional structures in Event-B is not only a readability problem but also creates a need for significant additional modelling and proving effort that could be avoided with proper use of modularity. Due to the combined effort of Newcastle and Åbo Akademi University, a modular extension of Event-B and the associated plug-in of the Rodin platform have now been created (they will be considered in detail in Sect. 5.2.5). There are also a few decompositional extensions of Event-B supported by the corresponding plug-ins, e.g., those designed at the University of Southampton and used in the space sector work mentioned in Sect. 5.2.1.

A trained person can construct an Event-B model of medium complexity in a few weeks. However, SSF engineers believe that it is more rewarding to spend the same amount of time writing a program in a high-level programming language, since it results in an "end-product".

Let us end the assessment with some positive observations:

- Manual proofs can often be mechanised using Event-B and the Rodin platform;
- Many things can be expressed in Event-B much more compactly than in many other formal languages;
- Ensuring correctness of proofs and increasing the degree of automation in proofs have been a high priority in the development of the Rodin platform;
- Many useful plug-ins [8] for the Rodin platform have been created in DEPLOY;
- The latest version of the Rodin platform is certainly much more convenient to use than the version that was available initially.

5.2.5 Modularisation

The modularisation plug-in for the Rodin platform, designed at Newcastle and Åbo Akademi University (presented, among others, in papers [12] and [16]), provides facilities for structuring Event-B projects into logical units of modelling, called modules. The module concept is very close to the notion of classical B imports. However, unlike a conventional project, a module comes with an interface. An interface defines the conditions under which a module may be incorporated into another project (that is, another module). The plug-in follows an approach where an interface is characterised by a list of operations specifying the services provided by the module. These operations allow a module to be integrated into the main project using an intuitive procedure call notation.

Below are some of the reasons for splitting a project into modules.

- Structuring large specifications: it is difficult to read and edit large models; there is also a limit to the size of model that the Rodin platform can handle comfortably, and thus decomposition is an absolute necessity for large scale developments;
- Decomposing proof effort: splitting helps split the verification effort. It also helps us reuse proofs; it is not unusual to go back in the refinement chain and partially redo abstract models. Normally, this would invalidate most proofs in the dependent components. Model structuring helps localise the effect of such changes;
- Team development: large models can be developed only by a (often distributed) developer team;
- Model reuse: modules can be exchanged and reused in different projects. The notion of the interface makes it easier to integrate a module into a new context.

An experiment with applying modularisation plug-in to modelling BepiColombo SIXS/MIXS OBSW, carried out by SSF in August and September 2010, focused on some of the requirements that had been considered in earlier non-modular experiments. The original goals of the experiment were as follows:

- Systematic isolation of activity details and related conditions in modules in such a way that the machines using the modules do not replicate much of what is expressed inside them;
- Making descriptions of the considered behaviour as precise as in the most detailed non-modular Event-B model available;

- Avoiding massive atomic activities. Long chains of atomic activities do realistically model concurrency;
- Consideration of "module integration invariants". Such invariants refer to variables of more than one module;
- Reasonable total proof effort (including time spent on "iterative optimisation") without compromising the goals above.

The final Event-B project of the experiment can be considered as sufficiently meeting all the above-mentioned goals, except possibly the proof effort reasonability goal. However, the proof effort was to a certain extent more reasonable than in some earlier Rodin platform experiments.

Much time in the experiment was spent on recognition and circumvention of bugs and dealing with other undesirable features. The problems that were reported during the experiment were solved in later releases of the plug-in. Since October 2010, the plug-in has been in good shape with respect to the features used in the experiment.

By following certain modelling conventions it is possible to significantly improve the usability of the modularisation plug-in. Designers of the plug-in recommend the following conventions:

- Avoiding very large and complicated operation postconditions, especially involving existential quantifiers, to simplify proofs. In general, complex postconditions can be simplified by introducing additional module variables and invariant properties of these variables.
- Refraining from using operation calls to model returned exceptions. To achieve the same effect, the preconditions of calling events could be strengthened by checking the external module variables. Rather than using returned composite values, which can include the status indicating success or a particular occurred exception, additional external module variables storing such a status of the latest call can be introduced.
- Avoiding generating new values supplied by the environment inside the operation postconditions. The problem can be circumvented by introducing additional local variables in a calling event and then forwarding the value of these local variables as extra parameters of an operation call. Alternatively, module processes can be used for modelling such inputs from the environment.

Even when these guidelines are followed, usability of the plug-in suffers from a "macro-style" approach. This refers to the fact that in interactive proving, the user deals with the output of a translator and is expected to understand that output as if it were the original form. However, solving this problem is likely to involve a lot of work, and it is not specific to the modularisation plug-in: for example, the record type plug-in [8] of Rodin platform has essentially the same problem.

5.2.6 Model Checking

It is often the case that a property of interest is expressible as a temporal logic formula but not as an invariant. Such properties are beyond the scope of proper Event-B proof methodology, but not necessarily beyond the scope of Event-B. ProB

(as a stand-alone tool [21] or as a plug-in of the Rodin platform) provides model checking of Linear Time Temporal Logic (LTL) and Computation Tree Logic (CTL) formulas such that the syntax of atomic formulas is the same or almost the same as the syntax of predicate expressions in Event-B.

In the autumn of 2008, SSF used ProB to search for deadlocks in the "root machine" of an Event-B model based on the BepiColombo SIXS/MIXS OBSW requirements. A deadlock was indeed found, and the machine was revised accordingly. Some attempts to use ProB for more detailed machines were made, but proved to be cumbersome since it was not clear how to compute consistent instances of the Event-B contexts used.

In the autumn of 2011, SSF used ProB in the checking of LTL formulas that concern the "three modes and two managers" case of the mode synchronisation protocol designed for the distributed AOCS (see Sect. 5.2.2). Except for the formulas, the protocol descriptions were written in Event-B using the Rodin platform. The ProB plug-in was needed to produce machines for the stand-alone ProB used for the actual model checking. Though the plug-in itself has model checking facilities, SSF chose to use the stand-alone tool to avoid unnecessary feature interaction and to take advantage of the later improvements available for the stand-alone tool but not for the plug-in.

The Event-B description of the mode synchronisation protocol was revised several times as ProB reported counterexamples of expected properties. The errors found were modelling ones, i.e., mismatches between the verbal specification and the Event-B description. (The verbal specification itself was later revised to eliminate an error identified during the inspection.) Our experiments with ProB [31] are representative of the modelled aspects and investigated properties.

Some attempts were made to use ProB in the "three modes and three managers" case (see Sect. 5.2.2). However, ProB tended to run out of memory. Once the developers had been informed about the problem, some support for memory-efficient state space generation was implemented in ProB. In model checking, it is often wise to consider several formalisms and tools. Between December 2011 and January 2012, SSF experimented with the symbolic model checker NuSMV [20] to work on verifying invariants defined for Event-B models (see Sect. 5.2.2). Several model-checking processes were run in several processors for several weeks. So far the results have shown that a desired property holds in all states that are reachable from the initial state within an expressed number of steps. Although NuSMV has options for concluding whether a check is complete with respect to the state space, it may be impossible to reach such a conclusion before NuSMV runs out of memory (for acceptable reasons). Our experiments with NuSMV [32] are representative of the modelled aspects and investigated properties.

5.2.7 Real-Time

A large number of dependable embedded systems have stringent real-time requirements imposed on them. Analysis of real-time behaviour is usually conducted at

the implementation stage. However, it is desirable to evaluate real-time properties early in the development cycle, i.e., at the modelling stage. Using data processing of BepiColombo SIXS/MIXS OBSW as a working example, the space sector team in DEPLOY has studied ways of augmenting Event-B modelling with verification of real-time properties in the model-checking tool UPPAAL [25]. In the approach described in detail in the report [13], a process-based view is extracted from an Event-B model and translated into a timed automaton readable by UPPAAL.

Essentially, this approach consists of three main stages: (1) “traditional” refinement in Event B; (2) the definition of an explicit concurrency model called a Process View (PV); and finally, (3) the creation of a timed-automata model and verification of the desired real-time constraints. Each part of this modelling chain has a specific purpose. The Event-B modelling automated by the Rodin platform facilitates reasoning about functional correctness of the system under construction. A PV model explicitly defines processes and their synchronisation. It is a projection of an Event-B specification that allows us to represent the targeted system architecture and the corresponding communication infrastructure. Finally, a timed automata system model enables verification of the desired real-time properties.

5.2.8 Supporting RAMS Through FMEA

The behaviour of a complex system is often structured using the notion of operational modes. Mode consistency and fault tolerance usually have a dominant role in specifying pre- and post-conditions for transitions between operational modes. Using Event-B as formalism and SSF’s verbal specification of a centralised AOCS as a working example, researchers at Åbo Akademi University have studied ways of deriving such conditions from the Failure Modes and Effects Analysis (FMEA) and of preserving mode consistency in machine refinements. They have defined a preliminary methodology which can be applied to control systems in various application domains, to a large extent independently of these. This work is considered in papers [19] and [22].

Ensuring dependability and, in particular, fault tolerance of complex control systems is a challenging engineering task. To cope with the complexity, control systems are often developed in a layered fashion, which provides the designers with a convenient mechanism for structuring system behaviour according to the identified architectural layers. Moreover, the behaviour of a complex system is frequently described and reasoned about using the notion of operational modes. While designing layered mode-rich systems, developers need to ensure mode consistency and guarantee that the mode logic accommodates fault tolerance.

It was proposed to conduct the FMEA of each particular mode to identify mode transitions required to implement fault tolerance. The resulting FMEA tables relate a particular mode with various failures (failure modes) as well as ways to detect them and the necessary remedial actions. The latter ones are described in terms of new target modes the system should roll back to so as to attempt its recovery. Recovery may also involve dynamic reconfiguration of the system by switching

some of its failed hardware components to spare ones if possible. Each remedial action is associated with necessary conditions linking the states and detected errors of the monitored components of the lower layer.

It was demonstrated that, by encapsulating the details of dynamic reconfiguration carried out in response to errors, we can arrive at an efficient mechanism for structuring fault tolerance according to the identified architectural layers.

5.2.9 Training

DEPLOY training in the space sector is considered in deliverables D5 [3] and D39 [2] as well as in report [23].

At the beginning of DEPLOY, ETH Zurich and Åbo Akademi University taught SSF how to use the Event-B modelling and proof methodology in the Rodin platform. Four people from SSF attended the Rodin block course by ETH from 9 to 11 April, 2008. Three of them and a fifth person attended training by Åbo Akademi University at SSF on 17 June, 2008. Due to the research background at Åbo Akademi University, a sixth person was already familiar with Event-B and the Rodin platform.

SSF has mostly positive memories of these training sessions, though it is fair to say that no short training should ever be expected to be equivalent to genuine experience of use. In order to ensure such experience for industrial partners, DEPLOY reserved a few months in 2008 for a mini-pilot that in the case of the space sector consisted of modelling a simple telecommand verification service inspired by BepiColombo SIXS/MIXS OBSW. The specification of the service was written by one of the designers of the OBSW at SSF. Two SSF employees worked on the modelling of the mini-pilot intensively and have been responsible for most of the Rodin platform activities at SSF since then. The mini-pilot was a useful learning experience for the academic partners too, as they created their own models of this service and in that way became familiar with some aspects of space software design.

From January to June 2010, two persons from SSF with no prior formal method experience trained themselves to use Event-B and the Rodin platform. The Rodin platform with plug-ins for text editing, modularisation and records was used as the Event-B work environment. The training material mostly consisted of the Event-B wiki, teaching material from the above mentioned block course, and a general educational material about formal logic. They also received help from DEPLOY space sector team members at SSF, Åbo Akademi University and Newcastle University.

The following conclusions were drawn from this internal training:

- It took effectively about one working month (spread over several calendar months as the training was part-time) to learn to work with Event B and Rodin in a reasonably independent fashion. More time would have been needed to become sufficiently familiar with the modularisation and record extensions.
- The documentation used was considered largely satisfactory. It was still clear there was a need for more detailed, searchable and up-to-date tool information,

as well as for Event-B reference material that would cover the entire syntax, including proof obligation notations, even at the level of examples.

- The basic features of the Rodin environment proved to be easy to use. Despite certain problems with its use, the Camille text editor was preferred to Rodin's default model editor. As in any sufficiently long experience of using Rodin, non-response phenomena and other problematic behaviours were observed.

5.3 Conclusions

SSF is seriously considering the use of formal methods though currently they are not used in the company's projects. More evidence is required to ensure that the use of formal modelling leads to an improvement in customer satisfaction or reduces costs.

Obviously, a correct-by-construction approach has demonstrated its usefulness for structuring complex requirements, for deriving the system architecture and for formal verification of safety properties. Nevertheless, we believe that the way to successfully deploy formal engineering techniques in the development practice in the space domain could be through strengthening links with "traditional" software development. Further research and automation are needed to facilitate a smooth transition from architectural modelling to formal modelling and verification. Moreover, formal verification should provide prompt and comprehensive feedback at early development stages to facilitate design space exploration and the assessment of design alternatives. Finally, domain-specific modelling guidelines and tool support for automating a significant part of the refinement chain are needed so that the refinement approach can be understood more easily and adopted more quickly.

References

1. DEPLOY. Deliverable D20 D3.1—Report on pilot deployment in the space sector (January 2010)
2. DEPLOY. Deliverable D39 D3.2—Report on enhanced deployment in the space sector (August 2011)
3. DEPLOY. Deliverable D5 JD1: Report on knowledge transfer (January 2009)
4. ESA. ECSS-E-70-41A: Space engineering: Ground systems and operations—Telemetry and telecommand packet utilisation. European Space Agency Requirements and Standards Division, Noordwijk ZH, The Netherlands (January 2003). http://www.ecss.nl/
5. ESA. ECSS-E-ST-40C: Space engineering—Software. European Space Agency Requirements and Standards Division, Noordwijk ZH, The Netherlands (March 2009)
6. ESA. ECSS-Q-ST-80C: Space product assurance—Software product assurance. European Space Agency Requirements and Standards Division, Noordwijk ZH, The Netherlands (March 2009)
7. ESA. Factsheet: BepiColombo (February 2012). http://www.esa.int/esaSC/SEMNEM3MDAF
8. Event-B. Documentation wiki: Rodin plug-ins (February 2012)

9. Fathabadi, A.S., Rezazadeh, R., Butler, M.J.: Applying atomicity and model decomposition to a space craft system in Event-B. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) NASA Formal Methods, Proceedings, Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18–20, 2011. Lecture Notes in Computer Science, vol. 6617, pp. 328–342. Springer, Berlin (2011)
10. Fathabadi, A.S., Rezazadeh, R., Butler, M.J.: Event-B project BepiColombo_Soton_v18.0 (Rodin archive of space system) (October 2010). http://eprints.ecs.soton.ac.uk/22048/
11. Iliasov, A., Laibinis, L., Troubitsyna, E.: An Event-B model of the attitude and orbit control system (March 2010). http://deploy-eprints.ecs.soton.ac.uk/213/
12. Iliasov, A., Laibinis, L., Troubitsyna, E., Romanovsky, A.: Formal derivation of a distributed program in Event B. In: Qin, S., Qiu, Z. (eds.) Formal Methods and Software Engineering, Proceedings, 13th International Conference on Formal Engineering Methods, ICFEM 2011, Durham, UK, October 26–28, 2011. Lecture Notes in Computer Science, vol. 6991, pp. 420–436. Springer, Berlin (2011)
13. Iliasov, A., Laibinis, L., Troubitsyna, E., Romanovsky, A., Latvala, T.: Augmenting Event B modelling with real-time verification. Technical report 1006, Turku Centre for Computer Science, Turku, Finland (April 2011)
14. Iliasov, A., Troubitsyna, E., Laibinis, L., Romanovsky, A., Varpaaniemi, K., Ilić, D., Latvala, T.: Developing mode-rich satellite software by refinement in Event B. In: Kowalewski, S., Roveri, M. (eds.) Formal Methods for Industrial Critical Systems, Proceedings, 15th International Workshop, FMICS 2010, Antwerp, Belgium, September 20–21, 2010. Lecture Notes in Computer Science, vol. 6371, pp. 50–66. Springer, Berlin (2010)
15. Iliasov, A., Troubitsyna, E., Laibinis, L., Romanovsky, A., Varpaaniemi, K., Väisänen, P., Ilić, D., Latvala, T.: Verifying mode consistency for on-board satellite software. In: Schoitsch, E. (ed.) Computer Safety, Reliability, and Security: 29th International Conference, Proceedings, SAFECOMP 2010, Vienna, Austria, September 14–17, 2010. Lecture Notes in Computer Science, vol. 6351, pp. 126–141. Springer, Berlin (2010)
16. Iliasov, I., Troubitsyna, E., Laibinis, L., Romanovsky, A., Varpaaniemi, K., Ilić, D., Latvala, T.: Supporting reuse in Event B development: Modularisation approach. In: Frappier, M., Glässer, U., Khurshid, S., Laleau, R., Reeves, S. (eds.) Abstract State Machines, Alloy, B and Z: Second International Conference, Proceedings, ABZ 2010, Orford, Québec, Canada, February 22–25, 2010. Lecture Notes in Computer Science, vol. 5977, pp. 174–188. Springer, Berlin (2010)
17. Ilić, D., Varpaaniemi, K.: Event-B project BepiColombo_Models_v5.0 (May 2009). http://deploy-eprints.ecs.soton.ac.uk/136/
18. Lopatkin, I., Iliasov, A., Romanovsky, A.: Rigorous development of dependable systems using fault tolerance views. In: Dohi, T., Čukić, B. (eds.) Proceedings of the 22nd IEEE International Symposium on Software Reliability Engineering, ISSRE 2011, Hiroshima, Japan, November 29–December 2, 2011, pp. 180–189. IEEE Computer Society Press, Los Alamitos (2011)
19. Lopatkin, I., Iliasov, A., Romanovsky, A., Prokhorova, Y., Troubitsyna, E.: Patterns for representing FMEA in formal specification of control systems. In: Agarwal, A., Gokhale, S., Khosoftaar, T.M. (eds.) Proceedings of the 2011 IEEE 13th International Symposium on High-Assurance Systems Engineering, HASE 2011, Boca Raton, FL, USA, November 10–12, 2011, pp. 146–151. IEEE Computer Society Press, Los Alamitos (2011)
20. NuSMV. A new symbolic model checker (February 2012). http://nusmv.fbk.eu/
21. ProB. Animator and model checker (February 2012). http://www.stups.uni-duesseldorf.de/ProB/index.php5/Main
22. Prokhorova, Y., Laibinis, L., Troubitsyna, E., Varpaaniemi, K., Latvala, T.: Derivation and formal verification of a mode logic for layered control systems. In: Thu, T.D., Leung, K.R.P.H. (eds.) Proceedings of the 18th Asia-Pacific Software Engineering Conference, APSEC 2011, Ho Chi Minh City, Vietnam, December 5–8, 2011, pp. 49–56. IEEE Computer Society Press, Los Alamitos (2011)

23. Räsänen, T., Nummila, L.: DEPLOY training evaluation document (September 2010). http://deploy-eprints.ecs.soton.ac.uk/314/
24. RTCA. DO-178B: Software Considerations in airborne systems and equipment certification. Radio Technical Commission for Aeronautics, Washington, DC, USA (January 1992)
25. UPPAAL. An integrated tool environment for modelling, validation and verification of real-time systems (February 2012)
26. Väisänen, P., Varpaaniemi, K.: DEPLOY satellite (an attitude and orbit control system) specification, version 15 (January 2010). http://deploy-eprints.ecs.soton.ac.uk/167/
27. Varpaaniemi, K.: DEPLOY work package 3 attitude and orbit control system software requirements document, DEP-RP-SSF-R-005, issue 1.0 (December 2010). http://deploy-eprints.ecs.soton.ac.uk/266/
28. Varpaaniemi, K.: DEPLOY work package 3 software requirements document for a distributed system for attitude and orbit control for a single spacecraft, DEP-RP-SSF-R-006, issue 1.3. (October 2011)
29. Varpaaniemi, K.: Event-B project BepiColombo_Models_v6.4 (September 2010). http://deploy-eprints.ecs.soton.ac.uk/244/
30. Varpaaniemi, K.: Event-B project DepSatSpec015Model000 (January 2010). http://deploy-eprints.ecs.soton.ac.uk/168/
31. Varpaaniemi, K.: Event-B projects DSAOCSSv002 and DSAOCSSv003 with special files for ProB Classic (October 2011). http://deploy-eprints.ecs.soton.ac.uk/331/
32. Varpaaniemi, K.: Some NuSMV experiments on the mode synchronization protocol in DSAOCSS (January 2012). http://deploy-eprints.ecs.soton.ac.uk/362/

Chapter 6
Business Information Sector

Sebastian Wieczorek, Vitaly Kozyura, Wei Wei, Andreas Roth, and Alin Stefanescu

Abstract Enterprise software helps modern corporates to automate their businesses in order to run efficiently and economically. We report a story of successful introduction of formal methods to business software development. This deployment came in three phases: modelling, formal verification and model-based testing. For each phase, we describe a few representative deployment cases in detail, and discuss the problems that we encountered and the decisions that we had to make. The work discussed here was carried out to focus on issues of interest to SAP, the world's leading provider of enterprise software.

6.1 Introduction to the Business Sector

Business application software has nowadays become indispensable to businesses because it provides the backbone and drives their activities through automation. For many areas, including manufacturing, supply chains, sales and human resources, data and services of various organisational units across the entire company need to be consistently integrated. With complex configuration options and business processes, it is no wonder that such software systems are very large and complex. Furthermore, business software constantly evolves and adapts to fast-changing business environments and requirements.

S. Wieczorek (✉) · V. Kozyura · W. Wei · A. Roth
SAP AG, Darmstadt, Germany
e-mail: sebastian.wieczorek@sap.com

V. Kozyura
e-mail: v.kozyura@sap.com

W. Wei
e-mail: wei01.wei@sap.com

A. Roth
e-mail: andreas.roth@sap.com

A. Stefanescu
University of Pitesti, Pitesti, Romania
e-mail: alin.stefanescu@upit.ro

A. Romanovsky, M. Thomas (eds.), *Industrial Deployment of System Engineering Methods*, DOI 10.1007/978-3-642-33170-1_6,
© Springer-Verlag Berlin Heidelberg 2013

In response to these challenges, two important methodologies, namely *Service-Oriented Architecture* (SOA) and *Model-Based Development* (MBD), have been adopted for business software development in the last decade. The essence of SOA is to break the monolithic structure of large software up into smaller business components, which are then presented as services easily composed to meet increasingly more complex business needs. The development of service-based systems is layered as follows:

- **Development of functional units** that encapsulate a basic piece of computation.
- **Bundling of functional units into components** with the aim of providing reusable composable services.
- **Definition of business processes** by composing services to realise end-to-end business scenarios of enterprises.

MBD documents different aspects of software as models at every development stage, which enables early prototyping and detection of potential errors to avoid a drastically more expensive correction later. A software model retains only the details important to the corresponding design emphasis, and gets rid of any irrelevant information.

Correct functioning of business software is very important because failures could cause great financial losses. Formal methods have attracted growing attention for their capabilities to offer unambiguous semantics and to prove correctness rigorously. Thanks to the rich repository of models already available to us, it becomes less difficult and makes more sense to apply formal methods to the development of business software. It is less difficult because models are usually much smaller compared to the final implementation code, so they can be better handled with formal methods. It makes more sense because, on the one hand, models are used to guide subsequent development, and therefore their correctness is critical. On the other hand, models are perfect for deriving tests, since they capture the essence of how software is supposed to behave. The focus of our deployment of formal methods is on both formal verification and model-based testing.

One important objective during our deployment is to achieve a high degree of automation. Ideally, the application of formal methods should be completely hidden from designers and developers, because in our industry they should not be expected to understand or master formal methods. The high expense of training required is only part of the problem. Manual applications of formal methods, such as manual proofs, are usually very time- and resource-consuming, while the results are often not reusable due to frequent model changes. This would not only further increase the cost of software development, but also interrupt or even delay the development process to an extent that can no longer be tolerated. Therefore, we need to either make reasonable trade-offs (e.g., by sacrificing expressibility of modelling languages in favour of verifiability) or enhance the existing formal methods with improved automation. Finally, hiding formal modelling behind the surface of the existing MBD abstractions not only allows for seamless integration with the current development processes but also makes it easy to re-use the existing model contents.

6.2 Modelling

The early and wide adoption of MBD at SAP resulted in a vast collection of models that cover almost every aspect of software design, from high-level descriptions of business processes to low-level behavioural model for business objects. Some models are described using public standard modelling languages or their variants, and others using completely proprietary languages. We identified two main challenges that we needed to address at the modelling phase.

First, even though compared to implementation code models are small, their size and complexity could still be overwhelming for a formal analysis tool. To overcome this, we decided to compromise by leaving out certain modelling features that are deemed either inessential or too expensive to be analysed using formal methods. We also tried to break up monolithic model structures into smaller components/layers in order to reduce difficulties in verification, as seen in the cases of message choreographies (using layered design) and business processes (using model decomposition).

Second, most industrial modelling languages lack formal semantics. For example, Business Process Model and Notation (BPMN) has many elements with very vague and ambiguous interpretations in its official documentation. Even in case of models whose semantics is more or less clear, we should avoid applying formal analysis techniques directly to models represented in a variety of individual modelling languages. A much better, scalable practice would be to translate these languages into an intermediate language, and use the formal analysis methods on it.

Therefore, we decided to translate all models into a common formal language in which any future formal analysis would be performed. This has the additional advantage of making it possible to formally capture and verify the relations between different models. This is particularly desirable because in this way we are able to guarantee and maintain consistency across different design aspects. In the context of the DEPLOY project, we decided to use Event-B as the common formal language because of its powerful tool support by the Rodin platform.

6.2.1 Message Choreography Modelling

At the beginning of our work, we identified a missing layer in the modelling stack. While there are higher-level models for business processes and lower-level models for business objects, there are no models to describe message protocols for communication between business objects. There was static communication information, such as service interfaces and message formats, scattered throughout various documents and sources. However, there was virtually no documentation about the dynamic aspect of communication, i.e., message sequences that would occur at runtime. Therefore, we decided to come up with message choreography models (MCMs) that would provide all this information in a unified and consistent way. We started with a careful investigation of the state of the art in choreography modelling and matched it with the initial requirements gathered from developers. We

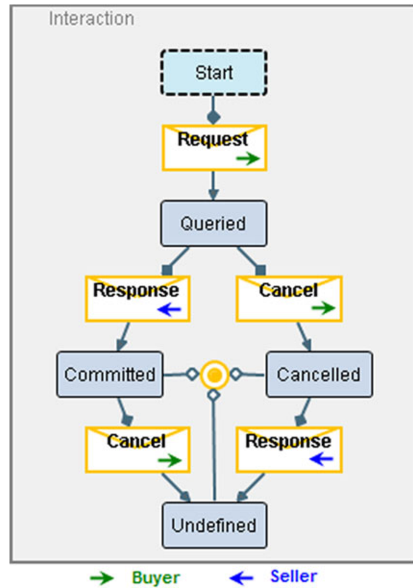


Fig. 6.1 A global choreography model

tried using several existing choreography languages, such as WS-CDL and BPMN-Choreography, to build simple models, and found that they were not suitable for our purposes. Finally, we adapted a proprietary language used internally to describe how business components call each other's services, and replaced business operation calls with message interactions between components. The result is the MCM language [9, 15], as shown in Figs. 6.1 and 6.2.

An MCM consists of a global choreography model that shows all possible sequences of messages exchanged at runtime, and a pair of local partner models that describe how each communication partner sends or receives messages with additional local constraints. In addition, for local partner models, we need to specify a property of the communication channel between partners: whether it is Exactly-Once (no message duplication, no message loss, but no message order guaranteed), Exactly-Once-In-Order (message order guaranteed), or something else. The global choreography model has no information about communication channels, and it is constructed from the viewpoint of an external observer. The decision in favour of the two-layer MCM structure was made to reduce verification complexity. Because the global choreography model is closer to user requirements, we can gain more confidence in their consistency through simulation and testing. Then we only need to verify that the local partner models are consistent with the global choreography models (more on this later) in order to assure that the MCM is consistent with user requirements.

The resulting concept soon gained developers' support, helped by the familiarity of its graphical representation. We built a prototype MCM editor based on SAP's internal platform, NetWeaver Development Studio, which includes a graphical modelling framework. We also used the editor as the basis for implementing various

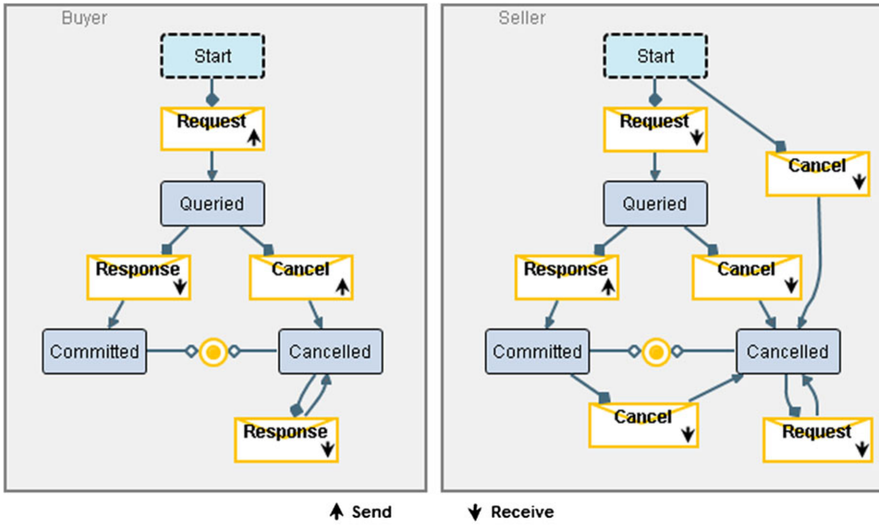


Fig. 6.2 Local partner models

formal verification and model-based testing techniques as plug-ins. Throughout the course of development, we continuously collected feedback from integration and testing architects in the field, thus involving potential users at each development stage. For instance, we extended the basic notion of state to allow for concurrency, which was an advanced feature needed in certain types of scenarios.

For evaluation purposes, we set up four pilots using real-life integration scenarios from the SAP platform, as described in [16]. The creation of each pilot model was conducted in *two guided sessions that lasted about one hour each*. In addition to that, we had *another two hours'* session of refinement and consolidation of the results. After the second session, semi-structured interviews were conducted with the pilot users. The general response was very positive. The participants perceived the possibility of formally describing the design as most beneficial, as it significantly improved communication between distributed development teams of interactive services. Furthermore, full integration of the existing modelling content (e.g., interface and component specifications) into the MCM was appreciated. The graphical modelling approach using a state-based representation was generally perceived as intuitive. The above case study showed that by using the MCM it is possible to model randomly chosen service communications that are part of a real SOA-based product using the MCM. The results suggest that the MCM is expressive enough to capture the relevant service communication.

As mentioned before, MCMs are first translated into Event-B for further formal analysis and test generation. The details of the translation can be found in [13]. Thanks to the state-based nature of MCMs, they are translated into Event-B in a quite straightforward fashion by using state variables and expressing transitions as state variable assignments in event actions. A global choreography model and its local partner models are translated into two separate Event-B machines. For any

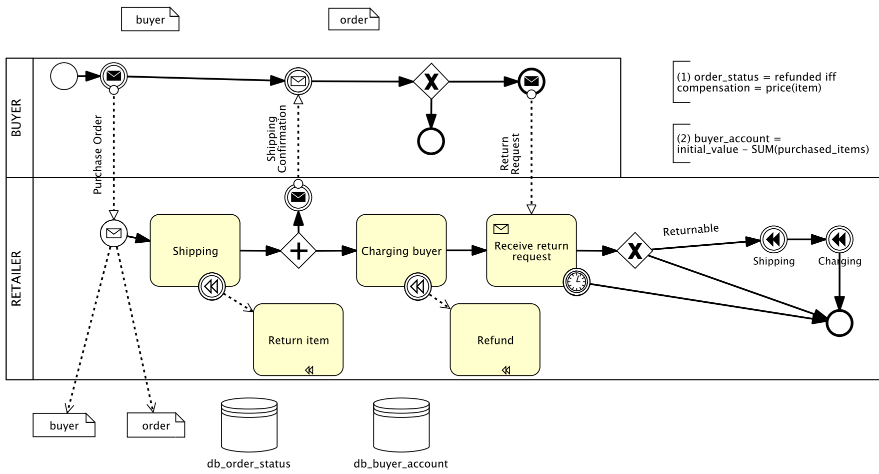


Fig. 6.3 A BPMN model of an online retailer

interaction between two partners in the machine representing the global choreography model, either the sending event or the receiving event of the respective message in the machine representing the local partner models is defined as the refining event of the interaction. These two possible refinement definitions are called *send-view* and *receive-view*. Message channels are modelled either as sets if they are EO channels (i.e. guaranteeing that the receiver gets each message Exactly Once), or as queues if they are EOIO channels (i.e. guaranteeing that the receiver gets each message Exactly Once In Order) The resulting Event-B model preserves the structure of the original MCM, so any verification results or generated test cases can be easily mapped back to their MCM representations.

6.2.2 Business Processes

Currently, business applications are usually built by integrating a broad range of highly configurable software components and services, which can be rapidly tailored to satisfy different and constantly changing business needs. Business process models are used to describe such integration scenarios and their workflows, facilitating an intuitive common understanding of the business logic between customers and developers. In addition to their use as documentation, business process models can also be simulated, analysed and verified to reveal design errors at an early stage of software development. BPMN has become the de facto standard business process modelling language, which is widely adopted in industry. A typical BPMN diagram is shown in Fig. 6.3: two collaboration partners (*BUYER* and *RETAILER*) and the flow of activities, events and messages.

BPMN is specified using natural and graphic languages, and comes without a rigorous semantics definition. Therefore, there are a lot of ambiguities in it that had

to be clarified as we designed the translation into Event-B [1]. Of course, these clarifications were made to meet SAP's specific needs. The translation works for most of the commonly used BPMN features, including comprehensive modelling of control flows, data modelling, compensation, message-based communication, error and exception handling, sub-processes, looping and multi-instance activities. The BPMN features not covered in the translation are most notably choreography and conversations as well as some types of flow objects, including call activities, transactions, conditional events and complex gateways. Some of these are rarely used in practice and would add significant complexity to the model. Others, such as transactions, are very vaguely described in the official BPMN specification and difficult to interpret.

Our translation was guided by three principles. First, the Event-B translation should be **structurally faithful** to the original BPMN model so that anyone with good knowledge of the original model can easily understand the translation. Also, any analysis result that we may obtain from the Event-B translation should be easy to map back to the original model. Second, the translation should be designed to improve **provability**, i.e., it should result in the automatic discharge of as many proof obligations as possible. Finally, we are interested in verifying properties for systems that allow **multiple instances** of the same processes.

We have tried two approaches to breaking down the complexity of the Event-B models that BPMN diagrams are translated into. In the first approach [1], we exclusively used the refinement relationship between machines to gradually add more and more information from a BPMN diagram. We start with a simple Event-B machine that contains only the control flow information of any collaboration partner, for instance, the *BUYER*. Then, we add a second machine that refines the first machine, and contains not only the control flow but also the data flow information of the *BUYER*. Subsequently, we gradually add the control flow and data flow information of the other partner in the machines. In the end, we use a final refining machine to add communication details. This approach has the following advantage: if we only want to verify a property related to the control flow of *BUYER*, then we can verify it in the first machine, which is much easier than verifying it in one that contains a lot of irrelevant information. However, since new information is always added to a refining machine without losing any old information, we still get an "all-in-one" machine, which becomes difficult to apply formal methods to.

Since our goal was separation of concerns, we experimented with another approach, taking advantage of the three model decomposition tools available in Event-B/Rodin [4]. Thus, we used modularisation, one of the decomposition tools mimicking the way that an object-oriented language uses interfaces/encapsulation/method-calls. The idea is to completely separate the local information of each collaboration partner. Each partner has an interface in which several publicly callable methods are exposed. The partner's details are, however, hidden in a series of refining machines invisible to other partners. In the end, we add a global machine to coordinate the interaction between partners. Such decomposition offers a clean separation of concerns. The detailed specification of each partner is replaceable and has no direct impact on the rest of the model, provided that its interface remains unchanged. Because of encapsulation, local behaviour can be completely verified within the

boundaries of the corresponding local partner, without information overload from other parts of the model. Verification of global properties can use local properties as intermediate lemmas. So, a proof procedure can be structured, and the degree of proof reuse is considerably higher.

6.3 Formal Verification

There are two kinds of properties that we focus on. The first kind is consistencies across different modelling layers, in particular, consistency between a global choreography model and its local partner models [5, 9, 15], as well as consistency between the MCM and the implementation models for business objects [7, 14]. The second kind is a selected set of invariants that a model must preserve during runtime, for example absence of deadlocks, absence of inconsumable messages in the MCM [6] and data consistency in business processes [1–3].

We applied both theorem-proving and model-checking approaches to the verification of the above properties. All domain-specific models were first translated into Event-B, and automated provers and the ProB plug-in of the Rodin platform were used to conduct verification. We could not achieve full automation of theorem proving even after several enhancements through various static analysis techniques and proof strategy optimisations. In contrast, model checking did not require much human intervention. However, we often ran into the state explosion problem, because checked models were usually too large for ProB to fully explore. In such cases, we had to manually reduce the explored state space by, for example, setting bounds on model variables. Nevertheless, we could still manage to obtain meaningful results by combining theorem proving and model checking. For instance, we usually applied model checking first with the hope of finding potential errors. We fixed the model accordingly and repeated model checking until no more bugs could be found. After that we started the more difficult and time-consuming proving procedure. Such a strategy can save a lot of time and is quite efficient for finding model errors.

In the following, we elaborate on how we apply theorem proving and improve automation with static analysis techniques.

6.3.1 MCM Verification

As depicted in Fig. 6.4, one major goal of our work was to enforce consistency between requirements and implementation. By introducing the MCM, we were able to divide this complex problem into manageable pieces:

- **Consistency Between Requirements and Choreography Models.** Requirements are not formalised in practice and hence applying formal methods at this level is impossible. Therefore, enforcing consistency between choreography models and requirements is a manual task. We found that the use of model simulation

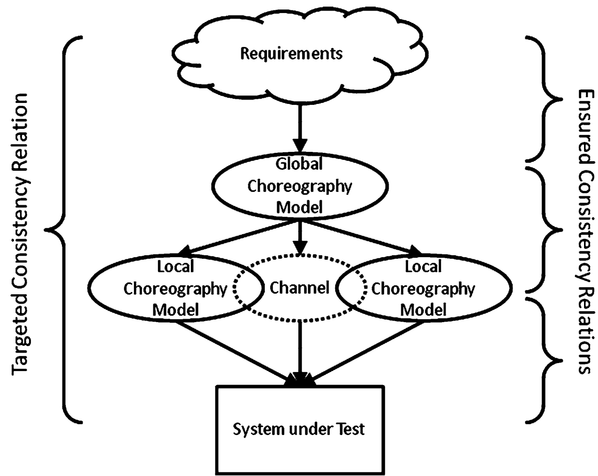


Fig. 6.4 Consistency relations in Choreography Modelling

based on ProB was usually sufficient to achieve high confidence that the choreography model captured what was informally described in the requirements.

- **Consistency Between Global and Local Viewpoints.** There are two possible solutions to enforcing consistency between global and local viewpoints: a generative approach, where the local views are generated from the global ones, and a checking approach, where global and local models are created separately and then verified for consistency with each other. In our work we implemented a mixed approach, which starts by generating local views from global ones but permits user modifications. The necessary consistency checks of manipulated views are realised by automatic transformation and verification, based on Event-B and Rodin.
- **Consistency Between Choreography Models and Implementation.** Service components are usually described with the help of implementation models, by specifying contained attributes (and their types) and state transition diagrams, and describing the effects of actions (such as service calls) on the internal states of components. We aimed to ensure consistency from choreography models to the implementation.

Figure 6.5 shows how we integrated Rodin into the MCM prototype for verification. The integration was made easier by the fact that both Rodin and our editor are Eclipse-based. A developer draws an MCM model using the editor. Within the editor, the translation of the MCM model into an Event-B model can be triggered. The resulting Event-B project is automatically loaded, and the consistency between the global choreography model and the local partner models is also formalised as machine refinements (through gluing invariants). The automated provers of Rodin then try to discharge all outstanding proof obligations (POs). A user can also interact with the provers to manually discharge POs. The ProB model checker can also be used to validate refinement relations.

We noted that in realistic scenarios MCM choreographies were not overly complex. Even the complicated real examples do not exceed 10–15 protocol states and

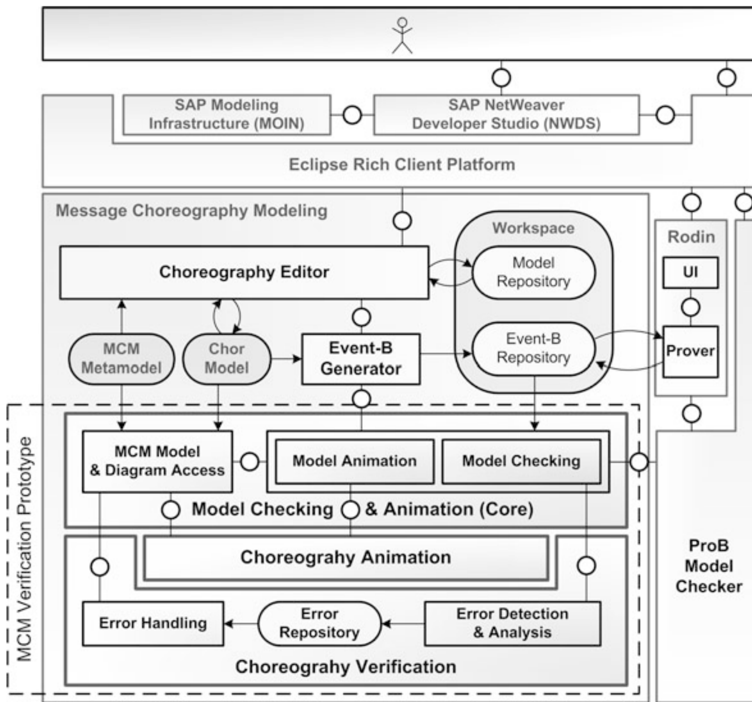


Fig. 6.5 Tool architecture for MCM and verification analysis based on Event-B

about 20 message events. We believe that this is a consequence of a good system architecture design that splits a complex system into manageable parts to be treated separately. However, even at this size of choreographies, subtle communication situations such as message racing and the large state space due to the datatypes of the exchanged messages occurring in a loosely coupled environment fully justify the use of automatic verification and validation techniques based on the MCM and Event-B.

Our experience showed that the automated provers were not able to automatically discharge a large set of proof obligations generated for gluing invariants. To improve automation, we enhanced our tool to automatically discover several kinds of invariants that describe certain dependencies between global states (those in the global choreography models) and local states (those in the local partner models) as well as dependencies between communication channel contents and MCM states [5]. The automated provers were then able to use these invariants as intermediate lemmas to discharge POs. Several experiments with realistic models show that about 300 to 600 POs were generated to verify consistency after invariant generation, and it was possible to automatically discharge up to 70 % of them. It was possible to discharge about 80 % of the remaining POs simply by manually switching to a specific prover (an interesting phenomenon related to how Rodin implements time-outs for

provers). It was possible to prove the rest of the POs manually without too much effort.

6.3.2 Several Remarks on Proof Automation

As illustrated in many cases, automatic discovery and generation of invariants is an important technique for reducing the difficulty of proving a property and increasing the number of automated proofs, a. However, which invariants should be generated depends not only on the types of models being verified, but also largely on specific characteristics of the individual models. For instance, for a business process model, it is important to have invariants specifying control flow dependencies and message flow dependencies [1]. For a business process involving data persistence, it is always helpful to discover additional dependencies between data flow and control flow. It is therefore hard to devise an automated algorithm to discover invariants for arbitrary models.

Another improvement to proof automation was achieved by making use of the Relevance Filter plug-in for the Rodin platform [8]. Using heuristics, the plug-in tries to pick most relevant and useful proof hypotheses from what is usually a very large pool of these. The use of this tool allowed a promising increase in the number of automatically discharged POs.

Inside SAP, we also developed a technique to better present the feedback from automated provers to designers, in case proofs fail to be derived [10]. The basic idea is to visualise the set of those states in the model that are associated with a certain proof step. The visualisation can be helpful in indicating and revealing potential errors in the design. We further enhanced this technique to allow a user to interact with the visualised state. For instance, the current state may ask the user to choose between two nondeterministic branches. By selecting one, the user is actually helping make a proof decision about which part of a disjunction should be focused on in the following proof.

We also discovered that the translation from a modelling language to Event-B can affect the level of proof automation. In proving consistency between the MCM and an implementation model [7], at the beginning we used logic formulas with quantifiers to define semantics for the implementation model. This proved to be ineffective since automated provers have always had great difficulties with quantifiers. Therefore, we decided to replace quantifiers with set operations, which automated provers can deal with better, using powerful simplification tools. After that we saw a large increase in the number of automated proofs.

No matter how much effort we have put into increasing the degree of proof automation, in almost every case there are some POs that require manual proofs. Even though the percentage of undischarged POs is quite low, their actual number is not small, and they are usually difficult to prove. This requires great knowledge and skills in theorem proving, which is not something we can expect from average developers and designers. Formal method experts could be hired to solve the manual

proof problem. However, it would still be a challenge to blend formal verification seamlessly and frictionlessly into software development in such a way that it supports other development activities without restricting or slowing down the whole process.

6.4 Model-Based Testing

It is hard to achieve full confidence in the correctness of software. Therefore, we focus on finding bugs with the help of software models in the last phase of deployment. Software models are very useful in guiding test designs, because they capture the essence of how software is supposed to behave. For instance, MCMs were used to automatically derive conceptual test cases, which can be easily mapped to actual test cases that run on the system under test [9]. According to the pilot users, the automatically generated test suites covered all tests that had been previously created manually. Another advantage of model-based testing is its complete automation. Test designers only need to create an initial test model, which is most of the time very intuitive, because we designed test models to be similar to the domain-specific ones that test designers are familiar with. Model-Based Testing (MBT) proved to be non-intrusive and very productive, replacing what is usually very tedious manual tasks of designing and creating test cases.

In this section, we illustrate how we apply model-based scenario testing to the existing business processes.

6.4.1 Scenario Testing of Business Process

As scenario testing is usually conducted on the user interface, we started working on MBT for graphical user interface (GUI) testing [11, 12, 14], which was a less studied research subject. Figure 6.6 depicts the envisioned testing approach that defined our deployment plan. Scenario testing is carried out when the whole system (or at least a major part of it) is developed and test-ready. The following describes the particular steps of this approach.

1. Based on SAP's expertise in industry's best practice, consultants, key customers and development architects derive business process models for a new product or feature or customer implementation so as to meet the market's or customers' requirements.
2. The created content, which effectively describes the usage scenarios of the new functionality, is used to generate test model skeletons. This step should be made automatic by using model transformation techniques.
3. The test models are then enhanced by test engineers in such a way that they reflect previously defined test goals and pin down the specifics of the concrete software architecture.

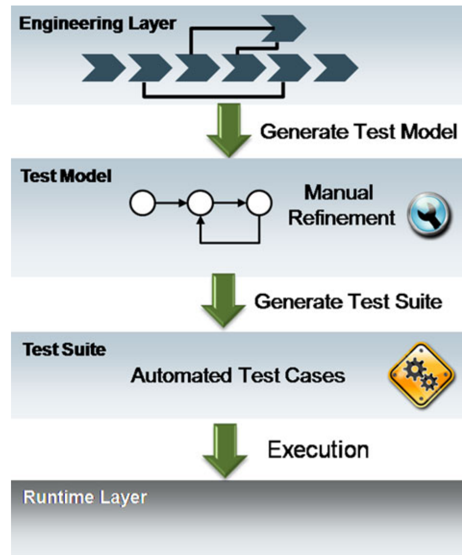


Fig. 6.6 Envisioned testing process

4. Abstract test suites are automatically derived from the test models, using MBT techniques.
5. The abstract test suites are optimised following best industrial practices (e.g., by minimising test case lengths while preserving test coverage). After further concretisations, the optimised suite is automatically executed on the user interface of the system under test.

In order to realise the envisioned deployment, we integrated various components into a testing framework productively used at SAP. Figure 6.7 presents the main blocks of this framework, including our components. The *Test Environment* offers UI-based keyword-driven testing capabilities through a *Scenario Editor*, which allows us to assemble captured test scripts and to visualise the generated executable scenarios (obtained from test cases). The scripts can be recorded through the *Script Recorder* component, which is connected to the System Under Test (SUT) for this purpose. Besides capturing user interactions on the SUT, the *Script Recorder* offers replay functionality, which is utilised for the stepwise execution of scenarios. Together with the *SUT* and the *Back-end Repository*, it assembles the original setup.

We extended the test environment by creating and integrating the *Test Model Editor*, which allows process-based test models to be created and edited. It further enables the triggering of the test generation and the visualisation of the resulting test suite.

In order to mitigate the risk of dependency on one single test generation technology, our goal was to integrate multiple tools and vendors, which we achieved by providing transformations from process-based test models (TMs) to abstract state transition machines (STMs); these can be further transformed into vendor-specific input formats in a straightforward manner. Since there is no standardised intermediate format, we created a proprietary STM. However, we published its concepts [12]

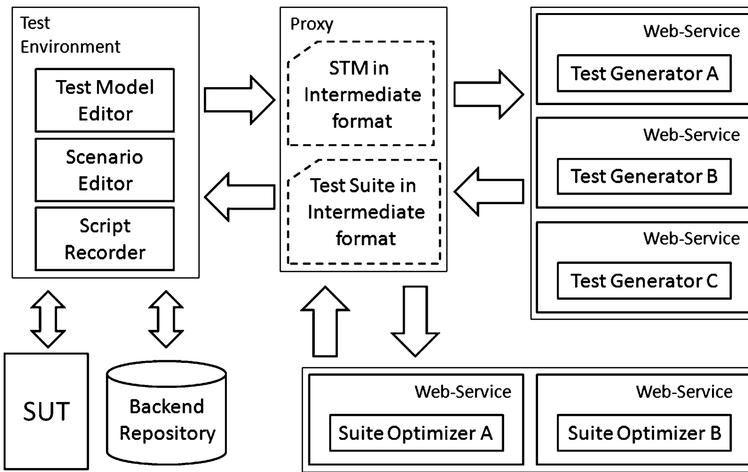


Fig. 6.7 Architecture of the MBT environment

and are active in contributing them to various emerging standardisation initiatives. A proxy has been set up for routing test generation requests in order to obtain a single communication partner, which allows us to add and update generator components without additional configuration of the test environment.

General-purpose MBT tools rely on various strategies to reduce the large initial test suites they produce during test generation. Therefore we decided to offer a unified test suite optimisation independent of the chosen test generator. This further allows us to consider custom requirements for the enterprise software domain. The different optimisation procedures are wrapped in another set of web services and can be used in the following way. After a test is successfully generated, the resulting traces are transformed into an intermediate test suite format and sent to the proxy component, which forwards them to an appropriate *Suite Optimizer*.

Test reduction is implemented on the intermediate format for test suites. Therefore, a further transformation of the results in the *Suite Optimizer* is not necessary. The proxy takes the reduced test suite and routes it back to the *Test Model Editor*, where it will be used to create a concrete test suite, containing executable scenarios.

Besides enabling the seamless integration of the *Test Model Editor*, detaching the test environment from the test generation services has the following advantages:

- **Reuse**: By using a generic input and output format, we are able to hide the complexity of specific model transformations in the input format of concrete test generators, thus making MBT accessible as a service to other potential test environments.
- **Performance**: Decoupling expensive computational functionality like test generation and test suite reduction promises better system performance and does not block front-end users. Replication of the Web services and introduction of load balancing to the proxy further increases scalability.

- **Maintenance**: The service-based decoupling in combination with a proxy also allows us to maintain and upgrade test generation components in a non-intrusive way.

6.4.2 Results

After prototyping, creation and integration of the components described in the previous section, we turned to one of SAP's major product areas in order to negotiate an evaluation strategy. It was agreed to set up a case study with seven development teams, which were asked to apply MBT in their scenario testing activities for a specific internal release. During these activities, the team members were asked to collect requirements and report bugs. At the end of the case study, further interview sessions were carried out with the participants, in order to get their overall assessment of the tool as well as information about their productivity and perceived learning effort. These results were consolidated and presented to the executive board of the product area, which consequently decided to start an unrestricted roll-out to internal development teams. As information about product and development activities needs to be handled with discretion (especially quality-related information), we will report the findings of the case study in a more general way.

Case Study Participants The participants did not have a background in formal methods but knew the basic concepts of business process modelling and were familiar with the concrete business process that they wanted to cover with different scenario tests. They were trained and experienced in the use of the proprietary testing environment, but did not have any knowledge of MBT. At the beginning of the case study, they received a two-hour tutorial on additional modelling and testing concepts necessary for understanding and operating our tool extension, as well as additional documentation and guiding samples. Furthermore, all participants could rely on remote expert support for any tooling, technology or process-related questions. On average, these support activities amounted to about one additional hour per participant.

Requirements Analysis Over the course of the case study, the participants collected 47 different requirements and ranked them based on their importance from 1 (an absolute showstopper) to 5 (nice to have). In the two-month period of evaluation we were able to incorporate all requirements ranked 1 and 2, and most of those ranked 3. The remaining requirements mainly concerned the automation of additional steps in the test generation process, which do not directly relate to test generation and had been manual in the original process, too (e.g., the linking of created test cases with test plans), or even addressed issues in the original test environment. Overall, only one requirement of the remaining concerned the enhancement of the test generation functionality, while the others mainly dealt with usability issues.

Interview Sessions Each participant was interviewed after the case study. All stated that the maturity of the tool improved dramatically during the evaluation phase, and agreed with the conclusion that both the tool and the new testing process were mature enough for wide use within the organisation. Furthermore, it was confirmed that the learning effort was small and the approach quite intuitive. The usability of the test model editor still left room for improvement, but was comparable to other internally used tools. It was noted by many participants that the MBT approach demanded greater care in the script recoding and test data definition activities. However, this was generally perceived as a positive side-effect.

Based on the requirements analysis and interview sessions, the executive board gained enough confidence to decide on a phased roll-out of model-based scenario testing to the whole product area. This roll-out will be accompanied by the handover of responsibility from our research unit to an operations team for the maintenance and further improvement of the tooling that was created in the context of DEPLOY.

6.5 Conclusion

Formal modelling and verification bring many quality assurance advantages to the development of business software. Design documents are complemented with software models that are accurate, executable, analysable, and can be used in deriving test cases directly linked to requirements. Formal validation and verification are not only used to prove correctness, but also to effectively find bugs, which is a nice alternative to traditional testing.

However, given the assumption that formal methods are hidden behind the existing domain-specific modelling abstractions, their success relies on the degree of automation. This is attested by the fact that model-based testing was more widely accepted by developers than formal verification, because MBT is fully automated. Users are not bothered by the technical details of test generation. They construct a test model that looks like a business process, and with a push of the button, test cases are generated and can be immediately run without any further effort. With formal verification, we are still in the process of increasing the degree of automation, to make tool usage and feedback more user-friendly, and to improve tool efficiency when dealing with large software models. Nevertheless, our pilot deployment of MCM verification is very promising and welcomed by software architects and designers. We will continue to work toward a seamless experience of using formal methods in business software development processes.

References

1. Bryans, J., Wei, W.: Formal analysis of BPMN models using Event-B. In: Kowalewski, S., Roveri, M. (eds.) FMICS. Lecture Notes in Computer Science, vol. 6371, pp. 33–49. Springer, Berlin (2010)

2. Bryans, J., Fitzgerald, J., Romanovsky, A., Roth, A.: Formal modelling and analysis of business information applications with fault tolerant middleware. In: ICECCS, pp. 68–77. IEEE Comput. Soc., Los Alamitos (2009)
3. Bryans, J., Fitzgerald, J., Romanovsky, A., Roth, A.: Patterns for modelling time and consistency in business information systems. In: Calinescu, R., Paige, R.F., Kwiatkowska, M.Z. (eds.) ICECCS, pp. 105–114. IEEE Comput. Soc., Los Alamitos (2010)
4. Hoang, T.-S., Iliasov, A., Silva, R., Wei, W.: A survey on Event-B decomposition. In: Automated Verification of Critical Systems AVOCS-2011. Electronic Communications of the EASST, vol. 46 (2012)
5. Kozyura, V., Roth, A.: Generation of gluing invariants for checking local enforceability of message choreographies. In: Jastram, M., Laibinis, L., Lösch, F., Mazzara, M. (eds.) Proceedings of DEPLOY Technical Workshop 2009. Newcastle University, Technical Report (2009)
6. Kozyura, V., Roth, A., Wei, W.: Local enforceability and inconsumable messages in choreography models. In: Proceedings of 4th South-East European Workshop on Formal Methods (SEEFM). IEEE Comput. Soc., Los Alamitos (2009)
7. Kozyura, V., Roth, A., Wieczorek, S., Wei, W.: Checking consistency between message choreographies and their implementation models. Electronic Communications of the EASST, vol. 35 (2010)
8. Röder, J.: Relevance filters for Event-B. Master's thesis, ETH, Zürich (2010)
9. Roth, A., Wieczorek, S., Kozyura, V., Wei, W., Wieczorek, S.: DEPLOY Deliverable D4.1: Report on pilot deployment in business information sector. Technical report, FP7-DEPLOY project EU (2010). http://www.deploy-project.eu/
10. Schur, M.: User interaction in formal verification of service choreography models. Master's thesis, Hochschule Karlsruhe Technik und Wirtschaft (2009)
11. Wieczorek, S., Stefanescu, A.: Improving testing of enterprise systems by model-based testing on graphical user interfaces. In: Sterritt, R., Eames, B., Sprinkle, J. (eds.) ECBS, pp. 352–357. IEEE Comput. Soc., Los Alamitos (2010)
12. Wieczorek, S., Kozyura, V., Schur, M., Roth, A.: Practical model-based testing of user scenarios. In: ICIT12, pp. 306–311. IEEE Comput. Soc., Los Alamitos (2012)
13. Wieczorek, S., Kozyura, V., Roth, A., Leuschel, M., Bendisposto, J., Plagge, D., Schieferdecker, I.: Applying model checking to generate model-based integration tests from choreography models. In: Núñez, M., Baker, P., Merayo, M.G. (eds.) TestCom/FATES. Lecture Notes in Computer Science, vol. 5826, pp. 179–194. Springer, Berlin (2009)
14. Wieczorek, S., Kozyura, V., Wei, W., Roth, A.: DEPLOY Deliverable D4.2: Report on enhanced deployment in business information sector. Technical report, FP7-DEPLOY project EU (2011). http://www.deploy-project.eu/
15. Wieczorek, S., Roth, A., Stefanescu, A., Kozyura, V., Charfi, A., Kraft, F.M., Schieferdecker, I.: Viewpoints for modeling choreographies in service-oriented architectures. In: WICSA/ECSA, pp. 11–20. IEEE Comput. Soc., Los Alamitos (2009)
16. Wieczorek, S., Stefanescu, A., Roth, A.: Model-driven service integration testing—a case study. In: QUATIC'10, pp. 292–297. IEEE Comput. Soc., Los Alamitos (2010)

Chapter 7
Formal Methods as an Improvement Tool

Aryldo G. Russo Jr.

Abstract This chapter describes the work of AeS in the context of the DEPLOY Associate program. It outlines the progress of the pilot project, the development of a specification of a simple railway system called "dead man control". In addition to this, we also present some parallel developments, some of them theoretical and others practical, in the use of formal methods in industry, focusing on such important points as requirements validation, productivity and dependability.

7.1 History

Grupo AeS is a Brazilian company established in 1991. At first it worked in the building automation field, but in 1998, when the first Brazilian General Door Control (GDC) System for rolling stock doors was developed, AeS became involved in the railway field. This equipment was designed to be the interface between train requests (operator requests, signalling requests and so on) and the door system itself. This system was safety-critical, as its major operation was sending an open command to the doors in each cabin. Since then, several different safety-critical and safety-related systems have been developed by AeS, such as "dead man control", parts of the brake system and the speed limit control.

Due to advances in technology, many safety functions that were handled by hardware are now the responsibility of embedded software. There are standards to be followed to increase the equipment safety level. One of the most widely used is IEC 61508 [2]. This standard presents four levels of safety (the higher the level, the higher its safety), the so-called Safety Integrity Levels (SILs); above level 2, the use of formal methods is required or recommended for achieving a certain degree of completeness, robustness and safety, which goes up with the level. The goal of using formal methods is to produce an unambiguous and consistent specification that is as complete, error-free and has as few contradictions as possible, and that is simple to verify.

A.G. Russo Jr. (✉)
Grupo AeS, R Domingos Barbieri 298, Sao Paulo, Brazil
e-mail: agrj@aes.com.br

A. Romanovsky, M. Thomas (eds.), *Industrial Deployment of System Engineering Methods*, DOI 10.1007/978-3-642-33170-1_7,
© Springer-Verlag Berlin Heidelberg 2013

In 2006, AeS began studying formal methods to decide which method to use, where and how to apply it and so on. Later AeS became part of the DEPLOY project as a DEPLOY Associate. This chapter tells the story of AeS's participation in this project, using case studies to present what was done during these years and what was learnt.

7.2 Context

First, a brief description of AeS's structure, projects, team and so on is needed to introduce the context of our applications.

- Our Research and Development team consists of ten engineers, none of them with a formal methods background, and one R&D manager (an engineer with some formal methods background).
- During the time of the DEPLOY project the team received a whole week of training covering the Rodin environment, refinement and UML-B.
- As AeS operates in the railway domain, several standards need to be followed, some of them related to safety, such as EN 50128 and IEC 61508. These standards recommend the use of formal methods for applications that require higher levels of safety (SIL3 and SIL4).

7.3 Case Studies

This section presents a chronological description of AeS's case studies, aiming to show what the problem was, what we applied to address it and what the results were. One important point is that in none of our case studies did we use formal methods (or formal methodologies) from the beginning to the end of the development process. We only applied them precisely when we thought they would be more effective, avoiding extra cost and effort. Nevertheless, we think that at some point in the future a completely formal development process could be used and evaluated for comparison with the traditional approach. This was to be our final case study, but unfortunately it has not been finished at the time of writing.

7.3.1 Early 2007: The B Method and Railway Domain—Breaking the Wall

In our search for formal methods, which began in 2006, we discovered that the B method had gained acceptance in the railway domain. At that time the reason for that was not important, and we decided to follow this flow. Of course, only theoretical publications could be found on the subject and no introductory guide was

available. Digging deeper, we discovered that in early 2007 a conference on B would be held in Besançon, France. Since from the industry's point of view, conferences are places to go to in order to learn how to use tools, techniques, methods and so on, it was thought to be the right place to start.

It was, indeed, but not in the traditional way. This conference was our first contact with the real world of academic development, with people committed to using the B/Event-B methods and with the Rodin tool. This conference led to our becoming a DEPLOY associate partner and successfully using formal methods in our applications.

7.3.2 Early 2008: Requirements Verification

In our first attempts to use formal methods, we tried to apply them from the beginning of the development process, by converting the natural language specification into a formal one. We could not find a tool that would help us do this manually, but studying the formalisation process and the notion of abstraction and refinement helped us change our way of thinking and improve our understanding of the specification.

As a result of this first attempt, I can report two small but important achievements:

- We found inconsistencies in the natural language specification. We started by formalising a small portion of a door system specification; at that point our objective was to formalise exactly what was written in that specification. In doing this we reached a point where the formal abstractions could not be proved, and while looking for a cause we found contradictory information imported from the natural language specification.
- We found that information was missing from the natural language specification. We tried to formalise the existing specification of our pilot project, a small system designed to terminate the train movement in case of problems with the human operator. Its specification was only one sheet long, and as in the previous case, we formalised exactly what we had, discovering in the process that a lot of the necessary information was missing. This prompted us to formulate several assumptions that had to be validated with the customer specialist on order to finish the specification.

In the end we realised that we could use formalisation in both attempts as a tool to verify and correct customers' specifications.

7.3.3 2009: Tool Comparison

In order to verify how the current tools can be modified to meet industrial needs, I conducted a brief comparison of some of the existing tools, restricting it to tools

Table 7.1 Comparison table

Aspect	Capability	Usability	Adaptability	*Results*
Atelier B	2	1	2	*5*
Rodin	2	2	1	*5*
SCADE	2	3	3	*8*

that I knew well and those that have been used in my application field, that is, in railway applications. These tools are Atelier B, Rodin and SCADE. This study was conducted three years ago, so results might be different if it was done now.

7.3.3.1 Methodology

The "oracle" I used to classify each tool as belonging to a particular category was my personal intuition, since a more detailed research had not been conducted yet. However, in the last three years I have been able to take into account comments from several people who I have trained.

It should be noted that the tools differ greatly in terms of their maturity. Whereas SCADE and Atelier B had been in the market for a long time, Rodin was about to be released in its first official version (1.0), which means that the former two tools had already generated plenty of feedback from industrial users, helping the developers change them when users were not satisfied (in the case of Atelier B, after a lot of complaints about the user interface, the developers completely changed the GUI), while Rodin had not had time yet to receive feedback or be adjusted.

The comparison methodology was based on the following three aspects:

- *capability*: how the tools can satisfy project constraints
- *usability*: the difficulty the user faces when trying to use the tool
- *adaptability to the current development process*: how well the tool fits into the process without causing too many changes to the way it has been conducted so far.

To evaluate these aspects I used the following simple ranking method:

- *1*—Very difficult
- *2*—Medium
- *3*—Easy

I present the results in Table 7.1.

7.3.3.2 Chart Comparison

- Atelier B
  - the capability to solve the project constraints is not bad, but you do need to know a lot of the formal language and its constructs to have easy proof obligations.

  - although version 4 of Atelier B has much better usability, all comments I have had so far are based on the previous version, where the lack of a good user interface made its use painful.
  - since it supports development from the specification to the code it can be considered as a good tool for that purpose, but as the interactions in the middle phases (refinements) are sometimes painful, it cannot receive a higher grade.

- Rodin
  - as it is not very different from Atelier B, a similar assessment is made. The capability to solve the project constraints is not bad, but you do need to know a lot of the formal language and its constructs to have easy proof obligations.
  - the way that Rodin was constructed is a great help to an inexperienced person, as you just need to fill in some fields to have a basic specification, but the lack of a text editor that could help more experienced users and speed up the specification process lowers its classification.
  - at the time of this evaluation, there was no possibility of decomposition, and the fact that help was available only in the system specification phase made it a difficult tool to use. These features have now been added.

- SCADE
  - because SCADE is based on a different concept, where formal methods are behind the scenes, it has the capability to deal with project constraints, but you still need some formal methods background to construct correct models.
  - as it was built from the very beginning to be an industrial tool, its usability is its strongest point, with a good interface and a lot of fancy features that appeal to the user. A lot of things can be done based on templates and patterns, which is of great help.
  - besides the capability to go from the specification to the code, it also has some other, complementary tools that help you with important auxiliary tasks in the project, such as requirement management, traceability and so on.

7.3.4 Late 2009: Writing a Formal Specification—The User's Point of View

B [2] is a formal method that allows us to produce proof obligations that demonstrate correctness of the design and the refinement. Nevertheless, there is no standard mechanism for mapping requirements to formal specifications. To overcome this issue, different solutions have been proposed by researchers. In [13], the authors have presented a traceability between KAOS requirements and B. Some authors are investigating the use of the Problem Frames approach [7] as a possible response. A mixed solution using natural language and UML-B has been proposed in [10]. However, these approaches use non-standard artefacts for requirement specifications, which

we believe discourages designers from adopting formal methods since they have to spend time learning them.

Use cases [9] can be considered as a de facto industry standard for requirement specifications. They provide a good way to capture how the end user interacts with the system by detailing scenario-driven threads. A typical use case describes a user-valued transaction in a sequence of steps expressed in natural language, which makes use cases readable to most end users. In [11] the author presents an approach to building B specifications from use case models. This approach is similar to ours, but his project focuses on bringing the object-oriented paradigm (including UML diagrams) to formal methods. His method also maps each use case as a unique B operation, which we believe is incorrect since, according to Jacobson [8], each use case can have many transactions.

Another relevant point about use cases is the possibility of deriving test cases. A recent and important development model is the so-called Test Driven Development [3], where the input information used to generate the source code is the test cases, instead of use case scenarios or other traditional requirement documentation.

We have tried developing B specifications from use and test cases. Use case transaction identification can be used as a guideline for defining B operations, including the pre- and post-conditions. In the same way, test cases can help with the definition of global invariants and constraints.

7.3.4.1 Mapping Use Cases to B

The aim of this approach is to support the earlier phases in the proposed development process. Typically, formal methods have so far been introduced only in the design phase, where the requirements are already well defined. In industrial projects, however, the requirements are not usually well defined.

The method proposed here would be helpful during early phases, both

- at the top, helping elicit the requirements (this might be combined with other techniques and methods, such as Jackson's Problem Frames)
- at the bottom, helping create the first abstract formal model, so that it can support the first definitions.

For mapping use cases to traditional B specifications we propose that use case scenario sentences should be written using a controlled natural language (CNL) described in terms of our use case transaction definition, which is based on Ochodek's transaction model [12].

Definition 1 A transaction is the shortest sequence of actions by an actor and the system, which starts with the actor's request and finishes with the system response. System validation and system executive actions must also occur within this sequence. The pattern for a transaction is written as a sequence of four steps in a

scenario:

n. An actor's request action (U).
$n+1$. A system data validation action (SV).
$n+2$. A system executive action (e.g., a system state change action) (SE).
$n+3$. A system response action (SR).

We have also decided to define the grammar using subject-verb-object (SVO) sentences because they represent the sequence of events clearly. We have mapped the use case actor as the subject, a set of action predicate synonyms (for example validate, verify and so on would be grouped together) as the verb and the rest of the sentence as the object.

7.3.4.2 Results

We approached mapping requirements to B (Event-B) models through use and test cases in a pragmatic way. At the time, we were not interested in the automatic translation of use cases for formal specifications since there are many natural language ambiguity problems. The intention was to take the use cases as a guideline to starting B specifications.

We also studied other approaches, such as Problem Frames and KAOS. The intention was to facilitate the beginning of the development process, where formal methods are supposed to be used, as much as possible.

This was only an attempt to find how well formal methods could fit into an UML development process, and a lot of further work will be required before it becomes a mature process.

7.3.5 Early 2010: A Methodological Approach to B Formalisation

Developing a formal specification of a system from informal functional requirements of a customer remains an open issue in the software engineering world. Good, that is well-written and easy to understand, requirements are essential [7], but not sufficient for achieving this goal.

We propose a framework for structuring and organising such requirements, resulting in a requirements specification document. This pseudo-model facilitates the development of a formal abstract model of the system functionality using the B method.

In this work, requirements elicitation is done without feedback from the customer, for two reasons. First, requirements elicitation often produces inconsistencies and redundancies, and one needs to produce a formal representation that presents the requirements to the customer in a completely coherent way; so, we consider the customer feedback once we have the model constructed and any inconsistencies

and lack of information are clearly pointed out. Second, this work is part of a larger project to develop a method for systematically organising the requirements of a system which has only natural language documents as input. As a side effect of this work, the formal model could help the customer rewrite the natural language document to resolve inconsistencies and redundancies in future projects with similar requirements.

To reduce the gap between the requirements and the formal model of the system and to clarify traceability, it is helpful to use an intermediate representation such as use cases, Problem Frames [7] or KAOS [5]. This work employs an adaptation of the WRSPM approach [6], a reference model that can be adapted without requiring a whole new language to represent the requirements (as opposed to KAOS or Problem Frames). It is the basis of the method explored in this work. WRSPM represents the customer requirements in five layers:

- **W** (World)—the domain knowledge (which includes the properties of the environment);
- **R** (Requirements)—the actual requirements of the system, constraining **W**;
- **S** (Specifications)—the connection between the world and the system, expressed as invariants and operations;
- **P** (Program)—the implementation (which may be generated from the formal model);
- **M** (Machine)—the execution environment for **P**.

In order to produce a model that we can verify and validate, and that will point out obvious inconsistencies, redundancies and lack of information, the B method is applied to the specification as defined in WRSPM. Once a formal model is constructed, one can use animation tools like ProB [14] to show it to the customer; it would be much easier to understand than mathematical proof obligations. This process can also be used to verify the advantages of the methodology and identify the main challenges in developing an abstract formal model using the horizontal refinements approach, which may suggest directions for future work.

7.3.5.1 The Methodology

The method for generating the WRSPM-based model that we propose consists in processing the customer requirements (expressed in natural language) one by one. This way we incrementally produce the system specification **S** in terms of *phenomena*, *invariants* and *operations*. We break requirements into **W** (the world assumptions) and **R** (the requirements), and establish traceability by relating the labels of the *atomic* requirements with the labels of the invariants and operations manually (in the future, with the development of tools, this should be done automatically).

In Fig. 7.1 we have a flowchart illustrating the process step-by-step. We take each requirement (already decomposed into an *atomic* requirement) and firstly identify all the *phenomena*. Then, for each *phenomenon* that has been identified for the first time we first define invariants regarding its type and next the invariants and operations

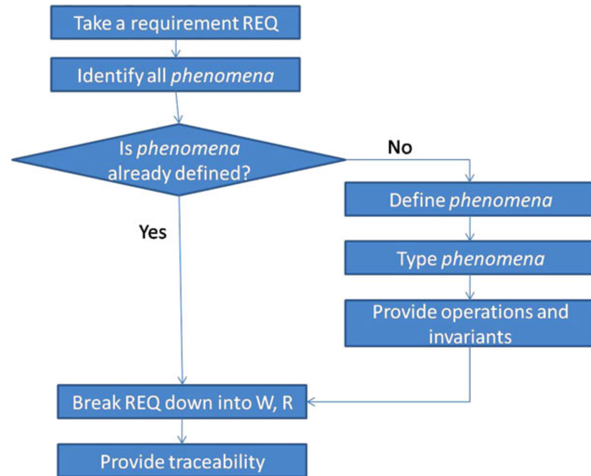


Fig. 7.1 Flowchart illustrating the processing of requirements

relevant to its state. Once all *phenomena* of that single requirement are identified and (the new ones) typed and the invariants/operations related, we break the text of the requirement into information about the environment (**W**) and constraints on the system (**R**). The last step is to provide the traceability, making the links between the labels of the *atomic* requirements and the elements of the WRSPM specification.

In order to model a complete system this way, we need to decide on a formal language that is rich enough to express the formal artefacts and phenomena.

Once all requirements are thus modelled, we have a complete abstract formal representation of the system (using, for example, the B method). This is the starting point of a series of refinements, the last one resulting in **P** (a system model in a programming language), to be executed in some execution environment **M**. In the process, dependency links should be maintained between all artefacts to guarantee traceability. Upon completion of these steps, we have a correct and complete implementation of the system (since the requirements are also correct and complete).

7.3.5.2 Horizontal Refinements

Some systems are too large and complex for their formal modelling (the abstract model) to be done in just one step, so it has to be done in successive steps. That is how the idea of *horizontal refinements* (or superposition) arises.

The idea is to start with an abstract model that only considers some of the requirements, and to introduce new requirements through successive refinements, new variables, stronger preconditions, and new invariants, actions or operations.

The person doing the modelling process has to explore the requirements document (in the methodology presented in this chapter, the WRSPM model) and gradually take from it the elements to be formalised, taking care to correctly define the architecture that supports the incremental addition of information and to define a good order in which to add the requirements, so that abstraction disparities do not

become a problem (as with a very *concrete* requirement being added at a highly abstract level). It is also necessary to guarantee completeness with the previous model (WRSPM): the horizontal refinements process must continue until all the requirements and definitions have been taken into account, and only when this is complete, can the designer start the process of vertical refinements, pushing the model into more concrete levels through design decisions until a level concrete enough to generate code is reached.

7.3.5.3 Lessons Learned

Once we had sufficient additions to the abstract model (to be further expanded with the horizontal refinements), it was analysed using the animations tool ProB [14], which provides such features as step-by-step model checking and execution to analyse changes in the system state by executing the operations defined in the specification. This allowed us to explore the B specification, checking for errors and identifying inconsistencies and lack of information: undefined states of the variables were reached since the customer's requirements did not say what to do in certain conditions. It would also have been possible to show the customer the result of the project so far, using ProB to demonstrate examples of the model execution.

The next stage would have been to horizontally refine the abstract model of the system, since we only considered some of the requirements in the process of construction of the model, so the new requirements would be introduced through refinements. It was at this point that we had big conceptual problems.

To model the architecture and the communication and complexity decomposition with the B Method, the clause **INCLUDES** is necessary, and this clause cannot be used to include a *refinement*, which means that, in a practical example, the *machine System* of the abstract model cannot include the *refinement CGP_r*, which cannot include the *refinement Opening_r*, and so on. In addition, one cannot add a new operation through refinements: it is necessary to go back to the abstract model and define the whole signature of the operation (with the variables to be received and returned), and then do the refinement step. To make it possible to horizontally refine the abstract model, one would have to analyse *all* the requirements to define at least the signature of *all* the operations that are going to be performed on *all* the machines. Therefore, in order to be able to apply horizontal refinements, we decided to use another approach: to build the model and do the refinements, using the Event-B [1] formal methodology and its extensions that enable the decomposition [4] process.

In Event-B we are allowed to add events (similarly to operations in B) through refinements, so we can indeed *expand* the model. The key point about the horizontal refinements in Event-B is, however, *decomposition*. There is no abstraction **INCLUDES**, and there is no communication between different *machines*, but one can introduce them through *decompositions*, based on the concepts of *shared events* (the same event is shared by independent *machines*, which models the architecture communication between different components) and *shared variables* (the same variable is shared by independent *machines*, which models complexity division inside the same component).

Using these features, we will be able to model the architecture of the communication between the CGP and the environment (through shared events), and the complex division of the CGP and its nested components (through shared variables). Once decomposition is done, we can independently refine the decomposed *machines*, getting a more expanded model and doing more decompositions if necessary.

7.3.6 Early 2011: A Changed Approach to Safety Verification

In this example, we present the architecture of a graphical tool prototype based on formal methods and its approach to validating track topologies and train movement conditions. The tool is Eclipse-based and compatible with the Rodin platform. Its main features are graphical simulation of railway specifications and validation of train movement properties. Called VeRaSiS, the tool uses Event-B to formalise, prove and verify the generated properties.

In the context of safety-critical applications, where failures in the system can potentially lead to life-threatening issues or huge financial losses, as in nuclear, medical, or railway domains, some properties need to be exhaustively verified in order to guarantee a minimum level of confidence in the system. Exhaustive verification aims at ensuring that the system will not allow dangerous situations or, if such situations occur, that it will activate the right protections to avoid their propagation. In the railway domain, one of the most important safety-critical systems applications is the signalling system which is, among other things, in charge of making a train move in a specific direction and secured condition. In order to guarantee these, the track topology and the movement conditions are usually manually translated into a set of properties that are verified and validated by relying on the validator's expertise. This process chain is not sufficiently reliable to guarantee that errors and/or inconsistencies, which are very difficult to pinpoint during the validation phase, will not creep into the writing process. The superior effectiveness of formal methods over manual validation in ensuring safety in critical systems has now been clearly demonstrated. Yet, despite the plethora of research on and tools developed to facilitate the use of formal methods, mainstream developers are still reluctant to use them. In order to help them overcome this reluctance, we developed VeRaSiS.

In the mid-1980s, Siemens Transportation Systems (STS), formerly called Matra Transport International, was the first company to lead a subway line automation project. Automation was a difficult concept because of the safety implications. Until then, the usual development process was not considered secure enough to cope with the paramount issue of the subway passengers' safety. The need to increase the confidence in the development process led to the use of formal methods. Line 14 of Paris Metro is one of the most exemplary outcomes of this project. Currently, the line carries more than 45,000 passengers per day, and since its implementation several automatic rail lines have been built. Nowadays, STS and some other companies have become experts at using formal methods in railway requirements. However, the mastering of these methods by mainstream software developers is still a challenge.

The use of formal methods in the railway domain depends mainly on the requirements type. The next section focuses on the train movement. More precisely, it takes a closer look at the issue of Boolean equation validation.

7.3.6.1 The Boolean Equation Validation Problem

Boolean logic is a calculus of truth values. It has many applications in electronics, computer hardware, and is the basis of all modern digital electronics. In the rail domain, Boolean equations are used to define safety properties, such as the constraints that specify the safe movement of a train from one location to another. A set of Boolean equations (called Boolean Equation Library, or BEL) has to be generated for all paths in the model designed.

In industrial projects, the equations are generated manually and validated either manually or with formal tools. Despite the time spent constructing and validating these equations, sometimes errors or inconsistencies can persist even after the manual validation, which can at worst lead to accidents or at best, if detected in time (that is, during the test phase), make it necessary for developers to rework the equations from scratch.

7.3.6.2 Using Formal Methods to Validate Boolean Equations

A preliminary analysis was done using a confidential real case provided by a Brazilian transportation company in order, firstly, to test the efficient use of formal methods, and secondly, to convince companies to change from manual to formal validation. This study aimed to revalidate a specification which had already been validated manually. First, the example was translated into a classical B machine. A parser that converts Boolean equations to B [5] notation was developed in order to automate the translation. The machine was run in ProB and gave a counterexample showing a small error that was not detected in the first validation: two speed limits could be selected, nondeterministically, at the same time. This case study was chosen at random, which means that probably there are other examples validated manually that contain errors. This study has, firstly, demonstrated the need for replacing manual validation with formal validation, and secondly, revealed the necessity of developing a formal tool to automate the validation of Boolean equations from a graphical model.

7.3.6.3 The VeRaSiS Plug-in

The VeRaSiS plug-in is a tool for validating the movement of trains in a railway system. It is Eclipse-based and is built for extensibility. It provides three main features: firstly, graphical simulation of use cases, secondly, fully automatic Boolean equation generation, and finally, validation of Boolean equations. The VeRaSiS plug-in

comes with a graphical interface allowing users to design railway systems. The BEL is automatically generated. After this, a formal tool is used to validate it. Error messages are translated into natural language in order to shield users from the formal language. Even so, for it to be used in an industrial environment, a minimum understanding of the formalism is needed.

7.3.6.4 Results

We have presented a new formal tool to model railway specifications and validate train movement properties. This Eclipse plug-in is designed for industrial needs and developed to address the requirement that mainstream users should be able to develop formal railway models while staying away as far as possible from difficult formal specifications. The VeRaSiS plug-in replaces the traditional Boolean equation validation process, which is not strong enough to guarantee the safety level required, as stated before. This project has proved the VeRaSiS concept in a case study based on an industrial specification. The study suggests two important conclusions. The first one is that we have to optimise the algorithm for generating the BEL from the graphical model and the translation template from BEL to Event-B. The second one is that further work is needed: normally, the translation is one-to-one, as one BEL becomes one event; however, as complexity grows, introducing new kinds of BEL, this rule might not be valid any more. We also need to generate a set of rules for each kind of BEL that needs to be created in a real project. While in the case study we selected only one or two kinds of BEL so as to be able to validate the concept, in an operational model all the required equations have to be generated. More case studies would certainly find other limitations of the VeRaSiS concepts. Our primary goal is to have a full first version of the VeRaSiS plug-in in order to prove its functionality.

7.4 Conclusions

Some comment should be made about the introduction of formal methods in the development process, particularly in small companies (as in big ones, you could just hire some specialists and put them to work together with the development team). So far, even with some training, we have not found a good way to completely change our development process. We do have some trained people who are able to understand and use Event-B and Rodin, with some restrictions. But it takes a lot of time to get rid of the traditional approach and establish a new one (especially when all products are already under development).

To overcome this problem, we have introduced formal methods gradually, in points where we believed they would help improve product quality and dependability. In a perfect world, parallel development (using formal and traditional methods) would be ideal to convince developers of the power of formal methods.

Using formal methods informally does really help change the way of thinking, meaning that even if the process is not entirely formalised, the minds involved in the development follow the refinement process.

So far, we have used formal methods in different ways:

1. We have verified gaps and mistakes in the natural language requirements. We have modelled the specified system (only the abstract model) and used it to reveal inconsistencies in the natural language specification. Requirement traceability is a real problem in big industrial projects as the main task of the management team is to show that what has been developed satisfies the requirements. We have not found a way to do so at this point (although we are closely following the ProR project, which may solve this problem).
2. We have specified the door system using different approaches (B, Event-B and UML-B). It is a distributed system and different parts of the "standard" system might be used in different customer projects: it is a kind of product line. We have not found any way to reuse parts of the specification, only a way to reuse the whole specification making the necessary modifications. Even with the Rodin decomposition tool, it is not clear that we will be able to reuse parts of the specification.
3. We think that the use of formal methods increases dependability of the system, though we lack strong evidence for it. As we cannot yet generate code directly from the formal design, it is possible that errors that were avoided in the specification and design steps might nevertheless be introduced in the manual coding. Formalising the system helps a lot in getting a deeper understanding of it; as a result we believe that some errors are avoided.

One point that is not yet possible to support by measurement, but I attest to (based on my professional intuition), is that the adoption of formal methods and, more precisely, the formal thinking has helped us a lot in structuring the development process even if the application of formal languages has not been fully achieved. It has helped us improve the quality of the final product and drastically decrease the time dedicated to final tests.

References

1. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
2. Abrial, J.-R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, New York (1996)
3. Beck, K.: Test Driven Development: By Example. Addison-Wesley Longman Publishing, Boston (2002)
4. Butler, M., Hallerstede, S.: The Rodin formal modelling tool. http://deploy-eprints.ecs.soton.ac.uk/4/1/eventb.pdf
5. Darimont, R., van Lamsweerde, A.: Formal refinement patterns for goal-driven requirements elaboration. In: Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering (1996)

6. Gunter, C., Gunter, E., Jackson, M., Zave, P.: A reference model for requirements and specifications. IEEE Softw. **41** (2000)
7. Jackson, M.: Problem Frames: Analyzing and Structuring Software Development Problems. Addison-Wesley Longman Publishing, Boston (2001)
8. Jacobson, I.: Object-oriented development in an industrial environment. In: OOPSLA '87: Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications, pp. 183–191. ACM, New York (1987)
9. Jacobson, I.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, Reading (1992)
10. Jastram, M., Leuschel, M., Bendisposto, J.: Mapping requirements to B models. DEPLOY Deliverable. Unpublished manuscript (2009)
11. Ledang, H.: Automatic translation from UML specifications to B. In: Proceedings of the 16th Annual International Conference on Automated Software Engineering Conference—ASE 2001. IEEE Comput. Soc., Los Alamitos (2001)
12. Ochodek, M., Nawrocki, J.: Automatic transactions identification in use cases. In: The 2nd IFIP TC-2 Central and East European Conference on Software Engineering Techniques, CEE-SET 2007, Poznan, Poland, October 10–12, 2007, Revised Selected, pp. 55–68. Springer, Berlin (2008)
13. Ponsard, C., Dieul, E.: From requirements models to formal specifications in B. In: ReMo2V CEUR Workshop Proceedings, vol. 241 (2006)
14. ProB: Animator and model checker. http://www.stups.uni-duesseldorf.de/ProB (February 2012)

Chapter 8
Critical Software Technologies' Experience with Formal Methods

Alex Hill, Jose Reis, and Paulo Carvalho

Abstract Critical Software Technologies (CSWT) participated in the DEPLOY project as an Associate; its chosen case study was the Reversionary Flight Display (RFD), used on board commercial and military aircraft. The case study was used to explore and learn about both the Event-B method and the Rodin toolset. This chapter relates the experience of CSWT with Event-B in the context of DEPLOY, describing the advantages of this formal method for safety- and mission-critical systems and the lessons learned. The conclusion reached is that Event-B has great potential in the safety- and mission-critical systems domains. The performance of the verification and validation process defined in the DO-178 and ECSS standards may also be improved if the formal method is adopted to verify the consistency and completeness of requirements. The formalisation of system properties early in the lifecycle and the formal analysis of these properties through controlled experiments are seen as the right route to reducing costs.

8.1 Domain Description

Critical Software Technologies (CSWT) has always been involved in the domain of safety- and mission-critical systems development. The company's role has not been confined only to the actual design of safety-critical systems but has included the verification and validation of high integrity systems developed by external suppliers.

The domain of safety- and mission-critical systems development relevant to CSWT can be characterised as having in general very well-defined processes that

- identify the planning, development, verification and validation, configuration management and quality assurance activities required to produce a system
- identify the evidence required to demonstrate that specific activities have been conducted. For instance, DO-178B defines the level of coverage required in Verification and Validation Activities based on the criticality of the software
- define what activities are required for each criticality level.

A. Hill · J. Reis (✉) · P. Carvalho
Critical Software Technologies Ltd., 2 Venture Road, Chilworth, Southampton, UK
e-mail: jreis@critical-software.co.uk

A. Romanovsky, M. Thomas (eds.), *Industrial Deployment of System Engineering Methods*, DOI 10.1007/978-3-642-33170-1_8,
© Springer-Verlag Berlin Heidelberg 2013

Table 8.1 Delta cost and schedule of DO-178 per criticality level [2]

Level E	Level D	Level C	Level B	Level A
Baseline	Level E + 5 %	Level D + 20 %	Level C + 15 %	Level B + 5 %

Examples of standards which are used in aerospace and defence are DO-178B, ECSS-E40 and DEF-STAN 56. Each standard has specific clauses that in general have impact on the quality of the output, the cost and the schedule of the project. DO-178, for example, identifies specific criticality levels for the software. Level A is the most critical level and E the least. Anomalous behaviour within a system classified as Level A would most likely result in a catastrophic event, e.g., the loss of an aircraft, whereas such behaviour within a system classified as Level C would result 'only' in a major failure condition of the aircraft, e.g., one requiring the pilot to take manual action. Anomalous behaviour within a system classified as Level E would not have any significant effect on the aircraft operation or the pilot's workload. The degree of testing coverage will vary depending on the criticality of the software in these terms. Avionics software classified as Level A requires 100 % MC/DC (Modified Condition/Decision Coverage), which means that every entry and exit point in the program have been invoked at least once, every decision has been analysed for all possible outcomes and each condition in a decision has been shown to independently affect its outcome. Avionics software classified as Level C requires 100 % statement coverage, which means that every statement in the program has been exercised. Table 8.1 illustrates the delta cost and schedule of DO-178 per criticality level.

Whilst the existing standards provide a very good reference, which among other things reduces the need to constantly "reinvent the wheel", they do not address all the practical problems faced by industry:

1. Heterogeneous development environments use a variety of tools, of which most are not integrated. One of the downsides of having a heterogeneous development environment is that the amount of effort required to propagate a change up and down the documentation hierarchy can be high and cost the project a significant amount of money. This propagation is also very error-prone, because it is very difficult to guarantee consistency between different levels of specification or abstraction.
2. Requirements ambiguity is an issue that ultimately impacts on the cost of a programme. In practice, requirements are specified and reviewed using various methodologies according to the applicable standards, but specifications are produced containing requirements which are not testable, or which are ambiguous or incomplete.
3. Late detection of major design issues follows from the previous two issues. Essentially, systems engineering programmes are constrained by commercial pressures which lead to the rushing of specifications generated at early stages of the lifecycle, and then reviews of those specifications are conducted under similar levels of pressure. The end result is the identification of major problems during the test campaign; problems which cannot be solved by simply making a change

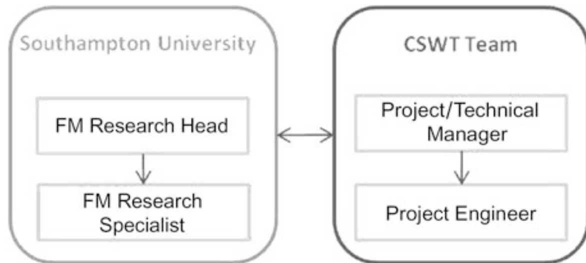


Fig. 8.1 Team composition

in the software code. The problem often goes all the way back to the poor quality of the requirements specification.

8.2 Case Study Overview

The RFD chosen by CSWT for this case study is used to provide attitude, air and navigation information in the event of a primary display failure, supporting pilots' decisions in terms of:

- Determining the correct aircraft attitude, and the exact altitude and airspeed.
- Determining the correct glide slope approach and localiser angles in relation to the runway.
- Determining whether there is a fault in the RFD unit.

From a single display, pilots obtain the following data: Pitch ladder, Indicated Airspeed, Altitude, Vertical speed ascending/descending, Side Slip, Status and fault data. The criticality associated with the software running within the RFD unit is at Level B.

8.2.1 Project Team Structure

The structure of the project team is presented in Fig. 8.1.

The CSWT team had no previous experience with Event-B or Rodin, and therefore went through four training sessions provided by the University of Southampton. These were half-day sessions followed by workshops, with the University of Southampton reviewing the models produced by CSWT and providing further training on specific topics such as model decomposition. The training sessions proved to be very useful and demonstrated on practical examples that the methodology is not theoretical.

8.2.2 Modelling Experience

The starting points for the work were two specifications developed externally: the High-Level Design Specification and the Product Description Specification.

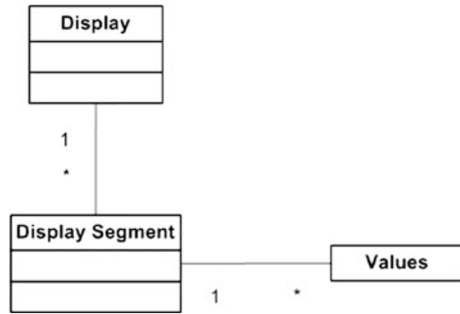


Fig. 8.2 Example class diagram used

The approach adopted was split into several phases, each one essentially a succession of iterations between requirements and the Event-B model:

- Phase I
  - Included the generation of a requirements document from scratch using the existing knowledge of the system
  - Allowed experimentation with the Rodin tool and the creation of some very basic models
- Phase II
  - Refactored requirements specified in the previous phase
  - Generated a new Event-B model covering System Power On/Off Events and Attitude Alignment events
- Phase III
  - Refactored requirements to cover Display properties and the relationship between segments and the display
- Phase IV
  - Created abstract model covering:
    - Display Modes, Segments, Display Data Values and Status sets
    - Events to address updates on values
    - The first refinement created to address range checking and attitude alignment

The earlier phases did not follow a consistent methodology because the input specification was not structured at a suitable level in that the specifications mixed logical aspects of the design with physical aspects. The first method used consisted of modelling the system using a UML class diagram, as shown in Fig. 8.2.

The class diagram proved to be quite useful when reasoning about the main sets in the model and the relationships between them. The UML class diagram enabled the identification of the sets in the Event-B model. Further experience in other projects confirmed that this step is quite valuable as a precursor to the modelling work in Event-B. UML class diagrams facilitate the analysis of the system and the

translation from a natural language specification to an intermediate model that can be easily translated into sets and relationships in the Event-B model.

An extract of the context model created is provided below:

CONTEXT deployRFDV01_C01
SETS

```
MODE
SEGMENT
VALUE
STATUS
```

CONSTANTS

```
Nominal
IBIT
ILS
PitchLadder
RollScalePointer
```

AXIOMS

axm01 : $partition(MODE, \{Nominal\}, \{IBIT\}, \{ILS\})$
axm02 : $partition(STATUS, \{ON\}, \{OFF\})$
axm03 : $SEGMENT = \{PitchLadder, RollScalePointer, BaroCorrection, Airspeed, Altitude, VerticalSpeedAsc\}$
axm04 : $Segment_mode \in SEGMENT \leftrightarrow MODE$
axm05 : $Segment_mode = \{(PitchLadder \mapsto Nominal), (RollScalePointer \mapsto Nominal), (AircraftSymbol \mapsto Nominal), (Airspeed \mapsto Nominal), (Altitude \mapsto Nominal), (VerticalSpeedAsc \mapsto Nominal), (VerticalSpeedDesc \mapsto Nominal), (SideSlip \mapsto Nominal), (Knob \mapsto Nominal), (PitchLadder \mapsto ILS), (RollScalePointer \mapsto ILS), (AircraftSymbol \mapsto ILS)\}$

The following extract provides examples of some invariants modelled:

INVARIANTS

inv01 : $current_mode \in MODE$
inv02 : $active_segments \subseteq SEGMENT$
inv03 : $segment_value \in SEGMENT \rightarrow VALUE$
inv04 : $system_status \in STATUS$
inv05 : $aircraft_speed \in \mathbb{N}$
inv06 : $f_attitude_init \in FLAGS$
inv07 : $alignment_button \in STATUS$
inv05 : $f_reference_system_correction \in FLAGS$
inv07 : $yaw \in \mathbb{Z}$
inv10 : $normal_acceleration \in \mathbb{Z}$
inv11 : $lateral_acceleration \in \mathbb{Z}$
inv12 : $baro_unit \in BARO_UNIT$
inv14 : $f_baro_set \in FLAGS$

The following extract provides examples of some events that were modelled:

EVENTS
Event $changeSpeed \widehat{=}$
begin
act01 : $aircraft_speed :\in \mathbb{N}$
end

```
Event    SystemPowerOn ≙
  when
    guard01 :  system_status = OFF
    guard02 :  aircraft_speed < 1
  then
    act01 :  system_status := ON
    act02 :  current_mode := Nominal
  end

Event    AttitudeAlignment ≙
  when
    grd02 :  pitch < 50
    grd03 :  yaw < 50
    grd04 :  roll < 50
    grd05 :  lateral_acceleration < 32
    grd06 :  normal_acceleration < 125
    grd07 :  normal_acceleration > 75
    grd08 :  system_status = ON
    grd09 :  current_mode = Nominal ∨ current_mode = ILS
  then
    act3 :  f_attitude_alignment := TRUE
  end

Event    PressAlignmentButton ≙
  when
    guard1 :  system_status = ON
    guard2 :  alignment_button = ON
  then
    action1 :  f_reference_system_correction := YES
  end
```

Some of the difficult decisions included how to refine the model and how to decompose it. CSWT's experience showed that it is easy to capture too many elements in the first level of abstraction in the Event-B model, with the consequence that proofs are more difficult to generate and the model becomes more complex. A cookbook developed previously by the Electronics and Computer Science group at the University of Southampton [4] aimed at developing a systematic process for modelling and refining a control problem system by distinguishing between the environment, controller and command phenomena; this proved to be a valuable method of separating concerns and dividing the problem into smaller ones, which are more conducive to a structured approach. The cookbook [4] identifies a pattern that helps separate the controller from monitoring and operator actions. Using this pattern, we identified several events related to the monitoring of sensor values and actuation of monitored values in the display unit (e.g. alarms). It was decided to capture sensors as part of the environment and not to detail sensor behaviour in the model, focusing instead on the display unit. Each of the three main types of events, i.e., the monitoring of values, actions resulting from certain conditions and operator actions, were refined vertically. Despite the number of refinements in this case study being limited, the experience of CSWT shows that the cookbook [4] is a valuable resource in simplifying the refinement process. The cookbook applies mainly to control systems but it can be tailored to other types of systems; for instance, in the context of the case study, the concept of controller does not apply.

Table 8.2 Event-B model metrics

Metric	Value
Number of axioms	8
Number of events	13
Number of invariants	14
Number of refinements	2

The case study developed in the context of DEPLOY provided a good basis for understanding how Event-B can help strengthen requirements. The proof generation mechanism flags up problems with the requirements, and the model checker confirms that the model has been constructed correctly. Future work is being conducted to further explore the use of Event-B to develop Cyber-physical systems[1] [3] and to formally verify properties of COTS-based solutions.

8.2.3 Summary of Results

CSWT's investment in this project was sufficient to allow conclusions to be drawn about the potential of the Event-B method, in particular, in the context of safety- and mission-critical systems engineering. Prior to this activity CSWT had not worked with formal methods but had known of successful applications of formal methods to real cases, such as the Paris metro, A380 avionics and the UK National Air Traffic Services (NATS)—Interim Future Area Control Tool Support (iFACTS), which set high expectations. CSWT's investment was carefully controlled and the model produced was relatively simple, as shown in Table 8.2. Nevertheless, the results achieved were encouraging, and CSWT is already working on two additional projects where the Event-B method is being used.

CSWT's experience with formal methods showed that there are two main advantages to be gained:

- Strengthening requirements: Rodin and the automatic proof generation mechanism highlight problems with the system requirements. Thinking in terms of sets and invariants pushes the system engineer to examine the problem and its solution more carefully and to come up with a more complete set of requirements that can be verified using proof generation.
- Traceability between refinements: Rodin and the refinement mechanism facilitate a breakup of the system into smaller parts and guarantee consistency and traceability between different levels of refinement. The fact that traceability between refinements is ensured by the tool is key to managing large-scale models.

[1] Cyber-physical systems are integrations of computing and physical mechanisms engineered to provide physical services, including transportation, energy distribution, manufacturing, medical care and critical infrastructure management.

There are still a number of outstanding issues, important for CSWT, which could be improved:

- Traceability between requirements and a formal model. Traceability between requirements specified in natural language and a formal model (static and dynamic models) is done manually, CSWT believes this should be automated. There are several tools which provide mechanisms to ensure requirements specified in tool A are synchronised with design models specified in tool B, and any changes in the requirements are propagated to the design model.
- Test case generation from a formal model. The generation of test cases from a formal model would be quite useful in confirming the correctness of the model and facilitating system level verification. Further work is already being done in the context of another FP7 project (see ADVANCE website [3]).
- Team-based formal development. The parallelisation of activities in a design model is required to enable efficient use of project resources and to meet stringent programme timescales. For instance, if the conventional approach was adopted, the system would be broken up into smaller parts and each part allocated to a different team. Each team could then work on its allocated part and various synchronisations points could be defined to better assure the interfaces between the various system parts. CSWT sees refinement and decomposition as essential features for facilitating teamwork, but several levels of refinement may be required before specific sections of the formal model can be allocated to different team members. From the process perspective, it is important to understand how to refine models to enable a quick transition to an environment where engineers can work on models concurrently. The toolset has to support the process; therefore, it is also important that the toolset enables automatic synchronisation of work produced by different teams, for instance, by providing a facility for merging models and highlighting differences between subsets of the formal model.

8.3 Lessons Learned

CSWT's experience in DEPLOY provided a number of lessons to be learnt:

- Definition of scope and system boundaries. It is very important to establish at the start of the project which parts of the system are going to be formalised and which are only supposed to be captured at an abstract level. This becomes quite important in a scenario where a company is taking its first steps with formal models. In his book, Jean-Raymond Abrial [1] presents several examples of how this can be achieved. For instance, in the context of the case study, CSWT decided to exclude any numerical processing done within the RFD, certain user operations such as display brightness adjustments, and everything that is outside the display unit, from the modelling.
- Focus on the principles and set theory. It is very easy to get carried away with the formal language syntax and lose focus on the principles associated with set

theory, refinement and decomposition. It is really important to learn about set theory and to think about the problem in terms of sets from the start. Class diagrams such as the one depicted in Fig. 8.2 and the relationships between different sets have to be well established at the start of the work, and the team should ensure this is the case at the earliest stages of the modelling. If for any reason the team gets stuck in proving a particular invariant, it is recommended that they go back to the drawing board to review the sets and relationships.

- Technology transfer. Training is important but it is even more important to be given a chance to experiment with the methodology on smaller case studies, especially if the engineering team background does not include any formal methods experience. Our work in DEPLOY showed that closely cooperating with a group of professors with several years of experience helped us grasp the concepts associated with formal methods and provided access to other practical examples of their successful application, which would otherwise not have been available.
- Mathematical language requires dedication. The approach adopted requires dedication and regular commitment. The team of engineers lined up to learn and use the methodology has to have time allocated to exercise the methodology week after week; otherwise, the classical development mindset will prevail over the formal methods mindset.
- Benefits of formal methods should be presented from the start. It helps to see case studies where the method has been used successfully, especially if the engineering teams are sceptical about the usefulness of the approach. In his book, Jean-Raymond Abrial [1] presents several real scenarios where Event-B is used.

Acknowledgements Critical Software Technologies would like to thank the DEPLOY consortium for the opportunity to participate in the project and would like to give special thanks to professors Michael Butler and Abdolbaghi Rezazadeh from the University of Southampton.

References

1. Abrial, J.-R.: Modeling in Event-B, System and Software Engineering. Cambridge University Press, Cambridge (2010)
2. Hilderman, V.: DO-178B costs versus benefits. High Rely (2009)
3. Advanced design and verification environment for cyber-physical system engineering. FP7 Information and Communication Technologies Project. Project web site: http://www.advance-ict.eu/index.html
4. Yeganefard, S., Butler, M., Rezazadeh, A.: Evaluation of a guideline by formal modelling of cruise control system in Event-B. ECS, University of Southampton (2010)

Chapter 9
Experience of Deploying Event-B in Industrial Microprocessor Development

Stephen Wright and Kerstin Eder

Abstract The XCore microprocessor is an embedded device developed by XMOS Ltd. of Bristol, UK. The Instruction Set Architecture (ISA) contains a range of typical instructions, for example, for control flow, register-to-register calculation and memory access, but also provides support for efficient multi-threaded programming, parallelism and communication with other devices via fast interconnects. Support for these features is integrated into the ISA of the XCore. This greatly improves run-time performance, at the cost of introducing specialist instructions to the ISA, which comprises 170 instructions. The ISA contains instructions of both two and four byte length, and implements a very compact encoding scheme. The XCore is general-purpose and has been exploited in a range of different markets, including audio, display, communications, robotics and motor control. As part of a Knowledge Transfer Secondment (Grant EP/H500316/1) at the University of Bristol, a formal model of the complete ISA was constructed in Event-B notation, using the Rodin toolset. This project applied the Event-B and Rodin-based techniques for ISA analysis, developed at the University of Bristol, and extended them to an industrial setting.

9.1 Introduction

One of the most fascinating aspects of a computing machine is its instructions, a relatively small set of obscure functions that, when combined in large sequences, form the familiar properties of a computer program. Thus, the same machine may be used to calculate a tax return, play a video game, or keep an otherwise uncontrollable aircraft in the air, simply by changing the seemingly unfathomable sequence of binary digits loaded into its memory. This "Instruction Set Architecture" (ISA) [5] also defines the interface between the hardware and software worlds; it is distinct

S. Wright (✉) · K. Eder
University of Bristol, Department of Computer Science, MVB, Woodland Road, Bristol, BS8 1UB, UK
e-mail: stephen.wright@bris.ac.uk

K. Eder
e-mail: Kerstin.Eder@bristol.ac.uk

A. Romanovsky, M. Thomas (eds.), *Industrial Deployment of System Engineering Methods*, DOI 10.1007/978-3-642-33170-1_9,
© Springer-Verlag Berlin Heidelberg 2013

from the micro-architecture, which is the hardware logic (and sometimes firmware functions) used to implement it. These two worlds are often populated by two distinct sets of engineers, communicating only informally by using natural languages such as English.

The lack of any intuitive connection between the details of a computer's ISA and the behaviour of a functioning program presents another problem: how to clearly define the ISA and how to ensure that all the combinations of instructions that could occur are covered. This is a classic instance of a problem where the sheer number of combinations exceeds our ability to understand and to exhaustively cover all the cases. Formal methods are best suited to overcoming problems of exactly this nature. This difficulty of understanding has also been exacerbated by the appearance of Reduced Instruction Set Computer (RISC) machines, in which the instruction set is designed purely for efficiency, with no thought given to easing the burden of a programmer attempting to write a program or understand the rationale of a knotty instruction sequence. Also, computers are most of their time fair-weather sailors, running programs that work most of the time. This is reflected in the lack of consideration routinely given in practice to error handling when things do go wrong.

In this chapter we introduce the XMOS XCore commercial microprocessor and describe a project which aimed to capture its pre-existing ISA specification formally, using Event-B and Rodin. We explore the need for accurate specification of the device and describe the origins of the project, its schedule, and the formal model that was produced. We then consider the issues that arose in developing a formal specification in an industrial setting, and the issues of introducing these techniques into a pre-existing design and verification flow based on conventional methods.

9.2 The XCore Microprocessor

The XCore microprocessor is an embedded device developed by XMOS Ltd. of Bristol, UK. XMOS is a "fabless" semiconductor company of about 60 employees which was founded in 2005, funded by a combination of enterprise and venture capital. The XCore is general-purpose and has been exploited in a range of different markets, including audio, display, communications, robotics and motor control. The technology is reused in multiple products, including a four-core device that can run up to 32 real time tasks, and a single core device that can run up to eight. The ISA contains a range of typical instructions, e.g. for branching and jumping, register-to-register calculations and memory access, but also provides support for efficient synchronised multi-threaded programming, parallelism and communication with other devices via fast interconnects. Support for these features is integrated into the ISA of the XCore, in contrast to the usual memory-mapped approach. This greatly improves runtime performance, at the cost of introducing specialist instructions to the ISA. The ISA contains instructions of both two and four byte length, and implements a very compact encoding scheme. Some instructions have long and short forms, which are functionally identical but allow greater range in their parameters. Thus the total number of decoded instructions is 209, a relatively small number.

The XCore ISA is published in a specification document [6]; this is similar to many other published ISA specifications, listing each instruction in the alphabetical order of its assembler mnemonic, and containing a mixture of natural language description and semi-formal pseudocode notation.

9.3 The Need for Accurate Specification

The way that microprocessors work implies that loaded binary images are extremely fragile mechanisms: a simple bit flip can cause subtle errors in the selection or parametrisation of an instruction, potentially leading to a catastrophe, even in the absence of any hardware bugs. Such errors are also extremely difficult to locate and correct, because bad values may be stored within the machine and not reveal themselves until many instruction cycles later. Source code debugging features are commonly available with most tools, but these are designed to correct high-level programming mistakes, and themselves depend on ISAs being fully understood. A published ISA specification is the primary reference source for engineers and computer scientists developing and optimising the compiler tools. More directly, functions of a device may be coded in its assembly language: this is particularly common in the mass consumer market. Therefore accurate specification is especially important to developers working away from the original processor designers, such as customers or third-party tools suppliers.

Although not yet relevant to XMOS, microprocessor vendors may and do license the use of an ISA without its proprietary micro-architecture, leaving it to the licensee to provide a separate implementation. In these cases it is vital for ISAs to be correctly defined both in terms of their self-consistency and completeness. This will ensure that all software targeted at the ISA will execute identically, regardless of implementation. The last point is particularly true for ISAs that partially expose micro-architecturally related behaviours, such as early instruction prefetching or scheduler mode changes.

9.4 The XCore Formalisation Project

9.4.1 Project Objective

Our primary objective for the XCore ISA formalisation project was to provide a rigorous restatement of the existing XMOS specification. In particular, we were seeking to identify errors or omissions in the document. These could be direct misstatements of behaviours or formats, omission of possible conditions or ambiguities in possible interpretations.

9.4.2 Project Context

The XMOS project was conducted as a 12-month Knowledge Transfer Secondment (KTS). KTSs are funded by the UK government's Engineering and Physical Sciences Research Council (EPSRC). One objective of the KTS scheme is to place academic researchers in commercial organisations soon after the completion of their studies, so that the results of their research are exploited fully in industry. The XCore ISA formalisation project was an exact fit for the KTS objectives: the project applied and extended the Event-B- and Rodin-based techniques for ISA analysis developed by our doctoral research at the University of Bristol, which derived a formal specification of a small-scale academic ISA, dubbed "MIDAS" [9]. The KTS benefited from the close links between XMOS and the University. XMOS hosted the project for one year from October 2010. As part of the original proposal we planned specific tasks and objectives, and then reported on progress throughout the 12 months. We describe these tasks next, and how they were achieved in the course of the project.

9.4.3 Project Tasks

9.4.3.1 Architecture and Tool Chain Familiarisation

The Architecture and Tool Chain Familiarisation task covered initial familiarisation with all aspects of the XCore, with particular emphasis on the tool support for it in ISA and XMOS [8]. It included studying the ISA documentation, understanding the aspects of the micro-architecture that affect the ISA, and writing test executables to run on the real and software-emulated devices of XMOS. We completed this task, gaining full understanding of all the instructions and their interactions, and were ready for their capture in the formal model.

9.4.3.2 Hand-Coded Virtual Machine

The Hand-Coded Virtual Machine (VM) task developed an ISA-level instruction set emulator written in C, using libraries developed during the MIDAS project. We then reran the test executables created for the initial familiarisation on this new VM, checking our understanding and flagging the inevitable mistakes. We completed this task for the "core" instructions of the ISA, i.e., the conventional operations found in almost all microprocessors. These were sufficient to run basic C programs and allow us to enhance the Event-B methods for completion of the entire ISA. In spite of the seemingly unnecessary duplication of effort, the hand-coded VM and test suite provided an efficient "bootstrap" of the combined tool/model development.

9.4.3.3 Rodin Toolset Review

Given the fast pace of progress in the field, as part of the Rodin Toolset Review task we researched all the advances made by the Rodin development and user community since the MIDAS project, with particular emphasis on editing, automatic refinement, automatic proof discharge techniques and automatic code generation. The latest stable release was installed. The task was successfully completed: a detailed review of the available Rodin toolset was conducted, and key extensions were selected to aid the development process. This task was more complicated than expected. Some extensions were clearly sufficiently ready and capable of boosting productivity, specifically, the newly available text editor [1] and theorem prover "Relevance Filter" [7]. Others were found to be potentially useful but could not be adopted since they were not sufficiently mature.

9.4.3.4 ISA Model Editor

Experience from the MIDAS project made it clear that the default editing tools provided by Rodin would be insufficient for the XCore. Therefore we planned the ISA Model Editor task, i.e., developing an ISA-specific editing tool to allow rapid construction of repeating patterns in events and theorem structures. Fortunately, we were able to provide the functionality of this proposed tool by combining the use of the Rodin text editor to allow rapid copy-and-modify development, and the expansion of the C auto-generation tool to allow more concise statements to be constructed.

9.4.3.5 ISA-Specific Proof Tactics

Another lesson from MIDAS was that, as model size increased, many formal proofs, although conceptually simple, required proof guidance and resulted in frequent time-consuming manual interventions. Therefore we planned the ISA-specific Proof Tactics task to enhance the supplied proving tools with new tactics, aiming to drastically reduce the need for manual proof guidance. Ideally, automatic proof discharge of all theorem proofs would be achieved. We enhanced the tool with a combination of the existing "Relevance Filter" plug-in and a modification of the Rodin core platform to introduce one new automatic proof tactic (possible due to Rodin's open-source release). These improvements were essential, boosting automatic discharge from approximately 10 % to 64 %, but still a long way short of the ideal 100 %, or the 95 % typically achieved with small models by the default tools.

9.4.3.6 Generic Modelling Framework Update

With these groundwork tasks completed, we were ready to begin: with the Generic Modelling Framework Update, we sought to customise and enhance the reusable

generic model developed for MIDAS, by capturing some of the special features of the XCore ISA at an abstract level. Due to tight time constraints, this task was completed partially, to be performed only for the core instructions.

9.4.3.7 XCore ISA Formalisation

The main objective of the project, the creation of a full specification of the XCore ISA in Event-B, had been divided into two subtasks to provide a clear milestone and in recognition of the greater technical complexity of the second part. The first iteration sought to define the basic infrastructure and core instructions of XCore. It was possible to extensively reuse the techniques originally developed for MIDAS. This task was completed well within the planned schedule. The second iteration sought to capture the significant XCore-specific functionality, potentially the entire ISA. Again, we successfully completed this subtask, but it dominated the project and consumed about half its resources, preventing other tasks from being completed. The reasons for this were the complexity of XCore special instructions (particularly multi-thread management and communication), calling for a considerable expansion of the modelling methodology and tools, and the issue of automatic proof discharge not being adequately resolved. The publishing of the complete ISA with all proofs discharged, however, represents a significant success.

9.4.3.8 B2C ISA Refinement and Automatic VM Generation and Test

Our MIDAS experience emphasised the importance of VM generation and testing, and initially we planned the B2C [10] ISA Refinement task to provide a special refinement step to allow automatic code generation via the B2C auto-generation tool, another product of MIDAS. Instead of following our original plan, we found a better solution: the initial intent was fulfilled by enhancing the B2C tool and developing more concise Event-B constructs, allowing a VM to be generated directly from the main formal specification. Although these developments are described separately, in reality the generation and testing of the VM went hand-in-hand with model construction and the expansion of the test-suite used to exercise them. The automatically generated VM allowed the development of a detailed test suite exercising both the generic and special features of the ISA.

9.4.3.9 Formally Derived ISA Specification

We had scheduled a task aimed at generating a natural language document derived from the formal specification, describing the ISA behaviour under all conditions. This was not achieved in the time available, although numerous corrections to the existing specification were identified and reported via the computerised reporting system of XMOS.

9.4.4 Project Results

Detailed documentation describing the structure of the formal model and instructions for its download, build and execution was produced, and all deliverables were made available in the public domain.

Overall, the project followed a familiar pattern for speculative development projects. Tasks resembling previous experience ran inside their anticipated schedules (e.g., construction of generic instructions and automatic code generation). More novel tasks overran their schedules (e.g., multiple register context description and automatic proof discharge). Equally typical were the unforeseen "showstopper" problems that could, without our intervention, have caused the project to fail completely (e.g., the scaling issues within Rodin and underestimating the complexity of XCore-specific functionality). These problems were only overcome by in-project development of new tools and techniques.

9.5 Selecting Event-B as a Formal Method

There are a wide variety of formal methods available: VDM, B-Method and Z are well-known examples. Z specifications and VDM modules allow for the description of individual machines, and Classic-B also allows individual machines to be combined into larger systems. Z notation is only capable of system specification and modelling, whereas B-Method and VDM are specifically intended to allow refinement up to the stage of implementation. VDM and B-Method support similar notions of action definition, although B-Method syntax looks more similar to that of a conventional programming style, using imperative substitutions to represent state transitions, as opposed to VDM's and Z's implicit representation through pre- and post-conditions. Therefore, it is probably more immediately familiar to engineers raised on conventional programming languages, although this is not necessarily an advantage when trying to emphasise the more declarative intent of model notations.

As with MIDAS, the XCore ISA specification was constructed using the Event-B formal notation. Event-B is an evolution of B-Method (now often referred to as Classic-B), originally intended for formal development of software in industrial applications. Event-B supports incremental decomposition of system behaviour into separate guarded atomic actions acting on a defined set of state variables representing the stored state of the system. Static verification of a specification is performed by automatic generation and automatically assisted discharge of Proof Obligations. Event-B's primary advantage is its flexibility, in both the notation itself and its supporting tools. To ensure tool flexibility, Event-B was developed in parallel with a supporting open-source toolset, Rodin. Rodin has a modular architecture based on the Eclipse framework with a clearly defined Application Programming Interface (API), and supports expansion via the addition of Eclipse plug-ins. Rodin is supported by a mixed academic and industrial development community, which has contributed many plug-ins, ranging in function from top-level user interface support to low-level translation of refined models into C.

The selection of B-Method and Event-B in particular was driven by several factors. Event-B is a simplification of Classic-B, decomposing machines into the eponymous discrete events, explicitly linked to their abstractions. This encourages more incremental refinement and easier verification by the generation of more easily discharged proof obligations, enabling the practical construction of larger systems. As already stated, Event-B's primary advantage is its flexibility, in both the notation itself and its supporting tools. Regarding the former, Event-B does not strictly define a notation, but rather a methodology for the construction and analysis of linked logical objects. The notation seen when using tools is actually an arbitrary front end for users familiar with Classic-B and Z notations. The Rodin toolset allows flexibility in the presentation and manipulation of the underlying linked database that represents the loaded model. This feature provides the capability to enhance the notation by adding syntactic sugar and automating repetitive procedures. The tool can be enhanced to expand its scope across the development process, allowing complete end-to-end refinement within a single development environment.

Offsetting the advantages of Event-B and Rodin was their immaturity. Rodin development is proceeding very rapidly but, as this project demonstrated, some functionality is still evolving. The flexibility offered by the Rodin architecture, however, allows immediate shortcomings to be overcome by providing locally produced plugins, reducing risk to development. This is an essential feature in an industrial setting.

9.6 Formal Specification

Modelling in Event-B typically starts from an initially very abstract representation and is conducted in a series of refinement steps, each adding more detail to the model, until the final model is obtained, containing all details required for coding or, ideally, for automatic code generation. In contrast to this "top-down" modelling methodology, which results in a hierarchy of increasingly detailed models, the XCore ISA is "flat", i.e., the complete specification is presented at the full level of detail. This leaves the challenge of developing a meaningful method of decomposition and abstraction, so that "top-down" stepwise refinement could be applied to arrive at a "bottom layer" that represents the full level of detail, i.e., all information necessary to execute compiled binaries. This detail could then be exploited by the C source code auto-generation tool to generate an instruction set emulator VM.

Following the modelling methodology of MIDAS, the entire instruction space of the XCore ISA was initially represented by an abstract set *Inst*. This technique allows all aspects of the representation to be abstracted, including any numerical values that may be assigned by a particular implementation. *Inst* represents all possible instructions that may be presented to the processor at run-time, both valid and invalid. Instruction groups are constructed by the successive partitioning of subsets of *Inst* based on common features, e.g., control flow instructions, instructions which access the internal storage, load/store instructions and so on. This gives rise to a hierarchy of abstraction layers. These subsets are then employed in the guards of

events describing the possible outcomes of executing a particular instruction group. An event describing a successful execution of the instruction is initially created by the appropriate refinement of its abstract event. Complementary events describing all possible failure conditions for the instruction are then derived by the negation of each guard in the successful execution event within the constraints of the corresponding abstract failure event. One failure event is constructed for each negated guard and inherits the actions of the abstract failure event.

In order to assist understanding by a human reader, enhance reuse of repeated constructs, and simplify manipulation, Event-B features were used to partition the specification into many smaller Event-B files. Partitioning of the model was performed both "vertically", i.e., instructions are grouped according to their general properties, and "horizontally", i.e., events are incrementally enriched using the Event-B "extends" mechanism.

The XCore ISA implements 176 operations, some of which have multiple encodings, resulting in 209 possible instructions. Constructing all of these instructions, with all their possible exceptions and a few non-instruction-related events, yielded a total of 690 "events". The formal model of the XCore ISA was structured with the intention of being the basis of a refined abstract model for some or all of the functionality of ISAs in the future.

9.7 Results

As expected, formal construction of the ISA specification revealed several issues in the published document, generally falling into three classes. The first class of issues identified was direct errors, where a detail is unambiguously incorrect, and a compiled executable would never execute correctly under the specified behaviour. An example of this was the transposition of instruction fields in a format descriptor. Secondly, there were ambiguities which were open to misinterpretation and could potentially lead to differences in behaviour between different interpretations, for example modulo arithmetic being used for address-offset calculation but this not being explicitly stated. Finally, omissions of certain functional conditions were discovered, typically esoteric error scenarios. An example of this was the omission of an exception being thrown when the same register index is specified for both destination fields in a dual-destination instruction.

The project yielded a particular curiosity: the XCore ISA has a total of 209 decoded instructions (i.e., 176 operations, with 33 having two encodings which yield separate events). The 690 events of the model therefore implied an instruction/event ratio of 3.3. For the MIDAS ISA, 34 instructions yielded 109 fully decomposed events, implying an instruction/event ratio of 3.2. In the absence of other information, we used the MIDAS-derived figure to estimate the magnitude of the project at its beginning. This would seem to be a reasonable initial estimate when planning modelling of other ISAs and test case coverage projects in the future.

9.8 Tool Experiences

The standard Event-B notation allowed all the required aspects of the ISA to be described, but not efficiently. The model events contain excessive repetition, mitigated by encapsulation of multiple statements into macro axiom statements, and by use of the "extends" function. However, this approach is not compatible with integration into an event refinement hierarchy.

In contrast to the typical verification process for small, complex academic Event-B models, errors in the ISA specification were not generally discovered directly by proof discharge (or lack of it). Instead, a sound description was achieved by careful construction and simulation of events, capturing successful instruction execution across a large but simple model, followed by systematic construction of counter cases (such as exceptions). Proof discharge played an essential role in enforcing model soundness across this large number of events, which rapidly expanded to more than a human "headful". The model yielded a vast number of "well-definedness" proof obligations, and these served to ensure that all the disparate components had been assembled in a compatible manner. Some direct discovery of bugs did, however, occur: an example was the modulo addition implicit in the address-offset calculation, signalled by the failure of the well-definedness proof of its result. On the subject of proof discharge, one conclusion became clear to us: discharge of all proof obligations is essential, as the last few to be discharged within a given piece of functionality tend to be where mistakes are revealed. Without discharge of all proof obligations, a flat model such as this provides little added value beyond that of one coded in a non-formal language.

As anticipated, scaling issues dominated the proof discharge process, causing many proofs that were amenable to automatic discharge to require manual intervention. Considerable effort was put into restructuring the model and enhancing the proof tools to mitigate this. Nonetheless, 36 % of the proof obligations had to be discharged manually, and modifications to the early horizontal partitions of the model had to be considered carefully. The ability of the Rodin tool to cache and reapply proofs after their being trivially altered was of some help, but many changes (such as renaming of variables) would break many proofs in such a way that complete reproving was needed. For a project requiring over 1,700 manual proofs, this was a problem in the industrial domain.

In contrast to many proprietary tools, Rodin provides good support for user and third-party extension development. The open-source model adopted by Rodin enhances these features by supplying copious, easily obtainable examples to serve as the basis of new developments (for example, development of the B2C tool was prompted by studying a rich text document generator). Beyond plug-in development, open-source also provided the ability to modify the core platform itself, which provided an essential escape method for some issues. However, modification of the core requires a further level of expertise beyond that of a plug-in developer. This flexibility allowed the tool to be extended to the point where the entire development process was affected, by inclusion of the VM testing step. The flexibility and fast development of Rodin and its open-source model is offset by its rapid evolution,

requiring some careful management. This can range from API evolution rendering plug-ins incompatible to the need for ongoing review of available tools and their level of maturity, in order to maintain productivity gains.

Event-B models and proofs could only be fully inspected with the help of the Rodin tool. This presented a problem: the company's technical officers wished to review the model as a conventionally formatted document, without the need to install and learn Rodin. Thus the model was manually formatted as an appendix of a Word document, with added structured headings allowing browsing via the table-of-contents.

The existing company design flow includes a non-formal statement of the ISA's functionality, corresponding to the formal restatement provided by this project. This existing specification is stored in a linked database and expressed in XML. Thus, if the formal model were to be integrated into this design flow, Rodin architecture would ensure easy integration by generating XML input files in the existing format. This would be achieved by developing an appropriate plug-in, which is a relatively straightforward process.

9.9 Industrial Perspective

Although it has been a technical and academic success, XMOS does not plan to integrate the formal ISA model into its existing design flow. The company has, however, allowed it to be published as open-source: it has been made available on the website of the EU industrial formal methods DEPLOY project and on XMOS's own open-source website. This approach has stimulated comment and questions in the forum of the tech-savvy customer developers of XMOS. Thus, the model will be further maintained and developed by the academic and industrial communities, and the possibility exists for its uptake in the future, when tools and model are more mature and resources are available. Such an approach is possible for the ISA of the machine, as this is the lowest level of functionality intended for the public domain, which is in contrast with the highly proprietary nature of its micro-architectural implementation.

The goals of the formal ISA project and the setting up of the secondment that implemented it were strongly supported by the company's Chief Technology Officer, motivated by academic interest and longer-term quality goals. Such goals could not be supported in the short term by the product support and development teams elsewhere in the company. In particular, the long-serving design flow maintainer left the company early in the project and his role was split between two replacements, who found themselves fully occupied with simply picking up the existing tools and methodology. Cultural and practical issues such as this cannot be dismissed, as staff turnover and reallocation of responsibilities within a company are common occurrences.

The existing verification methods of XMOS are based on conventional methods, i.e., some static verification of designs supported by EDA tools, complemented by

extensive test suites running on emulators of various complexity as well as on actual silicon. As is to be expected with such a methodology, the company has several functional models of the ISA already, and any attempt to introduce another one will need to be based on significant advantage over these. For example, the importance of complete proof obligation discharge is not immediately apparent to engineers unfamiliar with formal methods, as it provides a form of static analysis not performed with conventional methods. The added value of the additional check is easily grasped by experienced engineers, but needs to be made explicit. In addition to being technically important, formal model-based simulation is also important in another respect. It fulfils the role of existing emulators and thereby instils confidence in the method in an audience familiar with conventional methods.

Most of the errors uncovered in the published ISA specification (such as ambiguous error definitions) would be significant for a safety-critical product, whose resilience to multiple simultaneous system failures is a major factor in the design process, but would not be considered so important for consumer products. This is a fact familiar to any user who has rebooted a PC after an unhandled failure or lost mobile phone connection: unhandled failure is acceptable. Thus current methods are seen to be sufficient for the fast-moving consumer market, and there is little motivation to change to more rigorous methods given their associated risks and costs in time and resources. Once a successful product has been developed, even an agile start-up is reluctant to modify its methodology quickly.

The XMOS secondment also allowed us to gauge the popular perception of formal methods in the engineering community: these were in keeping with the usual observations. Formal methods are generally assumed to be used to construct flat specifications, i.e., without the use of refinement and hierarchical models to capture abstract properties and requirements, and to aid understanding in the construction of well-validated specifications. Theorem proving is usually assumed to be used for static verification of equivalence between the flat specification and a conventional implementation; that theorem proving is used to enforce correctness within the specification, with the help of techniques such as well-definedness and invariant proving, is not common knowledge. Formal methods are also often assumed to replace all or most of the testing; that formal models can be tested dynamically and used for test generation is not common knowledge. This was perhaps demonstrated during the initial definition of the project goals: the industrial engineers were interested in a simple unambiguous statement of the existing specification, whereas the academics were interested in exercising other techniques, such as refinement, simulation and test generation.

9.10 Suggested Improvements

The XCore ISA is one of the largest Event-B models constructed to date, and the experience confirmed a point already understood by the Event-B community: scaling of tools to handle models far larger than academic examples is essential for

the industrial domain. This includes tools for tasks such as editing (by enabling subdivision of models as well as large file handling), building, proving (e.g., by enabling automatic proof discharge for large hypothesis sets) and work allocation (e.g., parallelisation of work and proving to allow teamwork). For example, editing and building performance should be comparable to that of conventional integrated development tools, and automatic proof discharge needs to be approximately 95 % of that achieved for small models. These capabilities are essential in any tool's core platform for providing a firm foundation for any subsequent high-level tools.

Once a reliable, scalable core tool is available, the provision of familiar user interfaces is recommended for gaining acceptance and leveraging existing industrial skills. For example, UML and Simulink front ends are not necessarily the ideal vehicles for presenting the underlying formal techniques being used, but are de facto industry standards and could therefore help achieve these goals. Automated presentation of a model's event and refinement interrelations in graphical and linked document form (such as XML) will also allow efficient reviewing with the help of standard desktop tools.

Finally, reliable support is needed for a successful industrial tool. Academic and open-source tools typically come with free and well-informed but unreliable support from a disparate community, relying on the good will and enthusiasm of its members. For many industrial projects, tool purchase costs are not the major cost of a project, whereas downtime costs due to inadequate support could be. Industrialist project managers are often justifiably nervous about relying on informal support from an unpaid community. The commercial organisations that exist to support (and fairly profit from) the open-source GNU toolset and the Linux operating system are possible examples of a successful model.

9.11 Event-B Modelling Vs. Conventional Formal Verification

It is important to appreciate the added value from modelling in Event-B as opposed to using other formal methods available to industry for verification, such as model checking and, still rather rarely, theorem proving. Model checking, sometimes referred to as property checking, is now used quite routinely in the microelectronics design industry, and has been recently used in embedded software development. Assertion-based design [4] has gained in popularity and is now widely used. Engineers understand the benefits of assertion-based verification; the fact that property checking is fully automatic has greatly improved uptake in industry. State-of-the-art verification environments exploit assertions both during simulation, for monitoring design activity, and for formal property checking. While formal property checking has become an accepted technique for demonstrating the desired properties of a design, in practice the challenge for engineers is to determine whether they have specified "enough" properties, and, also, how much of the design is actually covered by the existing properties. This remains a hot topic of research.

Event-B offers a different, complementary, much more structured and systematic approach that overcomes exactly this problem. Modelling in Event-B starts from

first principles, typically a trivially simple abstraction, whose properties must be preserved during refinement. Formal modelling forces engineers to think when selecting an appropriate representation and, as a consequence, they benefit from the implicit semantics of the constructs they choose. This has the added advantage of not having to define domain-specific semantics for most of the model as this comes "for free". Self-consistency and completeness are inherent in the model by construction, provided all proof obligations have been discharged. Mathematical proof thus serves several purposes during modelling. The first is establishing self-consistency and completeness of the model. The second is ensuring that each refinement step is valid. Both are enforced in Event-B by default through generation of proof obligations. The third is demonstrating domain-specific properties which are explicitly introduced during modelling in the form of invariants and theorems. For example, specific to ISAs are deadlock freedom, i.e., the ISA defining transitions for all input conditions for all processor states, and determinism, i.e., the ISA defining exactly one change to the processor state for each input condition.

For practitioners familiar with state-of-the-art verification approaches that use formal methods such as model checking or theorem proving, the question of *what exactly is being verified with this* (*new*) *method* may arise. This is not an unfamiliar question for us. To answer it we must understand the fundamental difference between *verification* and *modelling*. Verification is the process used to gain confidence in the correctness of a design with respect to its specification [2]. By its very nature, verification requires descriptions of a design at two levels of abstraction: one higher, typically referred to as the specification, and one lower. In addition, a method is needed to establish correctness of the lower-level description with respect to the higher-level one. In the context of microprocessor verification, the higher-level specification is typically the ISA, or a set of properties derived from the ISA, while the lower-level description is often the micro-architecture of the processor. Formal methods include model checking, used predominantly in industry, and theorem proving, used predominantly in academia. Verification establishes whether or not the micro-architecture correctly implements the execution of instruction sequences exactly as specified in the ISA. Verification thus relies on the fundamental assumption that the ISA is functionally correct, self-consistent and complete in that it covers all the behaviours of the processor. In practice, this is very difficult to achieve. In fact, a lot of time is spent resolving inconsistencies and filling omissions in the ISA during micro-architectural design and verification. Recent work has extended coverage metrics so that the degree of completeness of a specification can be established retrospectively [3]. Ideally, however, a specification should be developed in such a way that these important properties are an integral part of it from the outset. This is exactly what can be achieved by *modelling* in Event-B. The final product of an ISA formalisation project in Event-B is both self-consistent and complete and can thus serve as a solid specification at the front end of a traditional verification flow. Even more value can be added to the verification process when developing the next generation of the processor. Typically, such development is done by making extensions to parts of the current ISA, often in isolation. Such extensions, unless rigorously verified, can easily introduce inconsistencies within the newly extended

ISA itself, which then propagate into the design and finally, if undetected during design verification, into the end product. The Event-B ISA model naturally lends itself to modification and extension based on the principle of stepwise refinement. The need to formally establish model consistency between refinement steps, inherent in the Event-B method, guarantees that no inconsistencies are introduced during the ISA extension process.

9.12 Conclusion

In the course of the project we demonstrated that a formal specification of the XCore ISA, a medium-scale microprocessor of approximately 200 instructions, could be successfully developed in one man-year, by extending the methods of our smaller scale academic project. Pareto was right: large amounts of resource were expended on covering a small part of the specification footprint, in this case the specialist instructions of the XCore, and the resulting expansion beyond previous experience. This hot spot also created risk for the project, as new techniques had to be created if we were to cover the entire functionality and legitimately claim success.

Most, but not all of the initially set goals were achieved. The final product even included additional features, specifically, support for simulation and test. These were found to be essential for both technical and cultural reasons. Although a technical and academic success, the project results were not seen to be conclusive or mature enough to warrant integration into the host company's existing, stable design methodology. Perhaps deployment of such a modified design flow requires the resources of a larger company and a window of opportunity presented at the start of a new large project. However, the open-source culture and the infrastructure of the formal method used have allowed the model to be preserved and published outside the company, maintaining the possibility of future deployment and opening the application up for future research.

References

1. Bendisposto, J., Fritz, F., Jastram, M., Leuschel, M., Weigelt, I.: Developing Camille: A text editor for Rodin. Softw. Pract. Exp. **41**(2), 189–198 (2011)
2. Bergeron, J.: Writing Testbenches: Functional Verification of HDL Models, 2nd edn. Kluwer Academic, Norwell (2003)
3. Chockler, H., Halpern, J.Y., Kupferman, O.: What causes a system to satisfy a specification? ACM Trans. Comput. Log. **9**, 20:1–20:26 (2008)
4. Foster, H.D., Krolnik, A.C., Lacey, D.J.: Assertion-Based Design. Springer, Berlin (2003)
5. Hennessy, J., Patterson, D.: Computer Architecture: A Quantitative Approach. Morgan Kaufmann, San Mateo (2003)
6. May, D.: XMOS XS1 Architecture. XMOS Ltd. (2008)
7. Roder, J.: Relevance filters for Event-B. ETH Zurich (2010)
8. Watt, D.: Programming XC on XMOS Devices. XMOS Ltd. (2009)

9. Wright, S., Eder, K.: Using Event-B to construct instruction set architectures. Form. Asp. Comput. **23**(1), 73–89 (2010)
10. Wright, S.: Automatic generation of C from Event-B. In: Workshop on Integration of Model-based Formal Methods and Tools (2009)

Chapter 10
Industrial Deployment of Formal Methods: Trends and Challenges

John Fitzgerald, Juan Bicarregui, Peter Gorm Larsen, and Jim Woodcock

Abstract The DEPLOY project has provided a rare opportunity to explore and document the potential benefits of and challenges to creating and exploiting usable formal methods. Using the results of an updated review of 98 industrial applications, we identify trends relating to analytic power, robustness, stability and usability of tools, as well as to the quality of evidence on costs and benefits of deployment. A consideration of the DEPLOY applications reinforces these trends, additionally emphasising the importance of selecting formalisms suited to the problem domain and of effectively managing traceable links between requirements and models.

10.1 Introduction

The DEPLOY project has provided a rare opportunity to explore and document the benefits of and challenges to creating and applying formal engineering methods in industrial settings. It is fair to ask how the experience gained in the project fits with trends more generally, with experience in other industries and with results obtained using other formalisms. In 2009, we reviewed the state of the art in industry application of formal methods, taking as a basis a survey of 62 projects spanning about 25 years. This gave a picture of how industry applied formalisms and indicated the

J. Fitzgerald (✉)
Newcastle University, Newcastle upon Tyne, UK
e-mail: John.Fitzgerald@ncl.ac.uk

J. Bicarregui
STFC Rutherford Appleton Laboratory, Chilton, UK
e-mail: juan.bicarregui@stfc.ac.uk

P.G. Larsen
Aarhus University, Aarhus, Denmark
e-mail: pgl@iha.dk

J. Woodcock
University of York, York, UK
e-mail: jim@cs.york.ac.uk

A. Romanovsky, M. Thomas (eds.), *Industrial Deployment of System Engineering Methods*, DOI 10.1007/978-3-642-33170-1_10,
© Springer-Verlag Berlin Heidelberg 2013

extent to which concerns identified in earlier surveys from the 1990s, especially regarding tools, education and standards, had been addressed by subsequent research and development. We have now extended our survey to 98 projects, and have sought to relate the experience gained in DEPLOY to the trends identified in this wider review.

In Sect. 10.2 we briefly survey the findings of previous studies, including those from the 1990s and our own review from 2009. In Sect. 10.3 we reassess the state of the art in the light of our extended survey, and in Sect. 10.4 we discuss the experience gained in the deployment studies in relation to the trends that we have identified. We draw conclusions and identify challenges facing successful industry deployment in the future in Sect. 10.5. An extended bibliography lists the sources cited in the chapter as well as reports on many of the projects included in our review.

10.2 Trends in Industry Application of Formal Methods

Surveys and reviews in the 1990s [1, 3, 6, 8] reported significant early successes in the industrial application of formal methods, but also identified inhibitors, including a lack of useful tools and little objective evidence of commercial benefit. It was noted that standards, tools and education would "make or break" industrial adoption [10], and some authors observed a chasm between academics, who see formal methods as "inevitable", and practitioners, who see formal methods as "irrelevant" [9].

Our 2009 study [19] collated data on 62 industrial projects known from the literature, mailing lists and personal experience to have employed formal techniques [2]. A structured questionnaire was used. Contributions were sought from individuals involved in projects rather than from the published literature, and respondents were invited to record their views as free text. The largest single group of applications was in the transport sector (16), followed by the financial sector (12 mainly security-critical applications). There was also significant representation of the defence, telecommunications and office/administration sectors. Regarding application types, the most common were real-time systems (20), distributed computing (17), transaction processing (12) and high data volume (13). About 20 % of the projects concerned software development tools—these could be seen as formalists taking their own medicine. Certification standards applied to 30 % of the projects, notably IEC 61508, and Common Criteria and UK Level E6 for IT Security Evaluation. About one half of respondents estimated the size of the software developed in the project as split roughly equally between 1–10, 10–100, and 100–1,000 KLOC. Two projects were undertaken in the 1980s, 23 in the 1990s, and 37 in 2000–2008. Our review was certainly not a statistical study and does not form a basis for general inferences about formal methods. However, the uniform collection technique did allow projects to be compared and gave insight into a cross-section of domains. In the remainder of this section, we identify major trends highlighted by reviews up to and including our 2009 study.

Lightweight Approaches A lightweight approach appeared to underpin many recent industry applications, in which formal techniques are focused on specific subsystems and on the verification of particular properties [11]. The success of this approach is limited by the quality of tool support. We expected to see increasingly widespread use of enhanced static analysis tools incorporating model checking or proof support.

Tool Support In 1993, it was observed that tools are neither necessary nor sufficient for the successful application of formal methods [1], but by 2009 it was almost inconceivable that an industrial application would proceed without tools. However, although tool capabilities continuously improve, surveys, and about a quarter of the responses in our 2009 review, report a lack of robustness in tooling. Particular challenges include lack of support for human interaction in automated deduction, of common formats for model interchange and analysis, of support for "best effort" tools that point out likely errors, and the need to integrate heterogeneous tools into tool chains. Expectations of tools remain high, with respondents emphasising the need for tools to return the results of analyses in "seconds or minutes rather than days" and demanding tight integration of tools such as model checkers into the development environment. In general, tool usability was also criticised by many respondents, e.g., "Tools don't lend themselves to use by mere mortals". Increased automation and usability of sophisticated verification tools is a great challenge.

Increasing Levels of Automation The increasing use of model checkers rather than interactive theorem provers bears witness to the value of highly automated analysis. We observed an increasing interest in developing push-button technology whereby the trade-off between precision and cost in correctness analysis can be controlled by a few parameters. In 1999, Bloomfield and Craigen pointed out the value of this approach as an entry point to formal methods for first adopters [3].

Evidence to Support Adoption Decisions Previous surveys identified lack of convincing evidence to support decisions to adopt formal techniques [1], and paucity of appropriate cost models [7]. Only half of the projects that we reviewed in 2009 reported on the cost implications of using formal methods. We conjecture that this was either because financial data is considered sensitive or because projects were not collecting this kind of data, or did not trust it as a basis for long-term adoption decisions, once collected. The decision to adopt a particular development technology is risk-based, and convincing qualitative evidence of defect detection may be at least as powerful as a quantitative cost argument. The construction of a strong body of evidence about the effectiveness and usability of formal methods is at least as important as data on costs.

Evidence of Second and Subsequent Use The often-stated view that the entry cost for formal methods is rather high was borne out by some responses in our survey, but it has also been observed that repeated use can dramatically reduce costs [15]. Nevertheless, very few published experience reports describe repeated application, although 73 % of respondents in our 2009 study indicated that they would use similar methods again.

Skills and Psychological Barriers Responses to our survey uncovered several psychological or cultural barriers to the use of formal methods, such as the desire to see constructive results quickly, rather than after long periods of analytic work. Skill deficiencies were also reported as a major obstacle, suggesting that improved education and training remain vital to success.

Within the 2009 data set, correlations were observed between the techniques used and the types of software being developed, with model checking being more popular in consumer electronics and inspection more popular in transaction processing software. Furthermore, the use of model checking increased from 13 % of the projects surveyed in the 1990s to 51 % in the first decade of the 2000s. No significant change over time was found in the use of proof, refinement, execution or test case generation. Regarding the expertise of the teams undertaking the work, most project staff had prior formal methods experience (85 % of projects), and, of those reporting no previous expertise, half were in teams with mixed experience levels and half were introducing techniques to a novice team. Over half the responses indicated that training had been given. Many projects reported that the time required to complete a project was reduced by using formal methods: three times as many reported a reduction compared to those that reported an increase. Several projects noted increased time required in the specification phase. Many projects reported that projects costs were decreased, with five times as many reporting a reduction in cost as those that reported increases. Nearly all projects (92 %) reported enhanced quality compared to other techniques; none reported a decrease in quality. Quality improvement was due to better fault detection (36 %), improved design (12 %), assurance of correctness (10 %) and understanding (10 %).

These reviews reveal a landscape of industrial applications of formal methods where a diverse range of application domains are benefiting from formal modelling and verification technologies. Furthermore, a number of obstacles identified in the past are being overcome, particularly due to the availability of advanced tools exploiting increased computing power. However, many challenges remain, such as making tools more usable, integrating them into industrial development processes, providing evidence to support second and subsequent use, and overcoming lack of skills and other barriers to adoption.

10.3 The Position in 2012

In this section we present the results of our ongoing collection of data from industrial applications, using the same structured questionnaire that was employed in our 2009 survey. The analysis is based on data collected from a total of 98 projects. This includes 60 of the projects previously reported, several of which have been updated, and more projects that have completed questionnaires since then. Two of the records from the first dataset have been superseded as a result of the updating and data cleaning exercise. As was done in the report in 2009, we examine project timescales and size, application domains, techniques used, practitioners' views of effects on time, cost and quality, and overall satisfaction with the techniques used.

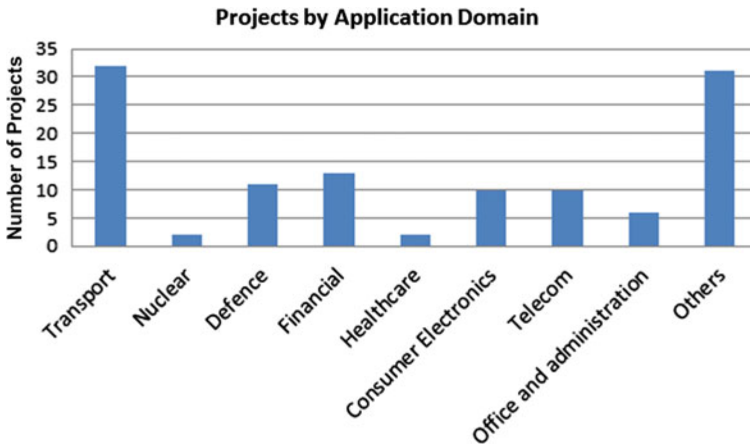


Fig. 10.1 2012 dataset—Projects by application domain

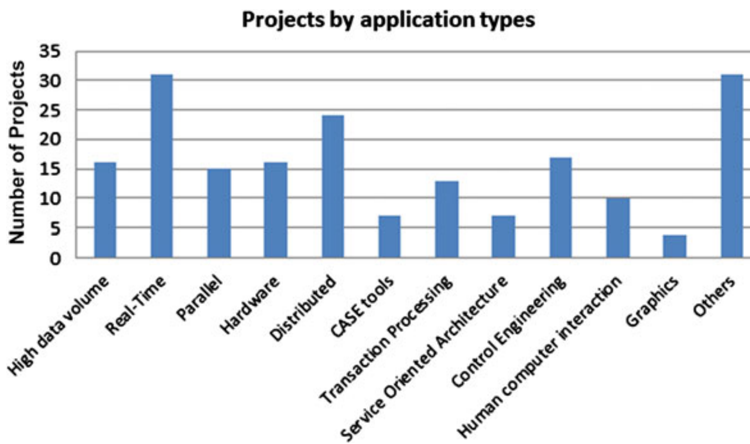


Fig. 10.2 2012 dataset—Projects by application type

10.3.1 Project Characteristics

Figures 10.1–10.3 show key characteristics of the 98 projects in the 2012 data set. The numbers of projects in the transport and consumer electronics domains are more than twice as large as those in the 2009 review (Fig. 10.1). Compared to the smaller 2009 data set, there is increased representation of real-time and distributed applications (Fig. 10.2), but the distribution of formal techniques applied is, however, quite similar to that of the 2009 set (Fig. 10.3).

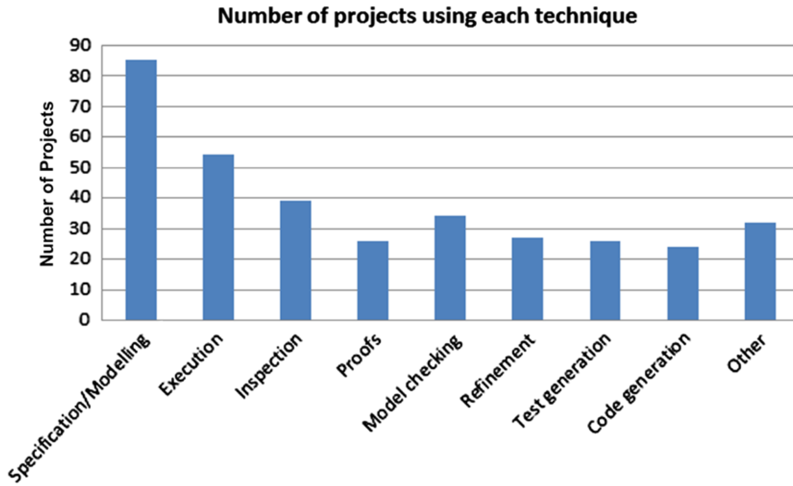


Fig. 10.3 2012 dataset—Projects by formal technique used

Fig. 10.4 2012 team experience

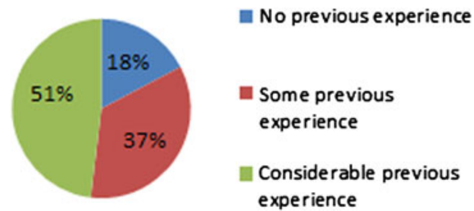
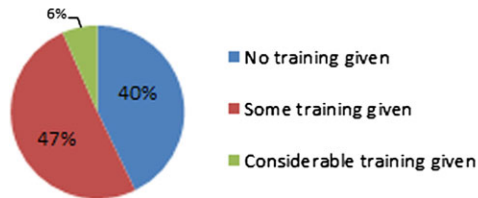


Fig. 10.5 2012 training given



10.3.2 Development Team Characteristics

Figure 10.4 indicates how experienced project teams are in formal methods. Over half the projects in the database reported that developers had considerable previous experience in formal methods, although this is partly due to the fact that some of the teams included in the set provided reports on several projects. Perhaps for the same reason, the level of training provided is rather limited, with less than half of the projects reporting providing training for the team in Fig. 10.5.

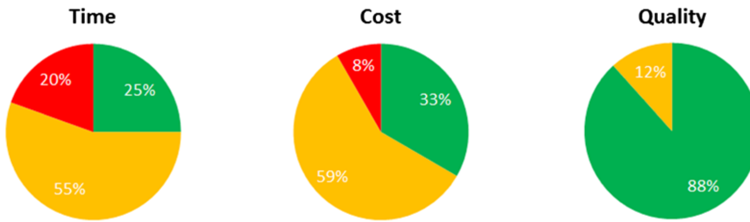


Fig. 10.6 Reported effects of use of formal methods on time, cost and quality

10.3.3 The Impact of Formal Methods on Time, Cost and Quality

Respondents were asked for their view on whether the use of formal techniques had had an effect on the duration of the project (Fig. 10.6). Responses were mixed, with 25 % reporting a decrease and 20 % reporting an increase in time. Noticeably, over half the respondents (55 %) either reported no change, or did not know the effect on time, with this group being roughly equally split between the two.

Respondents were also asked whether the use of formal techniques had an effect on the development cost or the development cost profile (Fig. 10.6). Here positive replies (33 %) outweighed negative replies (8 %) by over 4 to 1. However, again a majority (59 %) did not report any effect.

The reported effect on quality was much less equivocal. When asked whether the use of formal techniques had an effect on product quality (requirements, design, code, test, safety case), the vast majority of replies were positive (88 %) with no negative replies and 12 % unsure (Fig. 10.6). This would tend to confirm the tenet that software quality is the main driver for the use of formal techniques and aligns well with the results of our 2009 review, but must be viewed with caution in that some of those choosing to contribute project reports to the dataset may have been self-selected proponents of formal methods.

10.3.4 Overall Satisfaction with the Use of Formal Methods

Three questions were asked concerning overall satisfaction with the use of formal techniques. Respondents were asked how strongly they agreed with three positive statements about the use of FM:

- The use of formal methods in this project was successful.
- The techniques used were appropriate for the tasks undertaken.
- The tools used were able to cope with the tasks undertaken.

They were also asked whether they would repeat the use of FM:

- Will you be using the same or similar methodology again?

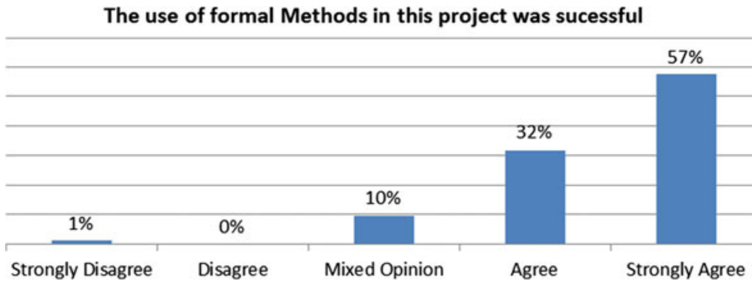


Fig. 10.7 Perceived overall success of the use of formal methods

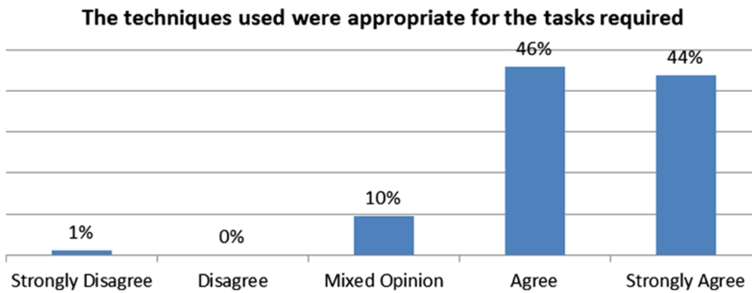


Fig. 10.8 Appropriateness of the techniques used

Figure 10.7 shows the responses with regard to the overall success of using FM. 89 % of respondents either agreed or strongly agreed with the statement, compared to just 1 % who disagreed. One of the free text responses explains the perceived benefits of abstraction:

> I can state that the formal thinking (or methodology) helps a lot during the development process even if the formal method itself is not fully used. It helped the development team in thinking about the model and not about the implementation itself, so the result was a clear solution for the problem that was being studied.

Another contributor describes the benefits seen in test automation:

> [The product] is a standard that allows thousands of different parameters to configure the software. The behaviour of the software varies depending on these parameters. A test case is both a configuration and a sequence of actions in that configuration. An attempt to write test cases [. . .] was a complete failure: the tests cannot be used in practice, since they are not parameterisable with the configuration. A model can easily be made parameterisable and we are therefore able to use the same model to generate test cases for millions of different configurations. In other words, we can test for that configuration that the. . . company is interested in.

The appropriateness of the techniques used also received very strong support, with 90 % agreeing with the statement, compared to just 1 % disagreeing (Fig. 10.8).

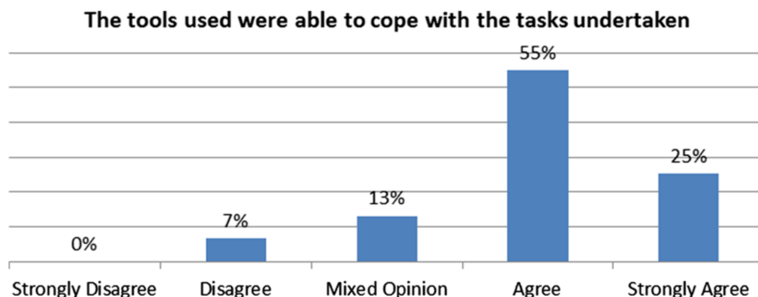


Fig. 10.9 Adequacy of the tools used

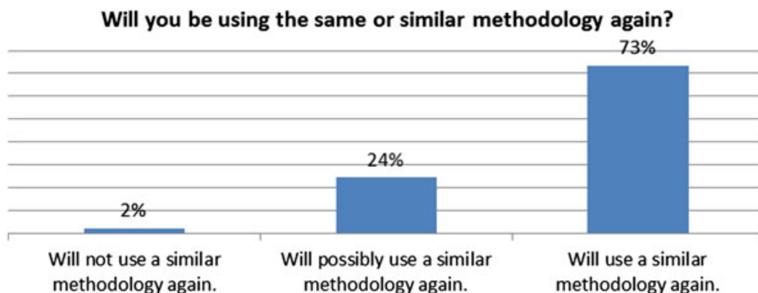


Fig. 10.10 Intention to use formal methods again

One contributor writes of the benefits of automated test generation:

> The original developers could not detect the faults discovered in this work, because these faults cause errors only in corner-case/unexpected scenarios. It is very difficult for a human engineer to detect faults that are manifest only in exceptional scenarios through manual testing…

The adequacy of the tools for coping with the applications also received strong support (80 %), although a significant minority (7 %) did disagree with the statement (Fig. 10.9). Of the projects reporting inadequate tools, the median start date was 2000, compared to a median start date of 2006 for our sample of projects overall, suggesting that this may have been less of a problem in recent years.

Figure 10.10 summarises the responses regarding intention to use a similar technology again: 73 % of replies were positive and 2 % were negative.

A notable comment from one contributor was:

> …formal methods were used to differentiate our bid and team from competitors; their use is why we won the contract.

On the other hand, several responses indicated a shortage of appropriately skilled personnel. For example:

> This project was a very specialized “bug finding campaign”. It required very high skills (which were fortunately available), and therefore we expect that similar methods will never

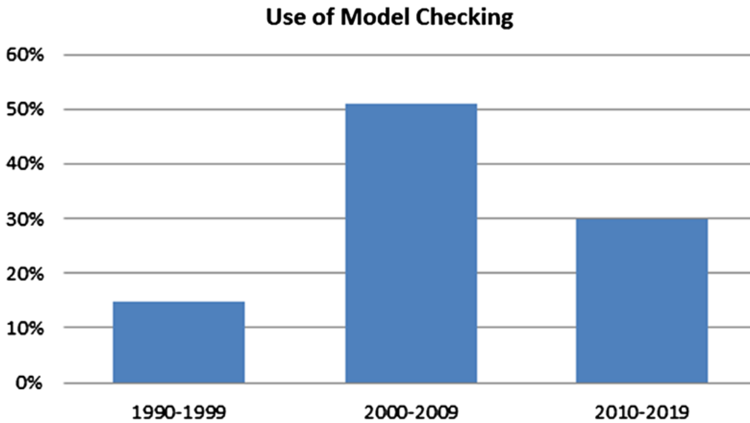


Fig. 10.11 Use of model checking

> be used on broader scale as "standard techniques" during system development and verification.

And another contributor wrote:

> I think that the main barrier is a lack of motivated people with FMs background in SW/HW development teams. It could be overcome by little reorganisation of teams and development processes. Teams should be slightly (+10 %) supplemented by FMs specialists (they can be used for requirements analysis, formal specification and verification of most critical parts of a system). Lightweight techniques and user-friendly tools simplify introduction of FMs.

10.3.5 Trends in the Use of Formal Techniques

The use of each technique was correlated against the project start date. As in the 2009 survey, a strong association was found between project start date and the use of model checking, with model checking being used in just 15 % of reported projects in the 1990s, compared to 51 % and 30 % in the 2000s and 2010s. The new data confirm the earlier conclusion that there was significant rise in the use of model checking in the reported projects over the period 1990–2009 ($p = 0.01$), although this rise would seem to have stabilised or even dropped since 2010 (see Fig. 10.11).

The new data also confirm the tentative conclusion drawn in 2009 that the use of test generation may also be increasing. This technique was used in 50 % of reported projects that started after 2010, compared to 7 % and 34 % in the 1990s and 2000s respectively. This increase is significant at the 1 % level (Fig. 10.12). Regarding the use of test generation, one contributor reports:

> Less effort to create test cases. Also, it is easier to keep the test suites up to date by automatically regenerating the test suite when the requirements change (use case models). The reduction is about half in the effort for test generation.

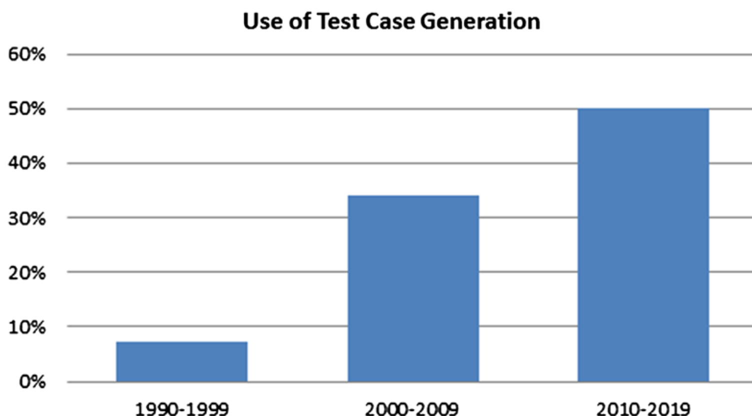


Fig. 10.12 Use of test case generation

The data set shows a mild decrease in the proportion of projects using specification/modelling, with this technique being used in 96 % of reported projects in the 1990s and 92 % in the 2000s, but only 70 % in the 2010s. This reduction was significant at the 5 % level. However, this is mostly due to a group of projects using test generation from C code, which also accounts for the decrease in the use of specification observed above. No other significant trends were found.

10.3.6 Rockwell Collins

In 2009 we commented on a lack of reported evidence from second and subsequent applications of formal techniques within organisations. Our extended data set now includes evidence on several studies conducted at Rockwell Collins. These include verification of the microcode in the AAMP5 [18] and AAMP-FV [13] microprocessors using the PVS theorem prover [17], verification of the intrinsic partitioning mechanism in the AAMP7G microprocessor using the ACL2 theorem prover [4], verification of the FCS 5000 Flight Control System mode logic [14] and the ADGS-2100 Adaptive Display System Window Manager [16] using the NuSMV model checker [5], verification of the Redundancy Management logic in an adaptive flight control system using NuSMV, and verification of the Effector Blendor in an adaptive flight control system using the Prover plug-in model checker. The contributors report:

> The primary lesson from the AAMP-FV project was that it is very hard to estimate the cost of formal verification from initial experiments. As with anything else, costs drop dramatically as techniques, tools, and skills are developed. The AAMP-FV experiment also pointed out that it is easier for the developers to master the verification tools than it is for experts in formal verification to master a complex product domain.

We have also seen from other contributions that when the domain experts get acquainted with formal techniques and tools, costs of using them drop significantly.

This means that if one measures the cost of using such a technology on a pilot project, this will not be accurate since on multiple usages the cost will most likely drop significantly. They also reported:

> An important lesson we learned from this experiment was to automatically translate the models the engineers used for code generation rather than to create a verification model by hand. This eliminated the tedious process of trying to keep the engineering model and verification model in sync and enabled us to rerun the verification quickly each time the engineering model was changed. Being able to re-verify quickly after each modification and to find errors rapidly built confidence with the engineers that the tools worked and could find real errors.

This corresponds to the trend we have been seeing for an increase in the automation and support with tools. There is no doubt that unless one can see a clear benefit from maintaining multiple models and keeping them in sync, this will not be done. It is a non-trivial challenge for the tool community to enable such support. Despite this, the contributors also report:

> However, the most important lesson we learned was that industrial systems have significant portions of their logic that can be verified with today's model checkers. Even if the entire system can't be formally verified, there are almost always large portions that are amenable to model checking or that can be made amenable with some small changes. This has been the case in almost every system we have looked at, leading us to conclude that model checking could be used in far more places than is the case today.

Thus, this agrees with what we have seen from other contributions, where the existing tool support for formal methods such as model checkers is actually good enough to address, to a large extent, the complexity in current industrial systems.

10.3.7 Observations from the 2012 Review

Trends that we identified in 2009 still appear to hold good in the extended dataset. In particular, we note that the robustness of tools and the ability to cope with scale also seem to have improved in recent years. Continued increases in analytic power continue to improve levels of automation in analysis and "invisibility" of tools to their users. There is evidence again for cost-effectiveness, with most projects reporting benefit from little or no extra cost. The potential for second and subsequent use is clearer with the new data, as illustrated by the Rockwell Collins case, which reports productivity improvements of around an order of magnitude between the first use and the second use of formal modelling and verification. However, there remain some perceived skills-related and psychological barriers, in particular surrounding the level of expertise required to deploy formal methods successfully.

10.4 The DEPLOY Deployments in Context

In this section, we relate the experiences of the seven deployment studies reported in the preceding chapters to the features identified in our review of the landscape of

industry application. It is important to note some aspects of the DEPLOY studies. First, the tools, and the methods around them, were in a relatively early stage of development and evolved during the deployments. Indeed, further development of the methods and tools was a goal of DEPLOY. Second, in most cases the developers had little or no prior experience of formal methods. Neither of these factors are unique to the DEPLOY projects, and many of the projects reported in our 2012 data set were tackled by teams with limited previous experience.

10.4.1 Automotive Industry Deployment

In common with many of the larger studies in DEPLOY, the automotive sector study based at Bosch used a staged approach, scaling up from a mini-pilot, through a pilot to an enhanced pilot project, as their expertise and experience grew.

The application was always intended to be lightweight in the sense of Sect. 10.2 [11]. Although a fully formal notation was used, it was always expected that its use would be targeted to selected aspects of the development process. It rapidly became clear that the level of abstraction in existing requirements descriptions was such that a direct transformation to a formal model was not likely to prove cost-effective. This led to the development of a staged modelling process which very effectively exploited Jackson's Problem Frames to manage the complexity and scale of the requirements before proceeding to formalisation. The technique was particularly effective at managing the crucial separation between controller, controlled plant and environment.

The pilot deployments highlighted the need to ensure a good match between the expressive strengths of the formalism, and those required by the problem domain. Many of the relevant automotive domain problems concern real-time properties such as responsiveness, and the utilisation of closed-loop controllers. Neither feature is well modelled in Event-B. The authors go on to comment on the difficulty of separating discrete and continuous parts of the pilot deployment cruise control.

The study reinforced the comments made by contributors to our 2009 and 2012 surveys that scalability supported by tools is essential for industrial deployment. One conclusion of the DEPLOY automotive study was that the size of the models produced presented a considerable challenge for the tools: the comment made was that techniques based on "classroom examples" may not work on an industrial scale. For example, refinement from a specification to an implementation may involve a large number of different levels of abstraction, each introducing a small, but significant, design decision. The software engineering principle of separation of concerns would dictate that these decisions should indeed be taken individually, but this may not be practicable. In spite of tools scalability problems, it is worth noting that the adaptability of the Rodin tools, based on a plug-in architecture, was found to be valuable in the study. Bosch engineers are already experienced in the use of state charts, and have found them to be a valuable tool. There is no support for their use in the Rodin toolset, but the authors recommended that they be added as an interface

to ProB. A more general comment is that tools tailored to model construction for particular classes of system are particularly valuable.

Regarding evidence for cost-effectiveness, successful use of formal methods in an industrial context requires adequate management processes, including project planning. The authors report that they found it difficult to estimate the effort required for requirements engineering, modelling, and proving. Requirements traceability is another important aspect of project management, and the team experienced the trade-off between preserving the architecture of the requirements and developing a new architecture for the implementation. Formal methods tools need to be further strengthened with more support for state-of-the-art industrial development processes, particularly for configuration management.

With respect to skill levels and other barriers to deployment, the report notes that there is considerable value in having dedicated support during the first deployment. The study also appears to have benefited from staff stability, including the "mentoring team" from the method and tool development group, who supported them.

10.4.2 Transportation Deployment

In contrast with the Bosch automotive case, the Siemens deployment is not a first-time application of formal engineering methods, and there were models to be reused there. The deployment was focused on the specific modelling and development tasks associated with high integrity SIL 4 subsystems. The project had a strong focus on a specific technical problem related to data validation.

The authors of this study conclude that the Event-B and Rodin tools were not sufficiently mature to be committed to critical developments. The authors stress that because it is expensive in terms of time and money to adapt models, it should be sufficient to carry out minimum adaptation for successful applications to take account of the requirements of new tools. Rather than their existing models being modified, the ProB tool was modified. The chapter explains several substantial changes that were made to the Rodin tool base to accommodate the size of model that needed to be analysed. The application to the SIL 4 development raised issues about the validation of ProB within the chain.

In contrast with many other projects, the transport deployment reports compelling evidence of the effort saved as a result of the automated checking: one person-month of effort was replaced by 17 minutes of computation.

10.4.3 Space Sector Deployment

In common with other deployment studies, a staged approach was taken in evaluating Event-B/Rodin in the space sector. A wide range of studies was conducted on specific aspects of modelling and verification in this setting. The overall findings are

mixed, and are not strong enough to support immediate adoption of formal methods for routine development, although serious consideration is still being given to this option, and several positive results are taken from the study.

The project raises interesting questions about the suitability of Event-B (or any similar formalism) as a modelling technique for SSF's application domain. In the OBSW modelling study, it was apparent that a model checker was a better match to the need (to analyse dynamic properties) than Event-B, which would be better suited to the modelling of complex safety invariants. In the AOCS study, requirements composed from Ada code modules were not naturally represented in Event-B. This is hardly a surprise since this approach deviates so markedly from the model-based development paradigm for which Event-B and Rodin were conceived. The report observes that state invariants do not seem particularly useful for describing a system that is naturally event-driven, and that a property of interest is often expressible as a temporal logic formula, but not as an invariant. Trace-based specifications and dynamic invariants may well address this concern. The authors are sceptical about an emphasis on modelling at a level of abstraction significantly higher than that of code, citing the volume of code-level verification work as their justification. On a detailed modelling and verification issue, they point out the lack of support for inductive types in Event-B and the consequent lack of support for induction. The project required support for real-time modelling, which is not directly available in Event-B.

With respect to tool support, the deployment studies in SSF reinforce our comments in the 2009 and 2012 surveys that lack of scalability can present a real impediment to adoption. The successful application of formal techniques in this setting relies on a stronger link being established with conventional practices. This means prompt and clear error reporting from verification, code generation support, and domain-specific modelling. There was a positive experience with the use of modularisation to facilitate teamwork.

SSF provided an extremely highly qualified internal deployment team: of the seven persons listed, five have doctoral or postdoctoral experience with formal techniques. In common with other deployment studies, the practitioners in SSF found that, while initial intensive training was satisfactory, a longer period of closer support is essential if users are to overcome the natural difficulties in using advanced features. The conclusions of the study add weight to the claims arising from our other surveyed projects that the chances for success of formal engineering methods are greatly improved by strengthening their links to established software development practices.

10.4.4 Deployment in the Business Domain

SAP described what our review data would suggest is an almost exemplary application of "lightweight" formal methods in their deployment studies. Rather than attempt to develop a universal solution to model-based development, SAP developed

a suite of related but specific tools addressing choreography, business process modelling and model-based testing. The experience was largely positive. Comparing this with the space sector deployment, it can be argued that the increasing acceptance of model-based approaches in the business systems sector (through informal or semi-formal graphically based modelling approaches such as BPMN) provides an already favourable environment in which to inject an element of formal verification.

The study built on a tradition of developing proprietary domain-specific languages. Deep integration with the existing model types was required, but a new language was reasonably readily accepted once its analytic advantages were clear. The new language's graphical representation was deliberately chosen to integrate well with other in-house languages.

Tools were built with the intention of rolling them out for mass use by developers. This implied that the formal elements should be as far as possible non-intrusive. Adapters were built in order to integrate Event-B and Pro-B with well-established environments. In the business process modelling work, it is worth noting the two goals for the BPMN to Event-B translation, namely the preservation of the BPMN model structure in Event-B (to ease interpretation of verification results) and maximum automation in the discharging of proof obligations. Indeed, a high degree of automation was an essential objective of the MCM (message choreography model), BPMN and model-based testing work, in order to achieve a "non-intrusive" introduction of formal methods. Although the level of auto-proving did not get as high as the developers might have wished, the outcome is nevertheless considered highly promising.

Even with the goal of non-intrusiveness, it was necessary to have a careful programme of evaluation in place using seven product development teams. It was notable that there was no significant expertise in formal methods in the model-based testing study.

In terms of cost-effectiveness, the jury may still be out on the proof-based analysis of business process and message choreography models (principally because of the automation levels and expertise required for prover guidance), but the model-based scenario testing tools are being rolled out across multiple product teams.

10.4.5 Deployment in the Railway Domain: Grupo AeS

Grupo AeS undertook a series of targeted case studies using Event-B/Rodin. Although these were targeted applications, this does not rule out the possibility of a completely formal development process being applied. Early attempts at formal modelling and reasoning helped inconsistency and incompleteness in requirements to be identified—a benefit shared with other projects in our 2009 and 2012 survey data sets.

An initial, subjective comparison of development tools suggests clear benefits for tools such as SCADE that benefit from a considerable investment in prosaic, but pragmatically and commercially significant aspects. In common with the automotive

study in Bosch, AeS felt that the mapping from requirements to formal modelling required some tool support at a pre-formalisation stage. This led to an approach driven by use case analysis, and ultimately to the development of the domain-specific VeRaSiS plug-in for rail specifications.

10.4.6 On-board Systems Deployment: Critical Software Technologies

Critical Software Technologies (CSWT) undertook a case study involving the Integrated Secondary Flight Display (ISFD) used on-board commercial and military aircraft, developed to DO-178B Level B (second-highest criticality) specifications. In addition to drawing conclusions about the use of Event-B in their chosen application domain, the authors describe the cost-effectiveness of using formal methods in the verification and validation processes in the industry standards DO-178 and ECSS-E40. The ISFD case study started from two externally developed specifications: the High-Level Design Specification and a Product Description Specification. The project team developed their system model in a series of phases, each one designed to formalise different aspects of the requirements. The team used UML class diagrams to help structure their Event-B model and guide its formalisation.

The authors report the difficulty they experienced in deciding how to decompose and refine their models. The University of Southampton provided a cookbook of patterns, and the project team found this to be particularly useful. This cookbook distilled previous experience in modelling and refining control problems, distinguishing environment, controller, and command phenomena, although, interestingly, there was no controller in the ISFD case study.

Several lessons were learnt from the case study, including the importance of determining an appropriate system boundary in order to get a useful model. Less formal class diagrams were helpful in understanding the relationships between the model elements.

The cost-effectiveness argument for this case study focuses on reducing the risk associated with errors arising from poor quality requirements, leading to significant effort being needed to propagate changes up and down the documentation hierarchy—an error-prone activity. Costs are increased if a requirements specification is untestable, ambiguous or incomplete. Commercial pressures dictate that these specifications are released prematurely, resulting in late discovery of errors, which often cannot be fixed at the code level.

The authors report that the case study demonstrates that the use of Event-B can help strengthen requirements. First, abstract modelling focuses attention on the problem and its solution, leading to more complete requirements. Proof and model checking are both important for discovering anomalies in requirements. Traceability between refinements, ensured by the Rodin tool, is key to managing large-scale models. The authors would like to have support for automated test-case generation

from formal models. They also report that CSWT is already using Event-B in two additional projects.

The CSWT team had no previous experience with Event-B or Rodin, and so attended four half-day training sessions provided by a university partner. Each session was followed by a workshop reviewing CSWT's models, and involved additional training on specific relevant topics. The authors identify this training as particularly useful, especially in bridging the credibility gap between classroom theory and industrial practice. They nevertheless comment that longer-term commitment is necessary to maintain a "formal methods mindset". The value of targeted assistance from experienced practitioners is emphasised in several projects in our survey database, and echoed again here. CSWT also comment that the benefits of formalisation, based on collated experience, should be clearly articulated from the outset.

10.4.7 Instruction Set Architecture: XMOS

This study concerns the development of a formal model of the complete instruction set architecture (ISA) for the XCore microprocessor, constructed in Event-B using the Rodin toolset. The objective was to establish an accurate specification of the ISA, which is especially important to customers and third-party tools suppliers. The formal specification of the XCore ISA, a medium-scale microprocessor of approximately 200 instructions, was successfully developed in one man-year, building on a smaller scale academic project.

The authors report that the default Rodin editing tools proved to be insufficient for XCore, and they developed a specific ISA Model Editor. They found that the use of Eclipse plug-ins was essential to overcome Rodin's immaturity.

The formalisation of the model uncovered errors and ambiguities in the XCore description. The authors considered that the discharge of all proof obligations is essential, as errors often lurk in the last few to be discharged. They report that size is a dominating factor in the automation of proof, and that a lot of effort had to be made to restructure the model and enhance proof tools to mitigate this. But 36 % of proof obligations had to be discharged manually (about 1,700 proofs)—a somewhat higher proportion than is typical of the other Event-B deployments.

Concerning integration with established processes, Rodin did not immediately fit into the company's review process, and documentation had to be manually generated. The completed formal ISA model did not become part of company's design flow. Errors uncovered by formal methods would have been significant for a safety-critical system, but were not so important for a consumer product.

The authors report concern about the level of support available for academic tools, compared with production-quality industrial tools. They would also like to have support for industry-standard UML and Simulink front ends.

The study provided an opportunity to assess perceptions of formal methods. Some of the assumptions expressed (for example, that formal methods are used to derive models at a single level of abstraction) and a lack of awareness (for example,

of the potential for automated verification of model consistency) contribute to the perceived barriers to adoption reported by other projects in our survey data set.

10.5 Conclusions

In Sect. 10.3 we gave an update on our general review of industry application of formal methods. Broadly, the trends that we identified in 2009 still apply in 2012: the use of lightweight, targeted formal methods, the crucial importance of tool support, the increasing trend towards fully or largely automated analysis, the continuing need for evidence of cost-effectiveness on which to make risk-based adoption decisions as well as evidence of second and subsequent use, and a range of skills and cultural or psychological barriers. To some extent, the project data on which we base our observations are derived from self-selecting "insider" views within the formal methods community. The strength of the DEPLOY stories summarised in Sect. 10.4 is that they mainly reflect the experience of organisations that are, to a large extent, new to the use of formal methods and do not have a stake in the success of particular formalisms. In this sense, the DEPLOY studies have shed a colder light on industry deployment of formal methods than much of the literature. What have these studies told us about the present and future of industrial application of formal methods?

All the applications tend to confirm the view that formal methods applications will continue to be focused and lightweight in character, targeting specific subsystems or aspects. As we remarked in 2009, tools can now become a single point of failure for industry deployment, and we have seen cases in DEPLOY where tool scalability and performance, notably but not exclusively in automated proofs, have had a significant influence on the extent of deployment. This argues ever more strongly for significant improvements in the degree of automation possible in proofs.

A theme that comes out strongly in the DEPLOY studies is the appropriateness of the Event-B formalism and Rodin tools to specific development challenges. In our general review, some 90 % of reported projects indicated that the techniques used were appropriate to the tasks required. However, the automotive and space sector deployments in DEPLOY both indicate areas of mismatch, notably in the ability to describe closed-loop controllers and real time. To organisations that are new to formal methods, the selection of an appropriate formalism is a risky business. There appears to be a need for informal and impartial guidance on this. It is just conceivable that research teams may have too rosy a view of the breadth of applicability of the formalisms that they propound!

Many of the DEPLOY studies emphasised the value of formal techniques in improving the quality of requirements—a benefit already well established in the literature (e.g., [12]). The link from requirements to formal models is a recurrent theme. In SSF, the low level of abstraction at which some requirements were expressed (as Ada code) was seen as limiting the value of formalisation in Event-B. In other cases, such as the automotive (Bosch) and transportation (AeS) studies, the need to bridge a gap in abstraction and precision between informally expressed requirements and

the formal representation of a model warranted the development of assistive methods and tools. Several studies (including Bosch, SSF, AeS and CSWT) identified the need for assisted traceability between requirements and model.

In its scale and ambition, the DEPLOY project has been unique in the history of software engineering. Its experimental deployments represent a valuable contribution to the growing body of evidence on the application of formal methods to realistic problems on a realistic scale. They show that formal techniques, applied with care to appropriate problems and with respect for the prevailing development processes and culture in an organisation, can prove highly beneficial. Equally importantly, these studies give valuable insights into the priorities for the development of this important emerging technology in the future.

Acknowledgements We thank all the contributors to our survey. The following indicated their willingness to have their names listed: Thomas Arts, Janet Barnes, Nick Battle, Philippe Baufreton, Dines Bjørner, Nikolaj Bjørner, Michael Butler, Egon Börger, Gert Caspersen, Mikhail Eir, Lars-Henrik Eriksson, Alessio Ferrari, Dave Greve, Wolfgang Grieskamp, Anthony Hall, Dave Hardin, Anne Haxthausen, Alexander Kamkin, Moonzoo Kim, Joseph Kiniry, Rafael Marques, Aad Mathijssen, Steven Miller, Ian Oliver, Jan Peleska, Alexander Petrenko, Ray Richards, Andreas Prinz, Peter Päppenghaus, Aryldo G. Russo Jr., Thomas Santen, Anna Slobodova, Karl Stroetmann, Nicholas Tudor, Yaroslav Usenko, Eric Verhulst, Michael Whalen, Matt Wilding, Kirsten Winter and Wolf Zimmermann.

References

1. Austin, S., Parkin, G.: Formal methods: A survey. Technical report, National Physical Laboratory, Teddington, Middlesex, UK (March 1993)
2. Bicarregui, J., Fitzgerald, J., Larsen, P.G., Woodcock, J.: Industrial practice in formal methods: A review. In: Cavalcanti, A., Dams, D. (eds.) FM 2009: Formal Methods. Lecture Notes in Computer Science, vol. 5850, pp. 810–813. Springer, Berlin (2009)
3. Bloomfield, R., Craigen, D.: Formal methods diffusion: Past lessons and future prospects. Technical report D/167/6101. Adelard, Coborn House, 3 Coborn Road, London E3 2DA, UK (December 1999)
4. Brock, B., Kaufmann, M., Moore, J.: ACL2 theorems about commercial microprocessors. In: Srivas, M., Camilleri, A. (eds.) Proceedings of Formal Methods in Computer-Aided Design, FMCAD'96, pp. 275–293. Springer, Berlin (1996)
5. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NuSMV: A new symbolic model checker. Int. J. Softw. Tools Technol. Transf. **2**(4), 410–425 (2000)
6. Clarke, E.M., Wing, J.M.: Formal methods: State of the art and future directions. ACM Comput. Surv. **28**(4), 626–643 (1996)
7. Craigen, D., Gerhart, S., Ralston, T.: An International Survey of Industrial Applications of Formal Methods, volume 1, Purpose, Approach, Analysis and Conclusions. U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Systems Laboratory, Gaithersburg, MD (Mar. 1993)
8. Craigen, D., Gerhart, S., Ralston, T.: Formal methods reality check: Industrial usage. In: Woodcock, J.C.P., Larsen, P.G. (eds.) FME'93: Industrial-Strength Formal Methods, April 1993. Lecture Notes in Computer Science, vol. 670, pp. 250–267. Springer, Berlin (1993)
9. Glass, R.L.: Formal methods are a surrogate for a more serious software concern. IEEE Comput. **29**(4), 19 (1996)
10. Hinchey, M.G., Bowen, J.P.: To formalize or not to formalize? IEEE Comput. **29**(4), 18–19 (1996)

11. Jackson, D., Wing, J.: Lightweight formal methods. IEEE Comput. **29**(4), 22–23 (1996)
12. Larsen, P.G., Fitzgerald, J., Brookes, T.: Applying formal specification in industry. IEEE Softw. **13**(3), 48–56 (1996)
13. Miller, S., Greve, D., Srivas, M.: Formal verification of the AAMP5 and the AAMP-FV microcode. In: Third AMAST Workshop on Real-Time Systems, Salt Lake City, Utah, March 6–8, 1996
14. Miller, S.P., Anderson, E.A., Wagner, L.G., Whalen, M.W., Heimdahl, M.P.E.: Formal verification of flight critical software. In: AIAA Guidance, Navigation and Control Conference and Exhibit, San Francisco, August 2005. AIAA, Washington (2005)
15. Miller, S.P.: The industrial use of formal methods: Was Darwin right? In: 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques, pp. 74–82. IEEE, Boca Raton (1998)
16. Miller, S.P., Whalen, M.W., Cofer, D.D.: Software model checking takes off. Commun. ACM **53**, 58–64 (2010)
17. Owre, S., Rushby, J.M., Shankar, N.: PVS: A prototype verification system. In: Kapur, D. (ed.) 11th International Conference on Automated Deduction (CADE). Lecture Notes in Artificial Intelligence, vol. 607, pp. 748–752. Springer, Saratoga (1992)
18. Srivas, M.K., Miller, S.P.: Formal verification of the AAMP5 microprocessor. In: Hinchey, M.G., Bowen, J.P. (eds.) Applications of Formal Methods. Series in Computer Science, pp. 125–180. Prentice Hall International, Englewood Cliffs (1995)
19. Woodcock, J., Larsen, P.G., Bicarregui, J., Fitzgerald, J.: Formal methods: Practice and experience. ACM Comput. Surv. **41**(4), 1–36 (2009)

Chapter 11
Introducing Formal Methods into Existing Industrial Practices

Martyn Thomas and Alexander Romanovsky

Abstract This chapter discusses various methodological issues faced when integrating formal methods into existing industrial practices. It describes the experience of the DEPLOY industrial partners and the DEPLOY Associates gained during the deployment of Event-B and the supporting toolset in their industrial settings.

11.1 Introduction

This chapter addresses various methodological issues faced when integrating formal methods into existing industrial practices, such as

- ensuring artefact reuse (in particular, how formal methods affect a company's ability to reuse specifications, designs and system components),
- dealing with changing requirements (in particular, the importance of establishing the system boundary, strong requirements, requirements analysis, the role of formal methods and the role of agile development),
- ensuring dependability (in particular, the effect of formal methods on dependability, and the role of proofs in assurance, including certification),
- supporting teamwork,
- achieving integration with other methods and tools.

It describes the experience of the DEPLOY industrial partners and the DEPLOY Associates gained during the deployment of Event-B and the supporting toolset in their industrial settings. The content of this chapter is closely related to the work described in Chaps. 3–9.

Making advances in methods and tools has been one of the core objectives of DEPLOY. This work has always been driven by deployment needs. Rather than report-

M. Thomas (✉)
Martyn Thomas Associates, London, UK
e-mail: martyn@thomas-associates.co.uk

A. Romanovsky
Newcastle University, Newcastle upon Tyne, UK
e-mail: alexander.romanovsky@ncl.ac.uk

A. Romanovsky, M. Thomas (eds.), *Industrial Deployment of System Engineering Methods*, DOI 10.1007/978-3-642-33170-1_11,
© Springer-Verlag Berlin Heidelberg 2013

ing the general advances in methods made in the project, this chapter summarises the industrial experience in the relevant areas.

This chapter is therefore the combined work of many authors. The editors have standardised the style and terminology as far as it was possible, but the opinions expressed remain those of the original authors, based on the work they have undertaken in their individual companies. As the reader will see, the different industrial contexts lead to some interesting differences of opinion.

This chapter has been written using input provided by the following deployment partners and DEPLOY Associates: Rainer Gmehlich, Sebastian Wieczorek, Dubravka Ilić, Timo Latvala, Kimmo Varpaaniemi, Steve Wright and Aryldo G. Russo. It has greatly benefited from comments by Cliff Jones.

11.2 Requirements and Formal Specifications

According to international regulations, enterprise software such as the SAP product range is not safety-critical, and creating formal specifications for such products is uncommon. Instead, these products are built according to informally described requirements or semi-formal modelling notations, mainly at the levels of business processes and entity/class diagrams. Enterprise software developers therefore have little experience in formally expressing other aspects of software behaviour or properties. The introduction of Formal Methods to the enterprise software sector might result in immense training costs if formal modelling skills were to be acquired by a large group of developers.

Instead of enforcing formal modelling, SAP based their concepts on transformations of established domain-specific modelling languages like Business Process Model and Notation (BPMN) [2]. While it is usually not possible to translate every construct of such languages, SAP have succeeded in creating customised transformations that generate a formal model suitable for further analysis. Their approach required them to enhance their existing modelling environment so that it supported the transformations and connectors to the Rodin platform. However, the fact that developers are able to utilise formal methods based on familiar modelling languages (therefore, without additional training) compensates for the effort involved in developing these enhancements. SAP found that there were additional benefits:

- By providing automatic transformations, they were able to reuse parts of existing models and thus benefit from previous investments of the company.
- The performance of tools for formal analysis often depends on modelling decisions (for example, the choice between deterministic or nondeterministic initialisation of variables). With customised transformations these design decisions can be optimised for each use case.
- SAP's domain-specific design artefacts, such as process models, are used to communicate with customers and partners. The transformation approach prevents inconsistencies and allows them to discuss the results of formal analysis more easily.

- Automatic transformation has the potential to minimise tool switching, as users are able to trigger formal analysis from within their commonly used modelling environment.

In contrast with the non-safety-critical enterprise software developed by SAP, AeS (Grupo AeS) develop safety-critical systems in the railway domain and elsewhere. They use formal methods in a number of different ways, for example, to discover gaps and mistakes in their natural language requirements. They model the specified system at an abstract level and find that this can reveal inconsistencies in the requirements written in the natural language. AeS have also experimented with mapping use cases to Event-B and they have built a tool, VeRaSiS, that translates railway use cases into Boolean equations and validates them using ProB; this work is described in Chap. 7. Their experience shows that there can be considerable benefits from introducing formal specification of safety-critical requirements.

The formal specification undertaken by the XMOS project differed from many formal method projects in that it was not directly targeted at requirement validation or implementation verification in order to gain certification or to provide evidence of a dependability target being met. Instead, it could be considered to have been an exercise in providing a clear definition of an existing interface (the Instruction Set Architecture of the XMOS microprocessor) to allow reliable integration with external systems (human programmers and tools).

The XMOS ISA application allowed clear boundaries to be defined, both in terms of scope (all instructions) and abstraction (in sufficient detail to permit code execution).

The formal development was done effectively as agile development, using automatic code generation to effect a fast construct-prove-test cycle. A test suite and pre-existing compiler provided rigorous test examples, and event-hit detection provided exact coverage analysis.

As might be expected, formal construction of the ISA specification revealed several issues in the published document. These generally fall into three classes:

- direct errors, where a detail was unambiguously incorrect, and a compiled executable would never execute correctly under the specified behaviour;
- ambiguities which could be open to misinterpretation and potentially lead to differences in behaviour between different interpretations;
- omissions of certain functional conditions—typically, esoteric error scenarios.

Despite these successes, ISA formal specification was not considered enough of an advance over the existing ISA models to justify being integrated into the existing XMOS design processes.

Space Systems Finland (SSF) joined the DEPLOY project to focus on one of the greatest challenges in space projects: coping with various requirements issues, such as

- handling changing requirements;
- tracing requirements to different abstraction levels of specifications;

- integrating requirements resulting from RAMS activities;
- validating that requirements have been implemented.

With this goal in mind, SSF traced the requirements from a real space project (namely, BepiColombo SIXS/MIXS OBSW) to Event-B models of those requirements. Their results included finding inconsistencies or ambiguities in some requirements. They say that these findings were not, however, a direct outcome of any modelling activity but rather an effect of closely inspecting the requirements.

The resulting comments for specific requirements did not make any actual impact on the BepiColombo SIXS/MIXS OBSW project, and SSF concluded that their project did not give them any additional evidence to support the use of the Event-B methodology for coping with requirements issues in software development projects.

Their later work explored the problem of integrating the requirements resulting from Failure Modes and Effect Analysis (FMEA). Åbo Akademi University proposed a preliminary methodology for deriving conditions for operational modes and mode transitions directly from FMEA and incorporating these conditions as fault tolerance mechanisms using Event-B. As a working example, an Event-B model was developed based on the natural language specification of a centralised Attitude and Orbit Control System (AOCS). One of the remaining challenges in this area is support for requirements traceability from FMEA to its formal specification.

The validation of requirements having actually been implemented has been implicitly addressed through Event-B model refinements, where the initial Event-B model was based on the system requirements specification. A formal model of the system under investigation initially models the system in a very abstract way, yet allows reasoning about its correctness. The required system properties can usually be expressed as model invariants, and proving model correctness requires that the specified properties of the system hold. This process also helps the developer to detect incomplete or missing requirements, which may be the case when the proof simply cannot be completed with the given requirements specification.

Bosch spent considerable time and effort investigating how they could generate the high-quality, complete and consistent requirements that are needed for formal specification. They used Problem Frames [4] to achieve this, and the quality of their requirements was improved significantly. They also used some aspects of the Hayes/Jackson/Jones approach [5] to reason about the system boundaries.

Bosch believe that there is an important problem of traceability between natural language requirements and a formal model, which in general is not solved by Event-B alone, though traceability was significantly improved by using Problem Frames with some extensions.

11.3 Getting the Level of Detail Right

At SAP, as elsewhere, software development has broadly adopted the model-driven approach, in which different aspects of software at various abstraction levels are represented as software models (either as informal documents or as rigorous mathematical models). Therefore, when SAP analyse their existing software systems, there is

a good chance that software models at all abstract levels are already available. Even though most of these models are described informally in natural languages, SAP find that they can still use them as a solid basis and transform them into formal models. SAP see the following major challenges when they deal with models:

- When designing or adopting a modelling language, it is necessary to make conscious decisions about which modelling details should be included, yet without sacrificing verifiability.
- It is necessary to express the interrelations between models at different abstract levels explicitly and to make sure that they are consistent with each other.
- It is possible that the same aspect of a software system is covered by different modelling notations serving different purposes. For instance, a business analyst may specify a business process; a requirement engineer may describe a use case diagram; a system architect may illustrate an integration scenario, and a tester may draw a test case for UI/integration testing. All these artefacts address the same aspect of the system: how various modules and functions are put together to serve a business purpose. SAP have found that they can maximise the reuse of existing models and automatically translate them to other modelling notations, which saves resources and modelling effort and also reduces the possibilities of inconsistency.

SAP's approach is to include as much information as they see necessary in each software model. Then, if too much information results in a reduced level of automatic verification and a significant increase in manual effort, they split the current abstract level and, instead of having one model for all details, construct a series of models to which they add information gradually. One example is their message choreography modelling. A choreography model consists of two layers: one layer specifies only permitted sequences of messages from a global point of view; the other layer then goes into details of how each communicating partner sends and receives messages. Such an approach has the advantage of modelling complexity being reduced and of verification becoming much easier. The consistency between two layers can be guaranteed by specifying and proving refinement relations, using the refinement paradigm of the Event-B/Rodin platform. They translate all models into Event-B machines, specify their refinement relations as gluing invariants and prove the resulting proof obligations.

Bosch encountered difficulties in modelling realistic systems because their large Event-B models needed greater support for decomposition. The available decomposition mechanisms in Event-B were not considered powerful enough for their applications, despite Bosch making considerable efforts to find an approach that would scale to the necessary size.

11.4 Introducing Verification

There are two major formal verification approaches: theorem proving and model checking. Both approaches can be at least partially automated. There are many publicly available automated theorem provers.

SAP found that the two provers currently available for the Rodin platform are powerful enough to discharge a large portion of their proof obligations fully automatically, but with two major caveats:

- the number of automatically discharged proof obligations depends on the specific settings of proof tactics that a user has to configure; inexperienced users are likely to choose either the default setting or a less effective proof tactic that diminishes the power of the prover;
- when there are proof obligations that an automated prover is unable to discharge, a very skilled formal method expert may be needed to get them manually discharged, and this usually takes a long time.

Model checking is fully automated by an algorithmic exploration of the whole state space of a software model. However, such a state space is often enormous or even infinite. As a result, the resource-intensive exploration may sometimes exhaust computer memory or user patience. For this reason, model checking is mainly used for discovering errors rather than for gaining full confidence in correctness, especially for large and complex software models. Moreover, manual intervention is still required for setting the boundaries for the portions of huge state spaces that need to be explored if meaningful results are to be obtained. The main difficulty here is that it often takes a few trial-and-error rounds to discover the optimal bounds that reduce the state space to a portion just large enough to either track down bugs or yield the required degree of confidence.

Regarding the choice of the right proof tactics, SAP significantly benefited from recommendations and training offered by their academic partners. SAP were also able to take advantage of academic research advances such as the Relevance Filter, which heuristically selects potentially useful proof hypotheses. SAP observed a significant increase in the number of automatically discharged proof obligations after using the Relevance Filter.

As for the problem of residual interactive proofs, SAP discovered that such proofs are difficult in that they needed many intermediate lemmas that were closely related to the specific structure and characteristics of the model that they were proving. Such lemmas are not obvious at all, but once they have been generated, the difficult proof often becomes much easier. Since the lemmas are closely related to the model under proof and often describe information such as control flow or data dependencies, SAP hope that they can automatically discover them using static analysis. They have made some progress on this front: for instance, they automatically discovered dependencies among message flows and control flows for message choreography models and used them as lemmas to prove consistency and the absence of inconsumable messages. In many cases, all the proof obligations were automatically discharged following the introduction of those lemmas. However, there are still obstacles in this approach. Firstly, it is nearly impossible to make sure that all useful lemmas have been discovered or that all discovered lemmas are useful. Secondly, even when all the required lemmas have been made available, it is still difficult to use them in an automatic proof. The prover needs to know which ones are useful in the current proof step and how to use them. These are open issues.

SSF needed to be able to verify that their designs were free of all possible deadlocks, yet deadlock freedom is currently beyond the scope of the Event-B proof methodology for any nontrivial Event-B machine. They believe that one way ahead could be to use the animator and model checker ProB to check deadlock freedom or a corresponding invariant property without any additional modelling. The animation and model checking facilities in ProB are based on enumeration of reachable states and on simple evaluation of guards and actions. For each constant or abstract set in the Event-B contexts directly or indirectly seen by the Event-B machine of interest, ProB replaces the constant or the set with an enumerative expression that is compatible with all the axioms expressed in those contexts and with all ProB configuration axioms. However, in SSF's first BepiColombo-specific pilot, numerous attempts to use ProB failed because it was impossible to instantiate the required axioms and sets within a reasonable time.

Apart from that, SSF explored the possibility of expressing and verifying some temporal properties in their second pilot project (the centralised AOCS). While these properties were easily expressible as temporal properties for ProB, they could not in practice be expressed as invariants for the Event-B provers. ProB was not able to verify those properties, however, because any execution cycle of the AOCS can make several nondeterministic choices and none of the model-checking methods in ProB was powerful enough to cope with such nondeterminism.

SSF also explored modelling and verifying real-time requirements. Using the data processing of BepiColombo SIXS/MIXS OBSW as a working example, SSF investigated augmenting Event-B modelling with verification of real-time properties in the model-checking tool Uppaal [1]. SSF extracted a process-based view from an Event-B model and translated it into a timed automaton readable by Uppaal [3]. Event-B modelling with Rodin facilitates reasoning about the functional correctness of the system under construction. The process-based model explicitly defines processes and their synchronisation. Finally, a timed automata system model enables verification of liveness and the desired real-time properties on Uppaal. SSF's research has so far managed to demonstrate that the approach makes sense (at least in their specific example), but more effort will be needed before it can be recommended for actual industrial use.

Bosch also needed to model the sequence of events (which is to some extent a very abstract model of time) and they also found it possible, though cumbersome, in Event B.

11.5 Interpreting Verification Results

In SAP's experience, developers normally lack the skills and strong expertise in formal methods that are required to interpret verification results. When validating a property using a theorem prover, either the proof can be derived (automatically or manually with automated assistance), or the developer is left with a partially constructed proof and a set of open subgoals to prove. The latter case does not

necessarily indicate that the property is invalid, because it is very likely that a proof can be developed. Even if the property is truly invalid, the stalled proof does not provide much guidance to developers about why and how the property is violated.

In contrast, model checking provides a counterexample that illustrates where the property does not hold.

Even when a formal method confirms that a property is valid, SAP say that there are still certain pitfalls one has to be careful to avoid:

- the software model may be ill-constructed, so that there are self-contradictory hypotheses about the system. Anything can be deduced from an inconsistent set of hypotheses;
- another danger lies in the incorrect specification of the refinement relation between models at different abstract layers. If the specified refinement does not fully reflect how two models are related to each other, then despite the proof that the refinement is preserved, some properties of the more abstract model may not be satisfied by the more concrete one;
- it is also possible to forget to prove some properties that are essential for ensuring the correctness of the model. For example, if we do not prove deadlock freedom for a model, it could be that the model is deadlocked and no "bad" event can ever get started. In this case, we may prove certain safety properties yet achieve nothing but false confidence;
- we may specify properties incorrectly in such a way that they do not correspond to what we really want to verify. In the case of bounded model checking, whatever has been proved is only valid within the given bounds (of the number of execution steps or of integer values). Beyond these bounds, nothing is guaranteed.

SAP's solution is to empower users of theorem provers with model checking and visualisation tools. When a proof obligation cannot be automatically discharged, the tools first help the user construct a manual proof by visualising (a) what a state would need to look like in the original model (not the intermediate model in a formal language) to satisfy the current proof hypotheses, even showing how to reach that state; and (b) how the decisions of a user involved in a proof step are directly reflected in the original model (which branch to take, which message to send and so on). A state satisfying a certain set of hypotheses is found by model checking, while bidirectional links between the original model and its formal translation must be maintained for visualisation. In cases where there seems to be no possibility of constructing a proof but SAP want to disprove this, they also use model checking, utilising the disprover plug-in of the ProB model checker.

11.6 Achieving Acceptance by Industrial Engineers

In SAP's view, despite all the advantages of applying formal methods, it is still difficult to integrate them with realistic software development processes in industry.

They see two major obstacles:

- average developers and designers do not possess the knowledge of logic and mathematics that formal methods require. Therefore, either the operation of formal methods must be fully automated or specialists must be brought in, which is very costly;
- the degree of automation is currently still very low, especially when formal methods are used on large software models. Formal verification is also resource-intensive and time-consuming. Moreover, software development is often subject to constant changes in requirements, design and implementation. Whenever a change occurs, it is likely that most of the expensive formal verifications must be redone on all the affected software models with a fairly low level of reuse of existing verification results.

SAP therefore decided on the following conditions to increase the chance of success in applying formal methods:

- the application of formal methods should be made optional; it should be possible to easily and cleanly switch off their integration into a software development process, to be brought back when seen as appropriate; developers should be encouraged but not forced to use formal methods;
- whenever possible, the application of formal methods should be fully automated and hidden from developers; developers can then still benefit from the power of formal methods, and yet not be intimidated; when formal verification is taking a long time, it should stay in the background and not delay any other development activities;
- when the results returned from a formal method need to be fed back to developers, they should be reformatted and presented in a friendly, intuitive, helpful way so that the developers can easily understand and use the feedback to improve their designs and implementations.

SAP describe their success in meeting these goals in Chap. 6.

AeS have found that they have not yet been able to change their development process completely to include formal methods. They do have some trained people who are able to understand and use Event-B and Rodin, with some restrictions, but they have found that it takes a lot of time to replace the traditional approach and establish a new one, particularly when products are already under development.

To overcome this problem, AeS introduced formal methods in stages, at those points in development where they thought the new methods would improve the quality and dependability of the product. They would like to have the time and resources to undertake parallel development using formal methods and their traditional methods, to convince everyone of the power of formal methods and to create a new methodology for the development process to replace the old one, but this is impractical without extra funding.

XMOS have found that the barriers to integrating formal methods into their existing tool chain are not technical. The main barrier was that the additional level of rigour offered beyond existing test-based methods was not seen to be needed enough to justify the acquisition of new skills and the supporting tools.

11.6.1 Reuse

In the context of model-based testing, SAP minimise the efforts of constructing new test models by reusing the existing contents in their model repository. There are existing business processes written in a BPMN-like language; SAP export them from the repository and translate them into the format of test models. SAP have found that such translation is usually quite straightforward and can be easily automated. Their approach has the additional benefit of linking designs directly to tests, so that they can be confident that all the necessary tests would be created to address concerns in the design phase.

AeS have not yet found a way to reuse parts of a specification other than by taking the whole specification and simply modifying it to meet the requirements of a new development.

In XMOS's work, the Instruction Set Architecture was deliberately designed to be expandable, with "holes" left in the instruction code space for future operations. Their Event-B model contains an appropriate event for these unused operations, reporting the "illegal opcode" exception that currently occurs. Thus, extensions to the ISA should be easily accommodated by populating these placeholder events, which would in turn directly expand the coverage of the test suite. The "vertical" partitioning of the model (described in Chap. 9) should also maximise the reuse of existing proofs.

SSF gained some experience with reuse, starting with the initial formal development of Telemetry/Telecommand software (TM/TC SW) used in the BepiColombo pilot project. They found that TM/TC handling can be described in a more or less standard way, and thus in similar projects involving it specifications can be reused.

In SSF's later work, various things were reused or done with reusability in mind. They investigated the feasibility of reuse by experimenting with the Rodin modularisation and decomposition plug-ins, which extend the Event-B language with reusable generic modelling patterns. (This allows modules to be exchanged and reused in different projects, with a relatively easy way of integrating a module in a new project context.)

They have started exploring a set of reusable patterns incorporating fault tolerance, but more extensive work is needed in order to investigate the practicality of a tool-supported approach that would allow the development of a specific family of systems with reusable proofs.

11.6.2 Dependability

AeS believe that the use of formal methods for developing their safety-critical systems does increase their dependability, although they currently have no strong evidence. They wonder whether the mistakes that were formerly introduced at the specification step (but that are now eliminated through the use of formal methods) might instead be introduced during detailed design and coding. AeS have found that

formalising a system is a powerful way to gain a better understanding of it, and they believe that this prevents some errors from being introduced.

11.6.3 Teamwork

Industrial development projects are typically undertaken by one or more teams of developers whose efforts need to be co-ordinated so that they can work on the system in parallel. In contrast, academic projects are more likely to be undertaken by one or two individuals, with the consequence that software tools that emerge from university research teams rarely have the powerful modularisation and version-control facilities that are needed in industry. This was an issue encountered quite early in DEPLOY, and the Rodin platform was enhanced to provide some support for teamwork.

SSF used a sequential approach in their pilot project, with one person working on the project for a while and then forwarding the developed Event-B model to another person to continue the construction. They found that fairly complex models can be built in this way with reasonable effort. Different modelling approaches were explored for control systems of realistic complexity in the space domain, all resulting in massive monolithic models, handled by one developer at a time.

With the modularisation extension of Event-B (developed by Newcastle and Åbo with SSF's needs in mind), team development became feasible, allowing models to be developed at the same time by different developers. Different modules, developed by team members, can be easily integrated through predefined interfaces.

The Rodin platform modularisation plug-in provides features for structuring Event-B developments into logical units of modelling called modules. A module comes with an interface that defines the conditions under which it may be incorporated into another development, which provides support for structuring large specifications. In addition to allowing models to be built separately, possibly by different developers, the approach also modularises the proof effort and helps proof reuse, since it is quite common to go back in the refinement chain and partially redo abstract models. Normally, this would invalidate most proofs in the dependent components, but modular structuring helps localise the effect of such changes.

SSF carried out a modularisation plug-in experiment on BepiColombo SIXS/MIXS OBSW requirements, focusing on a few of the requirements that had been considered in their non-modular experiments. This demonstrated some of the benefits of modular development, but also showed that the total proof effort remained high even though it was evidently lower than in some earlier experiments.

Bosch recognised immediately that teamwork and teamwork support are important if formal methods and their supporting tools are to scale up to handle big projects. They believe that Event-B does not have enough support for modularisation and that, without what they would consider to be a proper module concept, the reuse of model parts is impossible. Bosch have tried different approaches to solving this problem, but have concluded that none of the existing concepts or plug-ins is sufficient for their purposes.

References

1. Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P., Yi, W.: Uppaal—a tool suite for automatic verification of real-time systems. In: Proceedings of the DIMACS/SYCON Workshop on Hybrid Systems III: Verification and Control, pp. 232–243. Springer, New York (1996)
2. Business Process Model and Notation (BPMN) Specification. OMG (2006)
3. Iliasov, A., Romanovsky, A., Laibinis, L., Troubitsyna, E., Latvala, T.: Augmenting Event-B modelling with real-time verification. In: Proc. of Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches, held in conjunction with ICSE 2012. 2 June 2012, Zurich, Switzerland
4. Jackson, M.: Problem Frames: Analyzing and structuring software development problems. Addison-Wesley Longman Publishing, Boston (2001)
5. Jones, C.: From Problem Frames to HJJ (and its known unknowns). Newcastle University, UK, No. CS-TR-1163 (August 2009)

Chapter 12
Tooling

Michael Butler, Laurent Voisin, and Thomas Muller

Abstract Together with many Rodin plug-ins, the Rodin platform supports the application of refinement-based development using Event-B and linked methods. This chapter outlines the management of the development and evolution of these tools during the lifetime of the DEPLOY project in response to deployment needs and methodological developments. The planning and maintenance process is described and a range of specific tool features developed to meet specific needs are outlined.

12.1 Introduction

The Rodin platform provides a range of features to support refinement-based development using Event-B. A high degree of extensibility was designed into the Rodin platform from the beginning in order to support the evolution of the Event-B method itself and of its links with other methods. There are two main ways in which extensibility is achieved in the platform. Firstly, Rodin is based on the highly extensible Eclipse platform. Secondly, the Rodin platform itself defines extensibility points to be used by Rodin plug-ins. Our experience from the DEPLOY project is that this extensibility has provided a major gain in distributed collaborative tool development. The modular architecture of Rodin allows for strong separation of concerns and functionalities, allowing many features to be developed independently of each other and making it easy to distribute responsibility for different features to different development sites.

M. Butler (✉)
Electronics and Computer Science, University of Southampton, Highfield, Southampton, UK
e-mail: mjb@ecs.soton.ac.uk

L. Voisin · T. Muller
Systerel, 1090 rue Descartes, Bt. A, 13857 Aix-en-Provence, France

L. Voisin
e-mail: laurent.voisin@systerel.fr

T. Muller
e-mail: thomas.muller@systerel.fr

A. Romanovsky, M. Thomas (eds.), *Industrial Deployment of System Engineering Methods*, DOI 10.1007/978-3-642-33170-1_12,
© Springer-Verlag Berlin Heidelberg 2013

With deployment of the platform in an industrial context being the main aim of the tool development work in DEPLOY, it was essential for the tool developers to provide mechanisms for collecting user feedback, particularly from our industrial deployment partners. In DEPLOY, we achieved this by using three means of communication: direct feedback reports to the relevant developers (contact addresses were provided for each major feature of the platform), mailing lists dedicated to the discussion of platform issues and web-based trackers for feature requests and bug reports. This chapter outlines the planning and maintenance process for Rodin tools in the DEPLOY project. We also outline a range of specific tool features developed to meet specific needs. These are grouped into tools for editing models, tools related to mathematical theory and proof, tools supporting automated analysis, tools supporting composition and reuse of models and tools for linking Event-B with other notations.

12.2 Planning and Maintenance

12.2.1 Strategic Planning

While the original proposal of the DEPLOY project included an outline of the main expected tool developments, once the project got under way, we needed to develop more detailed plans, including time frames and allocation of responsibilities for various features. The planning of the tool development was mainly led by a dedicated workpackage of DEPLOY, with input from the workpackages on industrial deployment and on methods. Plans were updated during workpackage meetings that were held typically every six months. During the second year of the DEPLOY project a refocusing exercise was held involving all partners, which led to some new feature requests on tools. These requests were prioritised and incorporated into the planning, including the addition of two major new tasks on code generation and model-based testing. Technical specifications of major features were developed and made available on the public Event-B wiki so that the initial feedback could be incorporated into the development.

Biweekly teleconferences were held between the tool development sites, where progress was reported, and issues coming from tool developers and from the other workpackages were discussed and resolved. Minutes of these teleconferences were taken and distributed to project members to ensure visibility of the planning and progress.

12.2.2 Platform Stability and Performance

Maintaining the stability and speed of the platform while continually increasing its capabilities has the potential to become unmanageable. One lesson learned in

the project is that focusing on feature implementation to satisfy user needs cannot be done without tightly linking it with performance. Several major stability and performance issues faced during the DEPLOY project were located in the core code of the Rodin platform and its user interface, causing crashes, loss of data, corruption in models, freezing and so on.

It was essential to tackle such issues in order to retain usability of the tools. An effective method of doing this was to conduct a two-pass investigation composed of code review and profiling, the first pass being a preliminary investigation and the second a deeper one. Each phase was then followed by some refactoring of the code. Moreover, solving the performance issues sometimes eliminated induced bugs as well, which meant that the scalability improvement tasks also contributed to the maintenance goals. The profiling strategy allowed the developers to achieve a better localisation of the performance loss in both the user interface and the core code, while the code reviews helped them understand the intrinsic misuses or drawbacks of particular components and architectures.

This process was used iteratively to ensure that performance was maintained while increasing the capabilities of the platform.

12.2.3 Processing Tool Requests

Within the DEPLOY project, there were three ways for users to send requests to the developers:

- direct request to developers,
- a dedicated mailing list,
- a ticketing system tracking bugs and feature requests.

The requests were then prioritised and responsibility assigned with the aim of improving the toolset while following the initial workplan and addressing the feature requests.

During the final year of the DEPLOY project, this process was reviewed to give preference to some high-priority tasks and issues that were regarded as being essential to resolve by the end of DEPLOY. Preference was given to corrective maintenance and further improvement of usability. It was agreed to focus on addressing specific bugs and issues reported by the DEPLOY partners. Thus, no new feature or plug-in development was initiated after that time. The tasks to be performed by the tool developers were then listed, scheduled and prioritised. This list was regularly updated during the biweekly management meetings to rapidly capture and integrate some minor changes which were required by the DEPLOY partners to enhance the usability of the platform.

12.3 Model Editing

12.3.1 Basic Modelling Support

During the whole project lifetime, the developers of the Rodin platform have continually tried to meet user needs when designing its interface and its integrated features, to provide the user with a seamless experience. Some basic integrated features such as model navigation, editing and refactoring needed to be dealt with. These characteristics had to be taken into account when defining the initial architecture of the tool even if the tool features themselves were susceptible to change.

Indeed, after some experience of use, the notion of "ease of use" often turned out to be different from what was imagined during functionality design. The toolset refactoring and enhancements are guided more by the application of the method that it supports than by purely technical considerations.

Generally, when building a toolset dedicated to modelling, three essential feature aspects are important for ease of use:

- model editing: providing a seamless way to edit models, including refactoring support, undo/redo and other editing actions, and even auto-completion;
- model structuring: giving visual feedback on the model structure with dedicated outline views or diagram views;
- model metrics: dedicated tools to provide metrics and to ensure traceability between manipulated elements, among other things.

The experience from extensive tool use by the DEPLOY partners shows that users are influenced by their habits when choosing new technologies and tools. Editing, a basic feature in such a platform, is particularly sensitive to this kind of habits. It is obviously a good choice to take them into account, adapting the editors accordingly in order to increase the acceptance of the tools. The multiple editors that have been developed during the project originated from this consideration. Moreover, basic features such as copy/paste, undo/redo and auto-completion affect the success of particular editors.

Structuring in the Rodin platform is provided by dedicated views. For example, the Event-B Explorer gives an integrated view of the various elements and proofs of a given project. To give another example, the structure of a whole project can be viewed using the Project Diagram view. All these views are provided by either the core platform or by external plug-ins. Structuring helps the user understand and maintain models.

Metrics in the Rodin tool are provided by a set of features such as the statistic view of a project. Metrics give the modeller a measure of the complexity of their models, of how much verification has been performed and how much remains to be done, and of the degree of automation in verification.

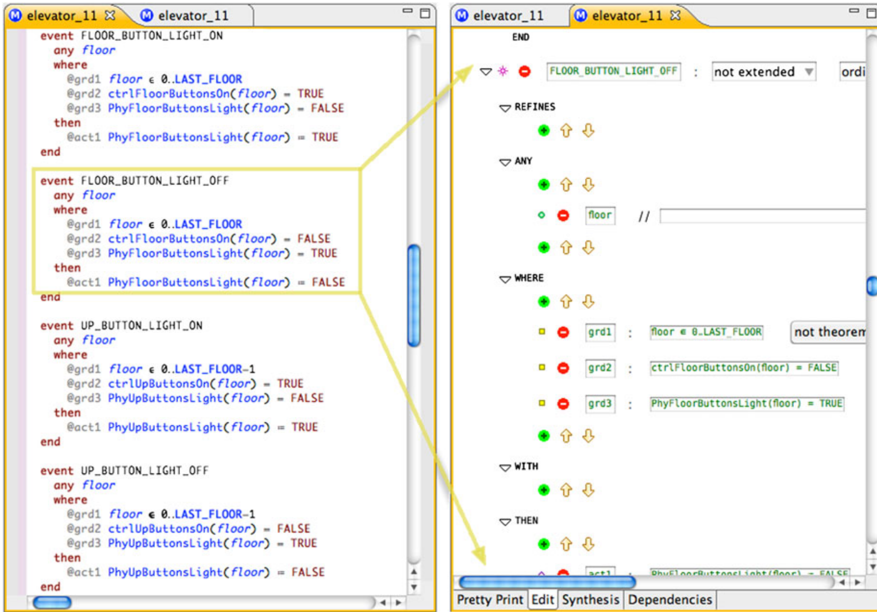


Fig. 12.1 Text editor and structure editor compared for an ETH Zürich elevator model

12.3.2 Camille

Initially, the Rodin platform for Event-B did not support a full textual representation of formal models. Model elements such as events, theorems and axioms are stored as elements in the Rodin database, and there is no classical text file for editing the models. Models are directly manipulated by a *structural form-based editor* that directly edits the underlying database and reflects its internal (tree) structure. This means the structure of a model is managed by adding and removing individual form elements, i.e., input fields, by using buttons within the editor (see the right-hand side of Fig. 12.1). The form shows the model structure and group elements of the same types, e.g., invariants, divided into sections. For example, when one wants to add a new invariant to a model, one has to press the *add* button within the invariant subsection, which will add a new set of form elements to the corresponding subsection (here comprising a label, an invariant and a comment field). The use of a database allows high extensibility and prevents a large number of syntactical and semantical errors. However, while this structural editor provides very useful guidance for novices, it became increasingly apparent that it was not able to cope with the needs of “power users”, especially the needs of the DEPLOY industrial partners, among which we will highlight the need for screen compactness, performance and flexibility.

The structural editor is very wasteful in *screen surface usage*. For example, according to our measurements, there is a difference of up to a factor of 30 in the

screen surface usage between the structural editor and a textual representation using the same font size. Figure 12.1 gives an illustration of this for a real-life model: while one event does not fully fit on the screen for the structural editor, we can see four complete events using the default layout of the textual editor. The structural editor is very slow at certain tasks (e.g., expanding all elements) and also has a limit on the number of elements that can be dealt with, mostly due to the use of resource-greedy graphical elements. A big drawback of the structural editor is the imposed *rigidity*. Indeed, many operations that come "for free" in a text editor are not available (and would require considerable effort to implement). Examples include arbitrary copy and paste, search and replace, movement of blocks of code, word count, detecting differences, balancing parentheses, and commenting on code blocks, to name just a few.

It quickly became apparent that a textual representation would have many uses, ranging from allowing simpler version control (e.g., using subversion or CVS) and sending models by email, to copying and pasting into other applications (e.g., presentation software). Some industrial users also require textual representation for internal documentation and auditing purposes. Textual representation is also less "fragile" with respect to changes to the database format. There was a strong demand from users for a text editor, which ultimately led to the implementation of Camille,[1] a semantic-aware text-editor for Rodin.

Camille features were mainly driven by power users who are experienced with IDE-based text editors. These include:

- Support of standard text editor features (cut/copy/paste, undo/redo and so on)
- Colour highlighting and error markers
- Auto-completion and templates

These features could be provided by leveraging the existing Eclipse infrastructure. The design was driven by the existing architecture and had to comply with its specific aspects. Firstly, an important design consideration was to be able to reuse the Rodin formula syntax and parser. The grammar for the Camille textual representation had to be carefully designed to be able to parse the structure of an Event-B model *independently* of the content of the predicates, expressions and actions. Another concern was that Camille should be able to co-exist with other editors. This means that all the user-relevant information has to be stored inside the Rodin database. A particularly tricky hurdle was the storing of layout information and comments. Indeed, comments have to be attached to specific elements in the Rodin database. Finally, another difficult issue was to keep as much proof information as possible when a model is changed, because one of the fundamental ideas of Rodin is incremental modelling. This means that the text editor has to update the Rodin model database in such a way that Rodin recognises that proofs can be reused. As such, the text editor needs to translate changes in the textual representation into "minimal" changes to the Rodin database.

[1]Named after Camille Claudel (1864–1943), a French sculptor and graphic artist. She also was Rodin's source of inspiration, his model, confidante and lover.

Overall, the most challenging technical part of Camille was synchronisation with the Rodin database, particularly in the presence of other tools concurrently manipulating the same model. To achieve this, we took advantage of a new abstraction of the Rodin database as an Eclipse Modelling Framework (EMF) [5] data model. It is possible to switch back and forth between Camille and the original editor. If no text representation of the Event-B model exists upon opening Camille, then Camille will generate the initial version of the text. Users expect the text layout to be preserved in every text editor. This required some effort to implement since we do not save a text file but only a model in the Rodin database. To preserve the original layout, we decided to persist the complete input text and its rendering timestamp along with the model. We use EMF *annotations* to achieve this. Annotations allow attaching arbitrary objects to every model element and are automatically persisted. Thus, whenever Camille loads a model that has already been edited by the text editor, it shows the preserved text representation. It uses the timestamp to detect conflicts with changes that were made by other editors or tools. In those cases, the layout is discarded, and the text representation regenerated from the model.

Storing the text representation also allows saving the editor's content, even if it is not parsable, i.e., when no model can be derived due to faulty syntax or other problems. Use cases in which the user wants to save an intermediate, broken state of the model are likely in textual editing. Last, we also annotate model elements with text range information that stores their location in the formatted text. This is useful for providing visual feedback directly in Camille. For instance, the Rodin core identifies non-existent variables as a problem in the model. In order to forward that information to Camille (by underlining that variable in red), we must know the position of the offending element in the text.

Camille has proven to be very useful and has received very positive feedback from industrial and academic users. In particular, Bosch found the structural editor inadequate for realising their pilot project. In their public Deploy deliverable D19 [26] they state that "we found the text editor very helpful and prefer to use this representation". There is, however, plenty of room for improvement. In particular, Camille currently has no mechanism for supporting plug-ins that extend the syntax of Event-B. We are currently evaluating alternative architectures that would provide this.

12.3.3 Rodin Editor

The original editor of the Rodin platform is a form-based one that supports the structure of Event-B models while allowing to edit its semantic elements. This editor suffers from a number of limitations. Its representation of models does not use the screen space efficiently, and navigation of large models can be uncomfortable. While a text editor (such as Camille) addresses these limitations very well, there are other desirable features that are not easily supported by a text editor. Firstly, it is important to be able to retain the ability of a plug-in to extend the syntactic structure

of Event-B models. In addition, in deploying the tool, other editor features were identified as being useful. One was the ability to display elements inherited from an abstract model by a refined model in an embedded way; this is useful when an event is extending an abstract event. Another feature identified is the ability to lock elements of a model to prevent them from being edited; this feature is useful for model elements that are generated from another source (e.g., a UML-B diagram that gives rise to generated invariants, guards and actions). The design of the original editor made it difficult to support these features, creating a need for a new editor based on an intermediary representation of the models. The new Rodin Editor was designed to go some way towards giving the look and feel of a text editor, while retaining the extensibility of the original editor as well as providing the ability to display elements inherited from an abstract model and to lock model-generated elements.

12.3.4 Teamwork

For industrial-scale projects, models quickly reach a size where a team of modellers is required to work in parallel in order to meet timescales. Each modeller needs to be able to work on an aspect of the model in isolation from other modellers' changes until an appropriate point for merging their work into the main development is reached. Team-working methods like these are supported for program development by version control repository systems such as SVN. In the DEPLOY project, Bosch in particular were keen to explore team-based development of Event-B models.

The Rodin database does not support versioning and branching of development streams. While the files of a Rodin project could be stored into a version control system such as SVN, they are in an XML format which is not readily readable except by loading via the Rodin toolset. Once a Rodin resource has been moved out of the context of a Rodin project or the Rodin toolset, it is not longer readable (except as XML text strings). Therefore although a copy of a model can be stored in SVN, it is then not easy to identify its differences from a copy in a workspace. For effective teamwork it is essential that differences between copies in different development branches can be easily reviewed and merged.

The 'team working' plug-in provides tool support to enable Rodin resources to be saved in a version control repository (e.g., SVN) in a format which enables the versions of models to be compared in a differencing editor. To do this, when a project is designated for sharing, the team working plug-in produces a synchronised copy of each Rodin resource in an EMF-based format that is independent of the copy in the Rodin database. Thereafter, whenever the resource is changed in the Rodin database, the copy is automatically updated to match it and vice versa. This copy of the Rodin resource can be shared using a version control system, so that modifications are committed or updated from the repository in the usual way in which such repositories are used for code development. Differences between the repository version and the local workspace one can be reviewed in a structural editor/viewer,

allowing individual changes in the former to be merged into the latter. A limitation of the current version of the differencing editor is that it does not properly support a three-way comparison. This comparison is needed in order to review changes when these have been made in both the local workspace version and the repository one since the former was last synchronised with the latter.

12.4 Theory and Proofs

12.4.1 Prover Enhancements

Rodin's Built-in Provers Enhancing the built-in provers was an important task carried out in DEPLOY. Naturally, all of the industrial partners in DEPLOY are keen for the proof to be as automatic as possible. This was done in various ways: addition of tactics and rewriters, addition of plug-ins to link tactics with external provers and so on. However, the components manipulating the proofs need to be "trusted". The architecture of Rodin isolated such components in a set called the "trusted base". Firstly, the reasoning rules to be implemented were reviewed and proven externally using other proving tools, internal or external to Rodin. Secondly, as their implementation was responsible for the correctness of the entire Event-B development, it needed to be made concise and clear enough to be trusted by using code review (maximum 30 lines of code).

Relevance Filtering Proof obligations of Event-B models typically contain numerous irrelevant hypotheses, i.e., hypotheses that do not contribute to a proof. Rodin's automated theorem provers (PP, newPP and sometimes also ML) perform poorly in the presence of irrelevant hypotheses: even a modest number of them can increase the time required to find a proof from seconds to days (or even longer). In some models, typically used for education, it makes sense to pass all available hypotheses to Rodin's automated provers; but in models of industrial scale Rodin's automated provers become useless unless (most) irrelevant hypotheses are removed from the input. A problem arises beyond Event-B and Rodin, indicated by the fact that the CASC competition[2] for automated theorem provers in first-order logic has a dedicated division for problems with irrelevant hypotheses.

At the beginning of the DEPLOY project, Rodin provided simple heuristics to determine which hypotheses are relevant. But these heuristics proved to be ineffective on models from industrial partners. Users therefore had to manually indicate on a per proof obligation basis which hypotheses were relevant. This is a tedious and error-prone process that is unacceptable in industrial models which have at least hundreds of hypotheses and proof obligations.

As a solution, ETH Zurich has developed a relevance filter plug-in. The plug-in implements sophisticated heuristics for detecting irrelevant hypotheses [15, 29,

[2] http://www.cs.miami.edu/~tptp/CASC

36, 40]. It provides a proof tactic, the *meta-prover*, which in a first step removes irrelevant hypotheses from the proof obligation at hand and then inputs the reduced sequent to one or several of Rodin's automated provers (PP, newPP, ML). The power of the meta-prover stems from the fact that it attempts to prove a single proof obligation by using several filter heuristics and several provers. It was shown that the meta-prover significantly increases the rate of automatically discharged proof obligations on models from various domains [35]; this includes models from industrial partners that have not been used for fine-tuning the underlying heuristics.

Evaluation by ETH Zurich shows that the meta-prover is useful for a variety of domains. Nevertheless, we do not claim that the meta-prover (in its default configuration) is useful for every model. Some proof obligations cannot be automatically proved even with a perfect relevance filter; this concerns about 7 % of the proof obligations in the benchmarks analysed [35], but the numbers vary from model to model. Moreover, we would not be too surprised if the parameters of the meta-prover needed to be readjusted for some future application. Readjusting parameters is essentially searching within a set of possible configurations and therefore requires some computing power but no ingenious insights.

In summary, the relevance filter plug-in often increases prover performance without demanding user interaction.

12.4.2 Mathematical Extensions

Although Rodin supports a rich built-in mathematical language of set theory, there is always a need for richer mathematical tools such as additional theories and proof rules arising from specific applications. For example, Siemens required the ability to specify and reason about some graph-theoretic operators in a generic way. For the code generation work (Sect. 12.7.3), it is desirable to express theories of implementation-level data type operations, such as bounded integer and array operations.

The theory plug-in provides a mechanism to enrich the mathematical language of Event-B, and enables users to contribute sound proof rules. It aims to address the following two issues:

1. The old proof infrastructure of Rodin had many hardwired proof rules. In order to add a useful rule, it was necessary to write Java code that contributes to a certain extension point in Rodin. Needless to say, this was cumbersome for users. A more complicated issue is the verification of the soundness of contributed rules.
2. The mathematical language of Event-B is implemented as a fixed grammar with an abstract syntax tree. In order to add certain useful operators (e.g., sequence operators), Rodin users tended to specify them axiomatically, using contexts. Two issues arise with this particular approach. Structures defined axiomatically in contexts are not reusable outside the scope in which they are defined. Secondly, defined structures are essentially monomorphic and can only be used with the carrier sets with which they were defined.

In order to address the aforementioned limitations of the Rodin toolset, the theory plug-in provides a construct called a 'theory', which is distinct from contexts and machines. The theory component enables Rodin users to

1. specify new datatypes, including enumerated datatypes (e.g., DIRECTION := NORTH, SOUTH, EAST, WEST) and inductive datatypes (e.g., lists and trees);
2. specify new operators that are polymorphic for the types for which they are defined;
3. specify polymorphic theorems that can serve two purposes: (1) verify the properties of defined operators and datatypes, and (2) be available for instantiation and incorporation in proofs;
4. specify rewrite rules, which are directed equations that can be used to simplify formulae in proofs;
5. specify inference rules which can be used to split, discharge or simplify proof obligation sequents.

Proof obligations are generated from definitions in theory components (e.g., to ensure soundness of theorems, and rewrite and inference rules). Datatypes, operators and proof rules can be used in models after the parent theory is deployed.

Certain decisions were taken as a response to some theoretical considerations. Starting from version 1.3 of the plug-in, mutually recursive datatype definitions were not supported, since this would require more theoretical groundwork. However, we anticipate that in the future such additions may be feasible. The usability issue involved making a particularly sensitive decision with regards to theory deployment. Two competing approaches emerged as a solution to this particular issue. Automatic discovery of theory extensions was deemed too complicated and confusing to the end user, and was also shown to be technically unusable because of the way that the static checkers in Rodin work. In the end, it was decided that the user should explicitly deploy a theory to make it available for modelling and proofs.

12.4.3 Isabelle in Rodin

At the beginning of the DEPLOY project, the user had three choices when valid proof obligations were not discharged automatically: (1) do manual proofs, (2) reorganise the model to make it "simpler" for Rodin's theorem provers, or (3) implement an extension of Rodin's theorem provers in Java. None of these options is fully satisfactory for models of industrial scale:

1. The number of undischarged proof obligations is typically very high, which makes manual proving an expensive task; in particular, users have reported that they had to enter very similar proofs again and again (for different proof obligations), which they found frustrating.
2. Reorganising the model helps in some situations, but not in others, and requires considerable experience with and deep understanding of Rodin's theorem provers.

3. Extending Rodin's theorem provers is quite effective in principle. However, in Java prover extensions need to be defined in a procedural style; the source code of prover extensions is therefore hard to read and write. Moreover, it is quite difficult to ensure that prover extensions are sound. If a model has been verified with an extended version of Rodin's theorem provers, it is questionable whether the model is indeed correct.

One solution to this problem is the integration of Isabelle/HOL [31] into Rodin. Isabelle/HOL is the instantiation of the generic theorem prover Isabelle [32] to higher-order logic [1, 2, 11]. The main advantage of Isabelle is that it provides a high degree of extensibility whilst ensuring soundness. In a nutshell, in Isabelle every proof has to be approved by its core, which has matured over several decades and is therefore most likely to be correct, and user-supplied prover extensions are sound by construction.

Isabelle comes with a vast collection of automated proof tactics that can be customised by the user in a declarative manner, i.e., by declaring new inference and rewrite rules. It provides linkups to several competition-winning automated theorem provers such as Z3 [3] and Vampire [33]. Last but not least, the default configuration of Isabelle's proof tactics has matured over years or decades and has been applied in numerous verification projects.

The Isabelle plug-in translates a proof obligation to Isabelle/HOL, and then invokes a custom Isabelle tactic and reports the result back to Rodin. The translation to HOL, unlike those to other theorem provers (such as PP, newPP, and ML), preserves provability: translations of provable proof obligations are provable and those of unprovable proof obligations are unprovable.[3] If the performance of the default configuration is unsatisfactory, users can inspect the translated proof obligations in Isabelle/HOL, test the behaviour of various proof tactics and declare new inference and rewrite rules to meet the desired performance goals.

The Isabelle plug-in was evaluated on the BepiColombo model [42] by Space Systems Finland. Isabelle discharged some of the proof obligations out of the box that could not be discharged automatically by Rodin. The automation rate was further increased by declaring new inference and rewrite rules within Isabelle. Adding rules to Isabelle had an important advantage over directly entering manual proofs: by declaring new rules, we did not only make the automated tactics of Isabelle prove the proof obligation under investigation, but also other proof obligations that we had not yet looked at. So the human effort on one proof obligation paid off on others. The details of this case study can be found in the PhD thesis of Schmalz [37].

The Isabelle plug-in currently has the following limitations:

- Proof search by Isabelle is slower than proof search by Rodin. There is still room for improvement. Even so, Isabelle often outperforms human proof engineers by several orders of magnitude.

---

[3] This claim rests on the assumption that the implementation of the translation is correct. We regard this a reasonable assumption, as the implementation of the translation is quite straightforward and concise.

- The full power of the Isabelle plug-in is obtained by tuning its proof tactics to the problem at hand. For this, one (but certainly not every) engineer needs to be familiar with Isabelle/HOL.
- The Isabelle plug-in does not interact with mathematical extensions (Sect. 12.4.2) in a useful way. A (limited) workaround is to unfold all definitions introduced by mathematical extensions before invoking Isabelle.
- If Isabelle is unable to prove a proof obligation, it outputs the subgoals that could not be proved. This output often makes it clear—to engineers familiar with Isabelle/HOL—why Isabelle was unable to complete the proof. Unfortunately, it is not clear how to translate this output into a format that can be understood by other Event-B users.

A way to customise Rodin's theorem provers for the user level, namely the theory plug-in, described in Sect. 12.4.2, was developed during the DEPLOY project. The theory plug-in avoids the overhead of integrating an external tool into Rodin and therefore has the potential of being faster and easier to use. Whether the Isabelle or the theory plug-in (or a combination of the two) is more suitable for a given task should be determined based on preliminary experiments. That said, Isabelle/HOL is an improvement over the current theory plug-in in the following ways:

- The theory plug-in is unable to express certain rules involving numerals (1, 2, 3, ...), enumerated sets ($\{a, b, c\}$), binders ($\forall$, $\{x \mid x > 0\}$), and Boolean connectives ($\wedge$, $\vee$, $\Rightarrow$). Isabelle/HOL does not have such limitations.
- Isabelle/HOL provides a default configuration of automated tactics that has proved itself over years.
- Isabelle's generic rewriter and classical reasoner are more powerful than the corresponding tactics of the theory plug-in.
- The theory plug-in has exhibited several unsoundness bugs. Unsoundness bugs in the Isabelle plug-in are unlikely (and have not been observed) because of the maturity of Isabelle and the intuitive nature of the translation.

What the Isabelle plug-in adds to Rodin is a theorem prover with a very good automation, and high degree of extensibility and soundness. To fully benefit from its power, familiarity with Isabelle/HOL is required.

12.5 Automated Analysis

12.5.1 Model Checking

While a prover can ensure that a formal model is consistent, it cannot prove that a model behaves in the way the modeller wants it to. ProB supports a user in understanding and analysing a model. ProB is an animator and model checker that was originally developed for the B method and is now extended to also support Event-B [25].

An animator finds concrete values for a specified model. It gives the user the ability to "play" with a model by showing effects that events have. It displays a current state and lists the events that are enabled in that state. The user can then select events to "execute" and see how the state is changed. Thus the animator gives the user early feedback and helps him/her better understand the model, avoiding the high costs of changing it later in the development process. An example of a model property that is usually hard to express in a formal way is whether an event is enabled in certain situations. ProB has a low entry barrier; a few mouse clicks are usually enough to apply it to a model. Another benefit of animation is that it can help narrow the gap between domain and formal method experts by making the model more concrete and understandable. This can be further improved by creating visualisations with BMotionStudio (Sect. 12.5.2), which relies on ProB and has been significantly improved during the DEPLOY project.

In the model-checking mode, ProB can be used to automatically search for flaws such as invariant violations or deadlock situations. For model checking, proofs done in Rodin are used to reduce the effort of validating invariants in explored states. Adding support for animating refinements was necessary because refinement plays a very important role in Event-B. Now a user can inspect the relation between refinements with ProB.

ProB was extensively used in all of the deployment cases studies in the DEPLOY project, and its performance was continually improved in the course of the project. The main motivation for the enhancement of its core was an industrial case study from Siemens, in which ProB had to automatically validate large data sets that describe a railway topology. In the end, checking whether the values fulfilled the assumptions took a few minutes, whereas previously one person needed a whole month to do this manually. Another case study from Bosch, in which ProB could search for deadlock situations, led to further improvements in the models. These developments are also beneficial for other users. Not only did ProB become faster, it also enabled features such as constraint-based deadlock checking and invariant, or assertion checking, which search for counterexamples for certain properties. This helps the user spot flaws in the model sooner than in a full-blown proof.

12.5.2 Model Animation

The communication between a developer and a domain expert (or manager) is very important for successful deployment of formal methods. On the one hand, to continue with development, it is crucial for the developer to get feedback from the domain expert. On the other hand, the domain expert needs to check whether his/her expectations are met. An animation tool such as ProB allows the presence of desired functionality to be validated, but requires knowledge about the mathematical notation. To avoid this problem, it is useful to create domain-specific visualisations. However, creating the code that defines the mapping between a state and its graphical representation is a rather time-consuming task. It can take several weeks to develop a custom visualisation.

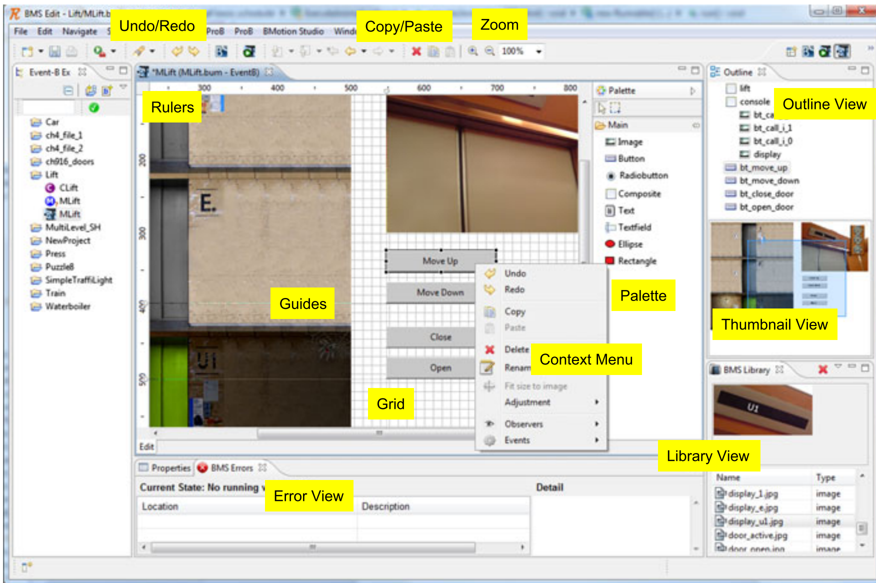


Fig. 12.2 BMotion studio editor

B Motion Studio is a visual editor which enables the developer of a formal model to easily set up a domain-specific visualisation to demonstrate to and discuss with domain experts. B Motion Studio comes with a graphical editor (see Fig. 12.2) that allows a visualisation to be created within the modelling environment. Also, it does not require the use of a different notation for gluing the state and its visualisation.

The main advantages of B Motion Studio are:

- The modeller uses the same notation throughout. B Motion Studio uses Event-B predicates and expressions as gluing code.
- An easy-to-use graphical editor allows visualisations to be created with a few mouse clicks (see Fig. 12.2).
- B Motion Studio comes with a number of default observers and controls that are sufficient for most visualisations.
- It can be extended for specific domains.

B Motion Studio uses two concepts that are based on the model-view-control design pattern: *Controls* and *Observers*. A control is a graphical representation of some aspects of the model. Typically we use labels, images or buttons to represent information. For instance, if we model a system whose characteristics include temperature and a threshold temperature that triggers cooling down, we could simply use two labels to display both values, or we could incorporate both pieces of information into a gauge display. It is also possible to define new controls for domain-specific visualisations. Observers define the control behaviour based on information about the model state, for instance, what image it displays or what position it has.

B Motion Studio is based on the ProB animator and is integrated into the Rodin platform. The user can perform a state change by using ProB, i.e., by double-clicking on an enabled event in the Event View or by using B Motion Studio, i.e., by clicking on a button which is wired to an event. As the state changes, the observers start to evaluate expressions or predicates using ProB. The results are used to change the visualisation.

B Motion Studio is being actively developed at the University of Düsseldorf. In particular, support for other editors based on GMF/GEF [12, 13] in conjunction with B Motion Studio is being actively researched and will be implemented relatively soon.

In summary, we hope that B Motion Studio provides a way to quickly generate domain-specific visualisations for a formal model, enabling domain experts and managers to understand and validate the model. We also believe that our tool will be of use in teaching formal methods, both in lectures and as a way to motivate students to write their own formal models.

12.5.3 Model-Based Testing

Model-based testing (MBT) for Event-B models was investigated in response to direct requirements of the deployment partners, in particular SAP. Such requirements are based on the fact that testing is currently the main software validation method used in industry. It is therefore a very important part of the software development cycle that consumes large proportions of project budgets, and any reliable methods that bring improvements to it are welcomed by practitioners. The possibility of MBT for Event-B relies on the ability to automatically generate tests from formal models, with the effect of reducing manual testing and increasing testing coverage. Thus, the Event-B framework is used not only to prove the consistency and correctness of the specifications but also to generate tests that can be used against existing implementations or during the implementation process using a test-driven approach.

Our MBT plug-in [4] takes an Event-B model together with its different levels of refinement, if available, as an input, and outputs a set of test cases (a test suite). The test cases consist of sequences of events together with appropriate test data that enable them. Commonly, MBT algorithms generate tests by traversing the state space of the given model, guided by coverage criteria. To address the challenge of what is usually a large state space, our solutions construct finite approximations based on machine learning algorithms. These finite state approximations provide the basis for test generation. Moreover, test suite reduction techniques are applied to reduce the size of the generated test suite. Last but not least, the approach is iterative, following the refinement methodology of Event-B.

One of the main difficulties in test generation for Event-B models is the large number of possibilities and data to be explored in order to find an appropriate test suite (cf. the state space explosion problem). Since application of explicit model checking as supported by ProB does not usually scale to large test data domains,

we explored the possibilities of (a) abstracting away data and using the built-in constraint solver of ProB (this is investigated by the University of Düsseldorf) and (b) using model learning (this is investigated by the University of Pitesti). Model learning can address the problem at hand incrementally and can take advantage of a human tester that provides input with the goal of influencing or improving the test generation. Human intervention comes at a price, requiring domain knowledge and some test generation expertise to obtain a better test suite in less time by guiding the search algorithm.

12.6 Composition and Reuse

12.6.1 Decomposition/Composition Plug-in

Decomposing an Event-B model is a syntactic process that distributes model elements (variables, invariants, events) among several submodels. Depending on the nature of the decomposition, these submodels may interact via either shared variables or shared events. This process is intended to be used so that several refinements of an abstract model may be performed, and that when the model is sufficiently detailed, it is decomposed into submodels. The submodels can then be further refined and decomposed. As submodels are refined, external events are used to represent the assumptions about the behaviour of the other submodels (the environment of a submodel) [38]. In addition to allowing the models to reflect architectural structure, decomposition makes it possible to keep models reasonably small: refinement tends to lead to an increase in model size, so decomposition can help prevent such excessive growth.

To support the decomposition process, a plug-in was developed to automate the construction of submodels from the source model [38]. The modeller provides names for the targeted submodels, chooses whether to use shared variable or shared event interaction between them and then decides how the variables or events should be distributed amongst them. The plug-in then uses this information to generate the submodels from the source model. Initially decomposition was defined by the modeller in a wizard, and decomposition decisions were not saved. But after doing this for a while, we realised that a “decomposition configuration file” was required to save and rerun decomposition whenever necessary. A composition file is also created automatically after a decomposition has been applied to allow the user to see how the events were split. The decomposition plug-in was applied to the BepiColombo system developed at Space Systems Finland [9]; this experiment demonstrated that it is feasible to decompose the model following the architectural principles of the BepiColombo system and to then refine submodels independently.

The composition plug-in was initially developed to study how to combine machines in a structural way while preserving their properties. That study led to the development of the decomposition plug-in, since both tackle the same situation from different perspectives: composition combines machines to generate a bigger

machine while decomposition splits a bigger machine into smaller ones. A current limitation is having proof obligations generated directly into the composition file. At the moment, a new machine is generated as a result of composition, which contains proof obligations that need to be discharged to ensure valid composition. In the future, proof obligations will be defined directly in the composition file.

Currently, the best approach to using decomposition is to tailor the model to a decomposition style that allows an easy split. When decomposition has been completed, submodels are usually easier to manage and proof obligations easier to discharge. The challenge is to apply decomposition: it is not always easy from a user's point of view to do so after several refinements. Often the model and its variables are so entangled that it is hard to separate them. We are going to address this issue by improving the automation of some steps in the tool as well as the user interface. The propagation of changes from the abstract, or non-decomposed, machine to its subcomponents is also desirable.

12.6.2 Design Pattern Management/Generic Instantiation

Formal methods are applicable to various domains for constructing models of complex systems. However, often they lack a systematic methodological approach, in particular in reusing existing models, to helping the development process. The objective of introducing design patterns within formal methods in general, and in Event-B in particular, is to overcome this limitation.

The idea behind design patterns in software engineering is to have a general and reusable solution to commonly occurring problems. In general, a design pattern is not necessarily a finished product, but rather a template of how to solve a problem, which can be used in many different situations. The idea is to have some predefined solutions, and to incorporate them into the development with some modification and/or instantiation. With the design pattern tool (which can also be referred to as the generic instantiation tool as it performs generic instantiation of design patterns) we provide a semi-automatic way of reusing developed models in Event-B. Moreover, the typical elements that we are able to reuse are not only the models themselves, but also (more importantly) their correctness in terms of proofs associated with the models.

In our notion of design patterns, a pattern is just a development in Event-B, including an abstract model called specification and a refinement [14]. In the course of development it might be possible to apply a pattern to the current development that was already developed before. The steps involved in using the tool are as follows:

Matching. The developer starts the design pattern plug-in and manually matches the specification of the pattern with the current development. In this linking of pattern and problem the developer has to define which variable in the current development corresponds to which variable in the pattern specification. Furthermore, the developer has to ensure the consistency of the variable matching by matching the events of the pattern specification with those in the development.

Renaming. Once the matching is done, the developer has the possibility of adapting the pattern refinement, such as renaming the variables and events or merging events of the pattern refinement with uninvolved events of the development. Renaming can become mandatory if there are name clashes, meaning that the pattern refinement includes variables or events having the same names as those of the existing elements in the development.

After the adaptation of the pattern refinement, the tool automatically generates a new Event-B machine as a refinement of the machine where the developer started the design pattern tool. The correctness of the generated machine as a refinement is guaranteed by the generation process of the tool.

Design patterns in Event-B are in fact ordinary Event-B models and are not restricted in size. As the pattern has to be manually matched to the development by the developer, its size affects usability. Furthermore, if the pattern is too specific, either it cannot be used in most situations or the developer is forced to tune his/her development in such a way that further application of the pattern is possible. A pattern-driven development could thus result in models including unnecessary elements, which would be avoided if no patterns were used.

12.6.3 Transformation Pattern Plug-in

In addition to the model instantiation approach outlined in the previous section, another approach to treating modelling patterns represents refinement patterns as transformations of a model. The transformation pattern plug-in supports writing (and executing) model transformation code in a simplified programming language without compilation. The plug-in addresses the need for scalable tool-supported reuse of already established modelling practices within domains or families of projects. Technically, the transformation patterns are scripts that are written in the EOL language [6], and are intended to introduce the desired behaviour into the models upon instantiation. However, the tool can also be used for general automation of modelling routines, e.g., model generation by developers and other tools. The tool support for transformation patterns includes a text editor, an interface for choosing a pattern for instantiation, and an interface for user input during execution, for example, for specifying Event-B elements required by the pattern. To develop patterns, the API provides facilities for accessing Event-B elements. During instantiation, a pattern script is executed against the models and the Event-B elements specified by user. More technical details can be found in [41].

Modellers should consider using transformation patterns when

- they can separate and explicitly formulate a specific aspect of their modelling;
- writing and executing a script is cost-effective in terms of effort and time compared to repetitive modelling.

For reusing proofs together with plain instantiation, see the design pattern and generic instantiation plug-in. The transformation pattern plug-in is not intended for

proof reuse and, based on the imperative programming paradigm, it is rich in terms of the available instantiation operations.

The transformation pattern tool was successfully used for the implementation of our work on FMEA patterns [28], and proved to be a pragmatic way of reusing the established modelling practices.

12.7 Linking Event-B with Other Notations

12.7.1 Requirements Management/ProR

The traceability between natural language requirements and formal models was one issue that the DEPLOY partners were struggling with. ProR [24] (Fig. 12.3) is an extensible requirements engineering platform which is part of the Eclipse Requirements Modeling Framework (RMF) [20, 34]. An integration plug-in for Rodin exists for creating traceability between Event-B models and natural language requirements.

A formal model acts as a specification for a system to be built. Since a system works correctly if it satisfies its requirements, the specification itself must meet them. By managing the requirements together with the formal model and by providing traceability between the model and the requirements, the task of verifying that the requirements have been satisfied correctly is facilitated. We developed an approach that iteratively formalises informal requirements and that creates traceability in the process (Sect. 12.7.1).

ProR was specifically developed with requirements traceability for Rodin in mind. We took a more general approach and built a generic tool that could be integrated into Rodin. The benefit of this approach is that the tool has applications beyond Rodin, thereby attracting interest and contributors from beyond the Rodin community.

We based the tool data model on the Requirements Interchange Format (ReqIF), a standard driven by the automotive industry. This gives us interoperability with a number of existing tools used in industry. Cooperation with the ITEA-project "Verde" [43], which supplied an implementation of the ReqIF data model, has been established. Interested parties include such companies as Airbus, Thales, MKS and many others. This has also triggered interest from other Eclipse-based projects, including Topcased for UML/SysML integration [21].

Tracing between informal requirements and formal models is challenging. A method for such tracing should permit us to deal with changes to both the requirements and the model efficiently. A particular challenge is posed by the interplay of formal and informal elements. We developed an incremental approach to requirements validation and systems modelling. Formal modelling facilitates a high degree of automation: it serves validation and traceability.

The foundation for our approach is requirements that are structured according to the reference model [22, 23]. We provide a system for traceability with a state-based formal method that supports refinement. We do not require all specification

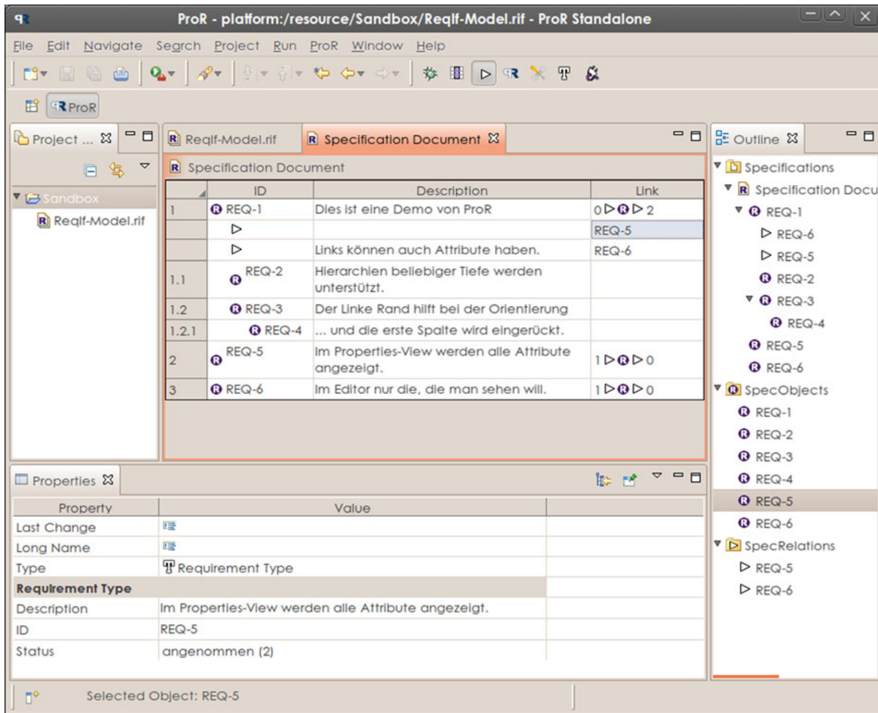


Fig. 12.3 The ProR graphical user interface, shown with some sample data

elements to be modelled formally, and we support incremental incorporation of new specification elements into the formal model. Refinement is used to deal with larger amounts of requirements in a structured way.

The integration plug-in for Rodin and ProR supports this approach by allowing the manual creation of traces between Event-B model elements and individual requirements using drag and drop. Users get additional guidance from the colour highlighting of model elements (see Fig. 12.4). This has been helpful in managing traceability. We plan on extending the existing functionality by providing advanced reporting (e.g., by finding untraced requirements or model elements) and data management (e.g., by automatic marking of traces where source or target have changed since the last validation).

12.7.2 UML Integration

The Event-B notation was developed with the goal of making the process of automated proof practical and efficient. For this goal it was important to keep the notation simple and closely based on the underlying concepts of set theory and predicate

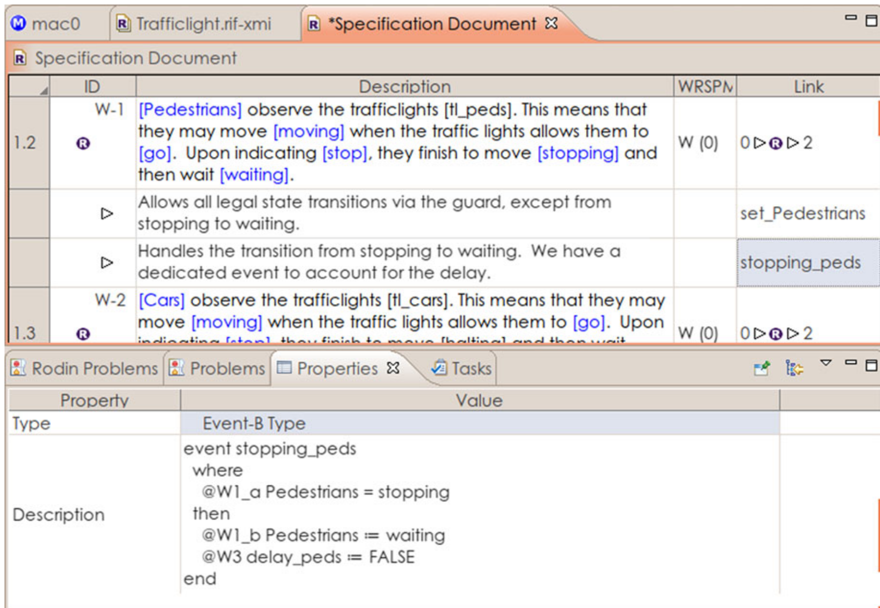


Fig. 12.4 Integration of ProR with an Event-B formal model. Model elements are highlighted in the requirements text, and annotated traces to the model show the model element in the property view

calculus, without introducing features that complicate the process of proof. Furthermore, the modelling notation is based on a paradigm of spontaneous guarded events without any means of expressing the ordering of events within a behaviour. While this philosophy achieves a simple and flexible notation and allows strong tool support, it can also give a 'low-level' feel to the language, leading to models that are difficult to create and understand. Many users feel there is a need for a higher-level notation, for example, for expressing the structure of data, related families of events and the sequencing of events, as 'syntactic sugar' to make the modelling language more useable.

UML comes with a variety of notations for expressing models in this way. The UML-B plug-in provides a UML-style diagrammatic editor visualisation of a model to help inexperienced modellers quickly develop model abstractions and experiment with them [39]. Class Diagrams support visual structuring of data and events with a lifting mechanism to 'promote' behaviours to a set of instances. Hierarchical state-machines express ordering constraints on events and allow localised state invariants to be expressed. Modelling is performed entirely via diagrammatic editors, with Event-B expressions being annotated in detail where necessary. While there is a need for the modeller to understand Event-B (in particular, proof verification is carried out entirely within generated Event-B), UML-B gives the modeller another view on the model, which can be very helpful for his/her understanding of it. Fur-

thermore, since UML is now widely adopted by engineers, the UML-B view of the model can be more easily communicated to other modellers.

The UML-B approach has strongly influenced SAP's work on integrating Rodin with SAP's in-house graphical modelling notation. The plug-in was also applied to train control modelling by Siemens and to a simplified version of the stop/start controller from Bosch.

However, more experienced modellers require greater flexibility and would like to be able to directly edit Event-B models that are enhanced with diagrams. The new state machine plug-in allows hierarchical state machines to be added to an Event-B machine and superimpose event sequencing on existing events. Modelling is mostly carried out in the conventional Event-B notation; only the sequencing of events (and localised state invariants) is expressed as a state machine. The transitions of a state machine contribute guards and actions (representing state changes) to existing events in a flexible many-transitions-to-many-events relationship. This mechanism allows state machines to synchronise at particular points (events). The plan is to add further diagrammatic notations in this contributory fashion. For example, support for class diagrams is under way.

12.7.3 Code Generation

Event-B is typically used to model software-based systems, and these require code to be produced that can be compiled. Automatic code generation tools enable implementations to be generated from Event-B. Users can therefore benefit from the use of formal development, and from the efficiency gains provided by automatic code generation. In many cases, there is a semantic gap between the modelling concepts used in Event-B, and the implementation components or source code constructs. To address this, we extend Event-B with a tasking language [8]. This approach was formulated to be of use for engineers developing multi-tasking, real-time embedded systems, so we chose the constructs of the Ada programming language as a basis for the tasking structure. We make use of high-level concepts, such as tasks and shared machines, and we model the environment in such a way that simulators can be generated as part of the development. The tasking Event-B feature bridges the gap between Event-B specification and implementation by providing implementation specification and code generation tools. Translators are currently available for Ada and C, and more can be added since extensibility has been considered during the development.

Tasking Event-B is an extension to Event-B, but includes restrictions to ensure the code is implementable. To use the tasking Event-B approach successfully, development must be structured in a particular way, which is achieved using decomposition. We decompose development into a number of Event-B machines; this makes reasoning about large developments easier since each decomposed machine may be refined further, and this, too, can be decomposed again. We use the Rodin decomposition tool (Sect. 12.6.1) to perform model decomposition, and the code generation

tool uses structural information from decomposition to structure implementation. In previous work [7] we found that models quickly turned intractable as they became very large and generated a large number of proof obligations. In that work, we described an object-oriented intermediate language that was used to guide code generation. However, we encountered difficulties due to the large semantic gap between Event-B and the intermediate specification language. In the current tool, the methodology maintains a small semantic gap; we achieve this by adding only a minimal number of constructs to the Event-B language.

Following decomposition and refinements at the implementation level, we identify implementation features such as AutoTask, Environ or Shared machines. AutoTask machines model controller tasks in the implementation. Shared Machines model protected objects (or a similar mutual exclusion mechanism), and Environ machines model the environment. We then specify some tasking features, such as the task type (e.g., periodic, triggered, one-shot, repeating), priority and the task body. We use a task body specification to write the flow control for an individual task. The flow control may be an event, sequence, loop or branch. An output construct is also provided, to allow text to be output to a console during simulation. During the early stages of the project, we realised that a self-imposed restriction on inter-task communication had forced us to adopt a particular modelling style, and this did not reflect the way that systems are implemented in industry. We removed the restriction on inter-task communication for task-environment communication since environ machines can be used to generate simulators. The main driver for the restriction on the deployable part of the system is that we wish to generate safe multi-tasking code. We aim to make the deployable code Ravenscar-compliant, and the restrictions remain in place.

The current tool also includes a solution for modelling low-level interaction with the environment, that is, sensing and actuating. The sensing and actuating events give rise to two styles of implementation: one using entry calls, and one using addressed variables. In the entry call style, calls are a way of updating sensed variables in a controller task, and controlled variables in the environment task. In the latter style, addressed variables associate address information with each of the parameters of sensing or actuating events in the controller task.

The development of the code generation approach was strongly guided by input from the industrial partners, especially Space Systems Finland, Siemens and Bosch. There was a common requirement from these partners that the tool should support generation of multi-tasking implementations with periodic and aperiodic tasks as well as variable sharing between tasks. Before the plug-in was developed, we devised a case study of a heater controller, including an Event-B development and a multi-tasking implementation in Ada. Feedback from the industrial partners indicated that the Ada implementation was representative of the typical implementation they would follow, so this served as a good benchmark for defining the structure of generated implementations. Once the code generation plug-in was developed, we were able to generate Ada and C versions of the heater controller. We also generated Ada and C implementations of a simplified version of the start/stop controller from Bosch.

12.7.4 Flow Plug-in

There are situations where the event-based modelling style of Event-B appears awkward due to the large number of event-ordering constraints necessary to express event-ordering properties. Such constraints, although trivial in their nature, can pollute a model with auxiliary variables, invariants and actions, which affects model legibility and proof automation (due to an increased number of hypotheses). The flow plug-in [10, 16, 18] helps us construct Event-B models with rich control flow properties in a concise manner. It greatly simplifies certain kinds of proofs and offers proof techniques (most importantly, assertion propagation along an event chain) that are difficult to exercise consistently in plain Event-B. The flow plug-in was used in the Bosch cruise control and stop/start system case studies.

The plug-in supports two development methods. One is to apply the graphical notation to express event orderings [18]. The effect is a set of proof obligations demonstrating that the given orderings are found among machine traces. One can effectively express point conditions (safety invariants that hold between the execution of certain two events) and prove liveness properties. The other approach is to translate use case scenarios, informally defined in a requirements specification, into the formal notation of the flow plug-in. Such scenarios are detailed in a refinement-like manner along with the refinement of the host Event-B machine [17].

The plug-in provides a modelling environment for working with graph-like diagrams describing event-ordering properties. In the simplest case, a node in such a graph is an event of the associated Event-B machine; an edge is a statement about the relative properties of the connected nodes/events.

A use case diagram is only defined in association with one Event-B model; it does not exist on its own. The use case plug-in automatically generates all the relevant proof obligations. A change in a diagram or its Event-B model leads to the recomputation of all affected proof obligations. These proof obligations are dealt with, as are all other proof obligation types, using a combination of automated provers and interactive proofs. As in the proofs of model consistency and refinement, the feedback from an undischarged use case proof obligation may often be interpreted as a suggestion for a diagram change, such as the introduction of additional assumptions or assertions—predicate annotations on graph edges that propagate properties along the graph structure.

12.7.5 Modes Plug-in

In the early stages of DEPLOY it was recognised that many challenging developments need to deal with dynamic system reconfiguration. Such models typically describe several “stable” phases of system behaviour and some activities that lead from one phase to another. This has led to the design and implementation of the modes plug-in [27]. The plug-in extends the Event-B modelling notation with a superstructure describing system modes and transitions between modes. It employs

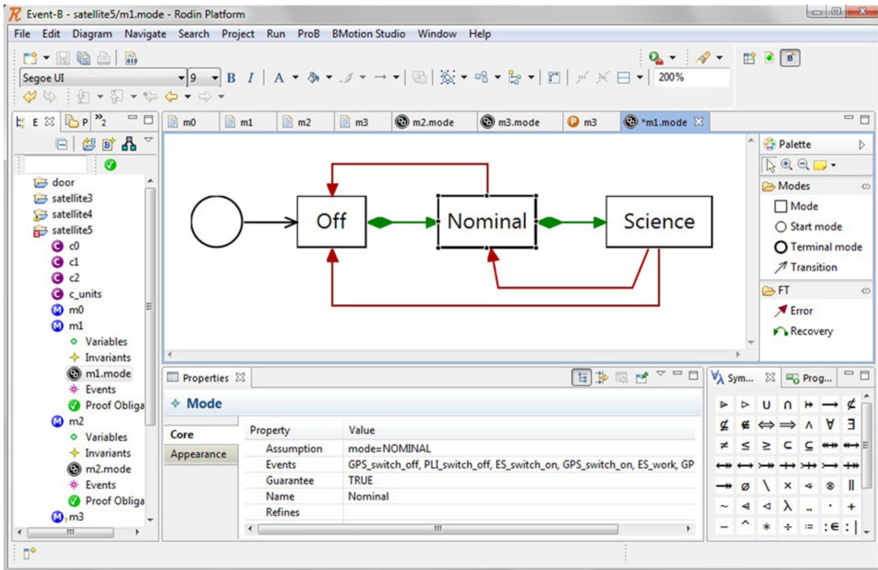


Fig. 12.5 The tool screenshot

a simple visual notation based on modecharts [19]. The graphical notation, called the modal view of a system, coexists with an Event-B machine; the two define differing viewpoints on the same design. A modal view has formal semantics (proof semantics and operational semantics) from which a set of consistency conditions are derived. These demonstrate that a modal view and the corresponding Event-B machine are in agreement.

A mode in a modal view is an island of relatively stable system behaviour. A system still evolves within a mode, but within far stricter limits than those of the safety invariant of an Event-B machine. Such limits are defined by a pair of assumption and guarantee predicates. The assumption predicate is normally interpreted as a set of conditions under which a system is able to stay in the mode; the guarantee describes what the system is doing while in the mode. The guarantee is a before-after predicate: it references both current and next states. Modes are related via mode transitions; these are also characterised formally and, in this respect, are similar to Event-B events (Fig. 12.5). The state model is borrowed from an Event-B machine.

The central feature of the tool is support for stepwise development of the modal view along with the process of Event-B refinement. When a machine is refined, a developer also needs to refine the modal view to reflect the changes in the machine (or state view). There are a number of refinement laws describing possible ways of refining a modal view; these give rise to transformational patterns offered to a developer. Hence, a refined modal view is produced by transforming an abstract modal view (Fig. 12.6).

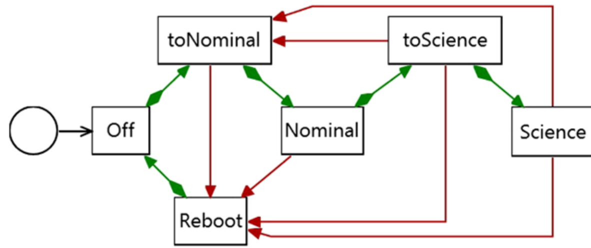


Fig. 12.6 The modal view refined

The tool automatically generates verification conditions that are necessary for ensuring that a given modal view is sound and consistent with an Event-B machine. A number of case studies have been developed using the tool to ascertain the scalability of the method and the implementation [27, 30].

The modes plug-in was used in the BepiColombo case study by Space Systems Finland. One conclusion to draw from the experience with this tool is that it is generally gratifying to stratify a design into aspects, or, as we call them, views. This permits a focused analysis and discussion of properties pertinent to a given view and an explicit connection to any requirements about modal properties.

12.8 Conclusions

At the start of the DEPLOY project we already had a version of the Rodin platform that had been developed as part of the RODIN FP6 project (2004 to 2007). Exposing the tool to serious industrial users in the DEPLOY project drove the developers to implement significant improvements in performance, usability and stability of Rodin. The experiences and expectations of the DEPLOY industrial partners also led to key innovations in tool functionality through a range of plug-ins, as outlined in this chapter.

The ease with which, in order to provide seamless functionality, the core Rodin platform may be extended with plug-ins by a range of teams indicates that the use of Eclipse, together with the Rodin extension points, provides a very effective architecture for tool extensibility. In addition to those developed within DEPLOY, several plug-ins have been developed by sites outside of DEPLOY. An up-to-date list of Rodin plug-ins is maintained on the Rodin wiki (wiki.event-b.org). It is clear that the Rodin platform has supported the evolution of a rich ecosystem of integrated tools whose development has been tempered by experiences of industrial usage. This ecosystem of tools continues to evolve.

Acknowledgements Contributions to this chapter were made by Andy Edmunds, Thai Son Hoang, Alexei Iliasov, Florian Ipate, Michael Jastram, Lukas Ladenberger, Michael Leuschel, Ilya Lopatkin, Chris Lowell, Issam Maamria, Carine Pascal, Daniel Plagge, Jann Röder, Vitaly Savicks, Matthias Schmalz, Renato Silva, Colin Snook and Alin Stefanescu.

References

1. Andrews, P.B.: An Introduction to Mathematical Logic and Type Theory. Springer, Berlin (2002)
2. Church, A.: A formulation of the simple theory of types. J. Symb. Log. **5**(2), 56–68 (1940)
3. de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: Proc. TACAS, Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer, Berlin (2008)
4. Dinca, I., Ipate, F., Mierla, L., Stefanescu, A.: Learn and test for Event-B—A Rodin plug-in. In: Proc. ABZ'12 Conference, Lecture Notes in Computer Science. Springer, Berlin (2012). http://deploy-eprints.ecs.soton.ac.uk/379/
5. Eclipse modeling framework. http://www.eclipse.org/emf
6. Eclipse Object Language (2011). http://www.eclipse.org/gmt/epsilon/doc/eol/
7. Edmunds, A., Butler, M.: Linking Event-B and concurrent object-oriented programs. In: Proc. Refine 2008—International Refinement Workshop (2008). http://eprints.ecs.soton.ac.uk/16003/
8. Edmunds, A., Rezazadeh, A., Butler, M.: Formal modelling for Ada implementations: Tasking Event-B. In: Proc. Ada Europe 2012, Lecture Notes in Computer Science. Springer, Berlin (2012)
9. Fathabadi, A.S., Rezazadeh, A., Butler, M.: Applying atomicity and model decomposition to a space craft system in Event-B. In: Proc. Third NASA Formal Methods Symposium (2011). http://eprints.ecs.soton.ac.uk/22048/
10. Plug-in, F.: Event-B wiki page. http://wiki.event-b.org/index.php/Flows
11. Gordon, M.J.C., Melham, T.F.: Introduction to HOL. Cambridge University Press, Cambridge (1993)
12. Graphical editing framework. http://www.eclipse.org/gef
13. Graphical modeling framework. http://www.eclipse.org/gmf
14. Hoang, T.S., Fürst, A., Abrial, J.-R.: Event-B patterns and their tool support. In: Proc. Seventh IEEE International Conference on Software Engineering and Formal Methods, pp. 210–219 (2009). http://deploy-eprints.ecs.soton.ac.uk/204/
15. Hoder, K.: SUMO inference engine. http://www.cs.manchester.ac.uk/~hoderk/sine
16. Iliasov, A.: Augmenting Event-B specifications with control flow information. In: Proc. NODES'10 (2010)
17. Iliasov, A.: Augmenting formal development with use case reasoning. In: Proc. Ada Europe 2012 (2012)
18. Iliasov, A.: Use case scenarios as verification conditions: Event-B/flow approach. In: Proc. 3rd International Workshop on Software Engineering for Resilient Systems, SERENE'11 (2011)
19. Jahanian, F., Mok, A.K.: Modechart: A specification language for real-time systems. IEEE Trans. Softw. Eng. **20**, 933–947 (1994). http://dx.doi.org/10.1109/32.368134
20. Jastram, M., Graf, A.: Requirements Modeling Framework. Eclipse Mag. **6.11**, 87–92 (2011)
21. Jastram, M., Graf, A.: Requirement traceability in topcased with the requirements interchange format (RIF/ReqIF). In: First Topcased Days Toulouse (2011)
22. Jastram, M., Hallerstede, S., Ladenberger, L.: Mixing formal and informal model elements for tracing requirements. In: Proc. Automated Verification of Critical Systems (AVoCS) (2011)
23. Jastram, M., Hallerstede, S., Leuschel, M., Russo, A.G.: An approach of requirements tracing in formal refinement. In: Proc. VSTTE. Springer, Berlin (2010)
24. Jastram, M.: ProR, an open source platform for requirements engineering based on RIF. In: SEISCONF (2010)
25. Leuschel, M., Butler, M.J.: ProB: An automated analysis toolset for the B method. Int. J. Softw. Tools Technol. Transf. **10**(2), 185–203 (2008). Tool webpage: http://www.stups.uni-duesseldorf.de/ProB
26. Loesch, F., Gmehlich, R., Grau, K., Mazzara, M., Jones, C.: DEPLOY deliverable D1.1: Report on pilot deployment in automotive sector (D19) (2010)

27. Lopatkin, I., Iliasov, A., Romanovsky, A.: On fault tolerance reuse during refinement. In: Proc. 2nd International Workshop on Software Engineering for Resilient Systems, SERENE'10, London, UK (2010). Available as CS-TR-1188 at Newcastle University, UK
28. Lopatkin, I., Prokhorova, Y., Troubitsyna, E., Iliasov, A., Romanovsky, A.: Patterns for representing FMEA in formal specification of control systems. Technical report 1003, TUCS Turku, Finland (2003)
29. Meng, J., Paulson, L.C.: Lightweight relevance filtering for machine-generated resolution problems. J. Appl. Log. **7**(1), 41–57 (2009)
30. Mode/FT Views wiki page. http://wiki.event-b.org/index.php/Mode/FT_Views
31. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL—A Proof Assistant for Higher-Order Logic. Lecture Notes in Computer Science, vol. 2283. Springer, Berlin (2002)
32. Paulson, L.C.: The foundation of a generic theorem prover. J. Autom. Reason. **5**(3), 363–397 (1989)
33. Riazanov, A., Voronkov, A.: The design and implementation of vampire. AI Commun. **15**(2–3), 91–110 (2002)
34. RMF: Requirements modeling framework. http://eclipse.org/rmf
35. Röder, J.: Relevance filters for Event-B. Master Thesis, ETH Zurich (2010)
36. Roederer, A., Puzis, Y., Sutcliffe, G.: Divvy: An ATP meta-system based on axiom relevance ordering. In: Proc. CADE, Lecture Notes in Computer Science, vol. 5663, pp. 157–162. Springer, Berlin (2009)
37. Schmalz, M.: Formalizing the logic of Event-B: Partial functions, definitional extensions, and automated theorem proving. PhD Thesis, ETH Zurich (2012)
38. Silva, R., Pascal, C., Hoang, T.S., Butler, M.: Decomposition tool for Event-B. Softw. Pract. Exp. **41**(2), 199–208 (2011). http://deploy-eprints.ecs.soton.ac.uk/293/
39. Snook, C., Savicks, V., Butler, M.: Verification of UML models by translation to UML-B. Lect. Notes Comput. Sci. **6957**, 251 (2011). http://eprints.ecs.soton.ac.uk/22921/
40. Sutcliffe, G., Puzis, Y.: SRASS—A semantic relevance axiom selection system. In: Proc. CADE. Lecture Notes in Computer Science, vol. 4603, pp. 295–310. Springer, Berlin (2007)
41. Transformation patterns plug-in wiki page. http://wiki.event-b.org/index.php/Transformation_patterns
42. Varpaaniemi, K.: BepiColombo models v6.4. http://deploy-eprints.ecs.soton.ac.uk/244
43. Verde, I.P.: Validation-driven design for component-based architectures. http://www.itea-verde.org/

Chapter 13
Technology Transfer

David Basin and Thai Son Hoang

Abstract This chapter presents our experience of knowledge and technology transfer within the context of the DEPLOY project. In particular, we describe some of the challenges that we faced over the course of the project and the decisions that we made, along with their justification. We also summarise the lessons learned and what we would do differently in future technology transfer projects.

13.1 Introduction

Technology transfer is essential for bridging the gap between academic research and industrial practice. It is the core of any successful deployment project. Moreover, it is especially relevant in the context of deploying formal methods, where industrial uptake is often quite slow.

Technology transfer is a two-way street. If the industrial partners are to be able to apply the project methods and tools, they must understand them and gain experience in using them. Only after this will they be capable of carrying out larger development projects on their own. Conversely, if the technology providers are to provide adequate methods, tools, and the accompanying technology transfer material that meets the industry partners' real needs, they must understand these needs and the kinds of domain-specific problems that the industrial partners wish to solve. The technology providers therefore benefit from a chance to acquire this knowledge. The aim of technology transfer within the DEPLOY project was to carry out technology transfer through this two-way flow of information.

One of the main challenges that we faced in the DEPLOY project was the diversity of the deployment partners' backgrounds. Another challenge was the difference between the academic research environment and the industrial context. These challenges posed difficulties in preparing knowledge transfer material and complicated the technology transfer process.

D. Basin (✉) · T.S. Hoang
Institute of Information Security, ETH Zurich, Zurich, Switzerland
e-mail: basin@inf.ethz.ch

T.S. Hoang
e-mail: htson@inf.ethz.ch

A. Romanovsky, M. Thomas (eds.), *Industrial Deployment of System Engineering Methods*, DOI 10.1007/978-3-642-33170-1_13,
© Springer-Verlag Berlin Heidelberg 2013

This chapter describes our experience of technology transfer within the DEPLOY project, focusing on how technology transfer was managed throughout the project, given the different deployment sectors. The rest of this chapter is structured as follows. In Sect. 13.2, we give an overview of our technology transfer context. In Sect. 13.3, we discuss the management of technology transfer within the DEPLOY project. In Sect. 13.4, we draw conclusions, including the lessons learned and what we would do differently with regard to technology transfer in future projects.

13.2 Context Overview

In this section, we present the context for technology transfer within the DEPLOY project, including our initial decisions regarding our approach to knowledge transfer.

As mentioned earlier, one of the challenges we faced was the diversity of the deployment partners' backgrounds. Within the scope of DEPLOY, there were four industrial partners from different deployment sectors:

- Automotive sector: Bosch (Germany)
- Rail transportation sector: Siemens Transportation Systems (France)
- Space systems sector: Space Systems Finland (Finland)
- Business information sector: SAP AG (Germany)

Moreover, halfway through the project, after a refocus meeting, the project members agreed to have additional "DEPLOY Associates" (DAs). The task of technology transfer was extended accordingly to cover the group of DAs, which included two partners: Critical Software Technologies Ltd. (UK) and Grupo AeS (Brazil). Our resource allocation had to be adjusted to accommodate these changes. The partners had different levels of expertise in formal methods and, in particular, in Event-B. Moreover, diversity characterised not only the industrial partners' teams as a whole, but also members of the same team.

The technology provider within the project was a group of five academic partners: Newcastle University (UK), Åbo Akademi University (Finland), ETH Zurich (Switzerland), the University of Düsseldorf (Germany) and the University of Southampton (UK). They included researchers who had laid the foundation for many formal methods and tools, in addition to having been involved in several successful industrial applications. While there is a shared knowledge base among the academic partners, which comprises refinement-based modelling methods, e.g. Event-B, the technology provider's expertise also includes various techniques for developing dependable systems, e.g., requirement engineering and model checking. Moreover, midway through the project, another two academic partners joined the project: the University of Bucharest and the University of Pitesti (both in Romania). The two additional academic partners further strengthened the technology provider team.

From the start, we employed the following technology transfer approach:

1. First, some introductory courses were organised for all industrial partners. The intended purpose of these was to introduce the industrial partners to all required aspects of the relevant existing DEPLOY methods and tools.
2. Subsequently, several mini-pilots were introduced by the industrial partners, and technology transfer was done through tackling these. The mini-pilots acted as the bridge between the industrial partners and the technology provider by clarifying domain-specific problems and by showing how to apply the techniques taught in the initial phase.

To smooth the technology transfer process, we set up a platform for communication and for exchanging technology transfer material [8]. The components of this technology transfer platform are as follows:

1. The project set up an *internal* platform (only visible to project members). This included a shared space for exchanging documents, presentations and so on, and a set of mailing lists, both structured by workpackages.
2. The project set up a *public* web platform [7]. This platform includes essential information about the DEPLOY methods and tools, for example, the Event-B modelling method, the Rodin platform and various useful plug-ins to the platform. Another part of this public web platform is a *repository* for storing publications and developments related to the DEPLOY project. The repository is organised by subject (e.g., Event-B, industrial deployment, methodology, tool development and training) and type (e.g., articles, books, deliverables, conference or workshop items, and Rodin platform archives). Another important element of this web platform is the *documentation system* in the wiki-style [6], which can be updated by registered users. The documentation system is specifically dedicated to Event-B and its supporting Rodin platform.

The technology transfer platform allows partners and other contributors to share various types of material with different audiences. Project-related (internal) documents were shared using the internal shared space and discussed via internal mailing lists. More general material was made available publicly to a wider audience via the web platform. An advantage of the wiki-style documentation system is that it also attracts contributions from outside the project. While the internal platform was only available for the project duration, we expect the external web platform to last beyond the lifetime of the project.

13.3 On Technology Transfer

13.3.1 The Initial Block Course

The importance of intensive initial knowledge transfer was recognised in the early project phases. Initial training is the basic building block for success in the later project phases. Two months after the start of the project, a three-day training course

was organised, which was attended by all industrial partners [2]. The training material was carefully prepared in order to address the differences in the participants' backgrounds [9]. An important decision made when designing material for the initial training was that it should be generic and reusable. As a result, the same material was later used to train new members of the project.

The subsequent evaluation revealed that the course was judged by all attendees to be useful for their understanding of the method (Event-B) and the supporting tool (the Rodin platform). However, it was apparent that we would have benefited from having a longer course with more intensive training, which would have helped our deployment partners to better master the DEPLOY methods and tools. Indeed, the original plan for the initial block course was to have a longer period (two weeks or at least one week) of intensive training. This would have required not only extensive effort from the academic partners in preparing materials, but also a greater commitment from the industrial partners in terms of resources for the attendees of the course. The latter was the main reason why the initial block course was not as long as planned originally.

Despite having a shorter course, we took several measures to make the most of the opportunity. Pre-reading material was sent in advance to the participants, including all the texts and slides for the course. This helped them prepare for it and familiarise themselves with the material to be taught beforehand. Note that it was important that the pre-reading material was at the right level in terms of technical details and quantity as heavy and excessive material has an undesired effect. In particular, it discourages the participants from engaging in learning before attending the course.

13.3.2 A Support Model

After the initial block course, several kick-off meetings were organised by the deployment partners to introduce the mini-pilots, as envisaged in the second phase of technology transfer. Typically, several academic partners were involved with one deployment partner. However, to strengthen the link between the deployment partners and the technology provider, we defined a model where each deployment partner is associated with one academic partner as the main contact point for conducting continuous training. For the DEPLOY project the links were as follows:

- Bosch was associated with Newcastle University;
- Siemens Transportation System was associated with the University of Southampton and later with the University of Düsseldorf;
- Space Systems Finland was associated with Åbo Akademi University;
- SAP AG was associated with ETH Zurich;
- DAs were associated with the University of Southampton.

Our intention was not to prevent other direct contact between academic partners and deployment partners. Rather, this model allowed us to consolidate the results of

the initial training, and provide a basis for giving more direct support to industrial partners for tackling their problems. This direct support model also complemented the use of various mailing lists. Whereas mailing lists are an excellent method for announcing and broadcasting information, they may lead to a lack of responsibility or prompt response to problems raised in the course of the project. By having a direct link to an academic partner, the industrial partners were assured that their problems would be addressed in a timely manner. Notice that this does not mean that the associated partner had to solve the problem by themselves; it was possible to delegate responsibility to other qualified academic partners. A danger of this direct support model is that it can lead to a lack of focus within the project. The reason for that is that despite sharing a common knowledge base, academic partners often have different research interests.

13.3.3 Technology Transfer Is Driven by Domain-Specific Problems

After the initial phase of general training, the second phase of technology transfer was working on the mini-pilots. The mini-pilots acted as the medium for sharing knowledge between the technology provider and the deployment partners. While the technology provider focused on understanding domain-specific problems, the deployment partners needed to see how the DEPLOY methods and tools could help them with these. The feedback for technology transfer in the first year of DEPLOY is detailed in [1]. The report contains the views of both deployment and academic partners. One of the main points raised in the report by the deployment partners is that technology transfer should address domain-specific issues that are key to deployment success in different industrial sectors. In other words, technology transfer should be directed by the need to solve domain-specific topics that are required for the deployment of formal methods in industry. Technology transfer should be adapted to meet this need and, as a result, there should be different technology transfer materials prepared for diverse audiences in different contexts.

Because of the differences between the domain-specific issues, one should not expect a single method or approach to meet all the needs of the deployment partners, in particular, across different sectors. Instead, there might be the need to adapt existing methods, create new methods, and even combine them. The supporting tools should be adapted as the methods evolve and as different alternatives need to be explored. As a result, the corresponding technology transfer material must also be updated as part of the process. While substantial effort is devoted to the evolution of methods and tools, the task of keeping technology transfer material up-to-date is also time-consuming and should not be neglected. Keeping technology transfer material up-to-date increases the effectiveness of methods and tools, helping transfer them to the relevant audiences more easily. In reality, however, updating technology transfer material is often overlooked.

13.3.4 A Procedure for Tracking Technology Transfer Needs

To manage technology transfer at the project level, and to avoid duplicated efforts, we defined a procedure centred around a "wishlist" that is maintained within the internal platform for technology transfer. This contains information about what material is available and what material is requested, which partner is responsible and how much time is expected to be necessary to fulfil the request. More information about this procedure is presented in [5]. The wishlist and the accompanying procedure helped ensure that the industrial partners' requests related to training were taken into account and managed accordingly. The requests that we received were mainly for adding and updating documentation, and varied in urgency. There are several ways of responding to requests, including providing new documentation, pointing to existing documents, courses or presentations, and private communication. In particular, some requests required long-term investigations of methods and tools.

What worked well in this procedure was that the initial response to a request was very fast. It was almost always possible to find a partner willing to take responsibility for a request. Some existing documents were corrected when minor problems were discovered through requests. These documents also helped the industrial partners find the right documentation.

A limitation of this procedure is that documentation is created or updated on demand, i.e., only when there is an explicit request for it. This does not ensure the quality and promptness of the delivered documentation. It was up to the individual involved to keep their commitment to fulfilling a request. As a result, commitments were not always met in a timely manner, e.g., due to interference of other project duties.

In fact, the industrial partners pointed out that while documentation was sufficient for carrying out the pilots, it did not meet the common standards for industrial-strength software. It was a real challenge to produce documentation given the resource constraints on the academic partners, who also were involved in method and tool development. The main difficulty with documentation for academics is that it is often not credited as research output. Publications, which are the measure for academic research, are not necessarily suitable as user documentation, and vice versa. As a result, the wishlist model works for minor requests. For more complicated problems, clear rules on responsibilities and financial compensation are necessary.

13.3.5 A Project for Improving the Documentation

As mentioned earlier, documentation quality is a key aspect of technology transfer. Midway through the DEPLOY project, feedback from one of the industrial partners stated that the documentation does not allow an engineer to start using the tools *without significant support*. It was clear that the documentation needed to be improved, and the task was coordinated by the University of Düsseldorf. The main

aim of this documentation subproject was to "minimize the access to an expert that a user of Event-B/Rodin needs to be productive" [10]. The project was scheduled for six months, involving representative stakeholders from both industrial partners and academic partners. While the work concentrated on the Event-B modelling method and the Rodin platform, there were also ways for plug-in developers to contribute documentation of their extensions to Rodin.

An essential part of the project was to have continuous feedback from various users (both beginners and experts) of the Rodin platform. As a result, the project was divided into four iterations. After each iteration, the result was frozen for reviews and giving feedback. This ensured that the documentation could be examined at the earliest possibility and comments could be offered and taken into account during the development of the handbook. Over the course of the project, feedback was received from various users; many of them are from the DEPLOY industrial partners.

The work described here aimed not only to reorganise the existing documentation, but also to improve its presentation. This involved introducing an editorial process with corrections by a native English speaker. The effort that this required can be justified by the notable improvement in the quality of the documentation.

The important lesson that we have learned from this exercise is that there must be documentation for different audiences, including industrial engineers. *Providing industrial-strength documentation is a complex task*. While academics who develop the methods and tools should provide the initial documentation, it still requires substantial effort to bring the documentation to the level needed by industrial engineers. It requires genuine commitment to plan and deliver resources for the task. Note that, to meet industrial standards, more collaboration is required between the documentation team, developers and experts on the documented topics.

13.4 Conclusion

We have presented our experience in carrying out technology transfer within the DEPLOY project. Technology transfer played an important role in the success of the deployment process since it is the means of equipping the industrial partners with the necessary knowledge of methods and tools. Undergoing initial training as a block course followed by continuous work with mini-pilots proved to be a successful approach to technology transfer. The initial training provided general knowledge to the industrial partners, while the mini-pilots were the medium for exchanging information about domain-specific topics. Moreover, we also committed to technology transfer by appropriately addressing requests from industrial partners, and providing necessary additional support.

The first lesson that we learned was that for an effective transfer, it is essential to *establish a strong link* between industrial partners and academic partners. Bowen and Hinchey [4] also highlighted the importance of this link: technology transfer from formal methods research to practice starts by having more *real* links between industry and academia. Moreover, it is essential to maintain these links throughout

the project, in order to facilitate bidirectional dialogue between the industrial and academic partners.

Secondly, it is extremely important to *plan for technology transfer early* in a deployment project, in particular by designing *initial training*. The value of using more resources for the initial training (including committing industrial engineers to training) will exceed the cost. The initial training provides the foundation for the later phases of the deployment process. Within the DEPLOY project, we had a strong focus on the initial training, investing effort in the preparation and delivery of the block course, as well as the course participants' time and energy.

Thirdly, technology transfer should be *driven by industrial problems*. The domain-specific issues should be the main target for knowledge transfer. While general knowledge is necessary for deployment, the goal of the deployment process in the long term is solving domain-specific problems. That is why after the initial training we used the mini-pilots as focal points for directing technology transfer.

Fourthly, another important lesson learned is about the *quality of documentation*: high-quality documentation is essential for knowledge transfer to industrial engineers. Clearly, there are different levels of documentation necessary for different audiences. Moreover, for successful deployment of methods and tools within industry, the technology transfer material must meet the standards required by industry. Producing high-quality documentation demands substantial resources and is often overlooked. Hence the plan for upgrading documentation should be designed into the overall strategy for technology transfer as early as possible. As for the DEPLOY project, we should have carried out the subproject of updating documentation in an earlier project phase.

Last but not least, we learned that *academic and industrial partners have different interests*, especially when it comes to documentation. It is understandable that industrial partners desire high-quality documentation. However, most academic partners are unwilling to spend time polishing documentation. Having wrong expectations often leads to frustration, which is harmful for a collaboration project.

Based on the experience gained through the DEPLOY project, we believe that our results would have been better if we had conducted a longer block course for the initial training. The main difficulty that we faced was the need for commitment from the course participants. Given that the participants are from industry, it was a real challenge to motivate them to attend the training course for one or two weeks. Another important fact was that the three-day block course for all industrial partners was organised at ETH Zurich. Some deployment partners stated that they would have opted for a longer course with more attendees if it had been held at their industrial site. Given that there are four different industrial partners in DEPLOY, this would have required four block courses. A disadvantage therefore would have been the longer time needed to complete the training for all industrial partners. This would have also required significant resources from the academic partners.

The link that was established and maintained within the DEPLOY project was carefully designed so that industrial partners and academic partners were able to communicate ideas and share material smoothly. However, communication, in particular, responses to requests from industrial partners, could have been better. The

use of mailing lists and shared internal space does not itself ensure satisfactory responsiveness. In particular, in the early phases of the project, when industrial partners usually had many questions about the methods and tools, responsiveness was critical for maintaining the links between industrial and academic partners. Our support model, i.e., having one academic partner responsible for each industrial partner, helped improve the situation dramatically. Despite this, the link could be further strengthened by having more direct support for the industrial partners, e.g., by having some academics working together with engineers at industrial sites. Again, this would require significant resources from the academic partners, but would speed up the deployment progress.

Acknowledgements We would like to thank Jean-Raymond Abrial for his generosity in providing technology transfer material related to his Event-B book [3]. We also thank Andreas Fürst and Matthias Schmalz for their comments on the early drafts of the chapter.

Appendix

Below is a list of the available technology transfer materials.

- *Material related to Abrial's Book* [3] (wiki.event-b.org/index.php/Event-B_Language). The material includes sample chapters of the book, slides and Rodin platform archives of developments corresponding to several chapters of the book.
- *The Rodin Handbook* (handbook.event-b.org/). This is an overhauled version of the original Rodin tutorial with additional information about Event-B and the Rodin platform.
- *The Event-B and Rodin documentation wiki* (wiki.event-b.org/index.php/Main_Page). The main wikiportal for material related to the Event-B modelling method and the Rodin platform. This is available for both users and developers of Event-B/Rodin. Contributions are made by registered users both inside and outside of the DEPLOY project.
- *Rodin at Sourceforge.net* (sourceforge.net/projects/rodin-b-sharp/). The home page of the Rodin platform and plug-ins. Several mailing lists are hosted at sourceforge.net, including the Rodin users' and the Rodin developers' mailing lists.
- *Materials for the three-day block course.* These are used for the three-day initial training of the DEPLOY partners.
  - Pre-reading material (sent in advance to the attendees): deploy-eprints.ecs.soton.ac.uk/53/
  - Lectures: deploy-eprints.ecs.soton.ac.uk/54/
  - Exercises and solutions: deploy-eprints.ecs.soton.ac.uk/55/
  - Rodin platform archives: deploy-eprints.ecs.soton.ac.uk/56/

  More information on this is given in [2].
- *The DEPLOY repository.* Various training materials are available at http://deploy-eprints.ecs.soton.ac.uk.

References

1. Abrial, J.-R., Bryans, J., Butler, M., Falampin, J., Hoang, T.-S., Ilic, D., Latvala, T., Rossa, C., Roth, A., Varpaaniemi, K.: JD1 Report on knowledge transfer. DEPLOY deliverable (January 2009). http://www.deploy-project.eu/html/deliverables.html
2. Abrial, J.-R., Hoang, T.-S., Schmalz, M.: D10.1 Teaching material. DEPLOY deliverable (January 2009). http://www.deploy-project.eu/html/deliverables.html
3. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
4. Bowen, J.P., Hinchey, M.G.: Seven more myths of formal methods. IEEE Softw. **12**(4), 34–41 (1995)
5. Butler, M., Hoang, T.-S., Iliasov, A., Jones, C., Laibinis, L., Leuschel, M., Nummila, L., Plagge, D., Räsänen, T., Schmalz, M., Snook, C., Wei, W.: D10.4 Report on training experience. DEPLOY deliverable (January 2011). http://www.deploy-project.eu/html/deliverables.html
6. Event-B.org. Event-B and Rodin documentation wiki. http://wiki.event-b.org
7. Event-B.org. Event-B and the Rodin platform. http://event-b.org
8. Hoang, T.-S., Clabaut, M.: D10.2 Design and implementation of the technology transfer platform. DEPLOY deliverable (January 2009). http://www.deploy-project.eu/html/deliverables.html
9. Hoang, T.-S.: D10.3 Initial port of technology transfer material. DEPLOY deliverable (July 2009). http://www.deploy-project.eu/html/deliverables.html
10. Jastram, M., Ladenberger, L., Plagge, D., Clark, J.: The Rodin Handbook (2012). http://handbook.event-b.org/

Chapter 14
After and Outside DEPLOY: The DEPLOY Ecosystem

Alexander Romanovsky

Abstract This chapter introduces the DEPLOY ecosystem that has been created over the project lifetime and will live on after its end to ensure that its results, including the methods and tools built for it, are widely used, extended and explored, and that the community of its industrial and academic users continues to grow. This ecosystem will help reassure newcomers that the community is active, welcoming and prepared to share the knowledge and experience accumulated. The primary aim of this work was to develop support for the growing community of companies using Event-B and Rodin.

14.1 DEPLOY Ecosystem

The DEPLOY ecosystem has been created to provide continuing support for the growing community of companies using Event-B and Rodin, and to allow an industrial company to adopt new methods to be used in product development and support, possibly over decades of service lifetime. This ecosystem includes

- the deployment partners of the project and the DEPLOY Associates,
- the DEPLOY Interest Group,
- a number of ongoing and completed research projects driven by industrial needs,
- a not-for-profit company Rodin Tools Ltd.,
- an active community of tool developers ensuring that new releases of the tools are regularly issued and that reported bugs are corrected,
- various sources of training, educational and scientific materials developed and enhanced during DEPLOY.

A. Romanovsky (✉)
Newcastle University, Newcastle upon Tyne, UK
e-mail: alexander.romanovsky@ncl.ac.uk

A. Romanovsky, M. Thomas (eds.), *Industrial Deployment of System Engineering Methods*, DOI 10.1007/978-3-642-33170-1_14,
© Springer-Verlag Berlin Heidelberg 2013

14.2 Deployment Partners and DEPLOY Associates

The work of the four deployment partners (Bosch, Siemens, SAP and Space Systems Finland) has been instrumental to the success of DEPLOY as they applied the Rodin technology in their domains during the lifetime of the project—see Chaps. 3–6. Three companies (XMOS, AeG and Critical Software Technologies) became the DEPLOY Associates in the last two years of the project, becoming involved in the assessment and the deployment of Event-B/Rodin—see the results reported in Chaps. 7–9. These seven companies represent the following application domains: aerospace, transportation, business information, automotive, railway, avionics and semiconductor. The experience gained during their involvement in DEPLOY has helped them improve their development processes in various ways and provided them with knowledge that will continue to be useful to them in further use of formal methods. Some of these companies are already getting involved in new R&D projects related to Event-B and Rodin.

With the first-hand experience gained during DEPLOY, these companies, as well as other DEPLOY partners, such as technology developers and consultants from Systerel, ClearSy and CETIC, will be key to the future development and transfer of the Rodin technology.

14.3 DEPLOY Interest Group

The DEPLOY Interest Group (DIG) was created at the beginning of the DEPLOY project. It was composed of companies, universities and individuals interested in the DEPLOY objectives and results. The project established tight bidirectional links with the DIG members and ensured that the group grew substantially during the project lifetime. Special attention was given to the DIG as dedicated means were allocated to help its members gain experience with Rodin tools. The members had privileged access to information and services (bi-annual newsletter, dedicated hands-on sessions and so on). They received regular updates about the project progress and development, as well as about plans and any changes. By the end of DEPLOY, the DIG was composed of around 70 members from all continents, with equal participation from academia and industry. Several R&D projects have been initiated with the participation of DIG members. The members also produced many useful comments and much insightful feedback related to the tools, plug-ins and materials developed for the DEPLOY project.

Several DEPLOY events were organised with the help and contribution of DIG members, including

- Workshops on B Dissemination in Salvador de Bahia, Sao Paulo, Eindhoven, Natal and in Tokyo, gathering up to 80 attendees each.
- Rodin Users and Developers Workshops in Southampton (2009), Düsseldorf (2010) and Fontainebleau (2012).

DIG members attended various workshops and tutorials organised by the DEPLOY members; the final project Federated Event was held in February 2012 in Fontainebleau.

It is expected that DIG members will be closely involved in the future development and industrial application of the Rodin technology through its use in product development, industrial or public projects, open tool development and maintenance, sharing deployment experience and academic research.

14.4 Projects Outside DEPLOY

Event-B and Rodin are currently being used in industry in various application domains, as well as in several close-to-application R&D projects. Outside the DEPLOY project, several application cases have been reported; some of these are briefly described below.

In the automation of the Flushing and Culver metro lines in New York, Event-B is being used by ClearSy to formally model the whole system, including a communication-based train control. This modelling will ensure that all the equipment specifications have a high degree of consistency and correctness, thereby contributing directly to Independent Safety Assessment and improving confidence in the overall safety of the system.

The new EU-funded ADVANCE (Advanced Design and Verification Environment for the Cyber-physical System Engineering) project,[1] which involves Alstom Transport, Critical Software Technologies Ltd., Systerel and two universities, Southampton and Düsseldorf, aims to deliver methods and tools for formal modelling, verification and validation that will make it possible to produce precise models for embedded systems and help eliminate design errors before projects go into the manufacturing stage. This will improve design of embedded software systems for automated railway signalling and smart energy distribution. The project will result in further improvements and extensions to the Rodin platform.

For several years, STMicroelectronics and ClearSy have collaborated on modelling and generation of a VHDL code for a smartcard-based microcircuit based on Event-B models.

Systerel has used Rodin and Event-B in a number of railways and aerospace projects. In the railway domain they have been working on modelling train controllers and signalling systems. A study carried out by Systerel and funded by the CNES has demonstrated the suitability of this technology for validation of flight software configuration data.

The PARSEC project, funded by the French Research Administration, aims at providing development tools for critical real-time distributed systems that must be certified as meeting the most stringent standards, such as DO-178B (avionics), IEC 61508 (transportation) or Common Criteria for Information Technology Security

[1] http://www.advance-ict.eu/

Evaluation. The approach proposed by PARSEC provides an integrated toolset, including the Rodin platform, which helps software engineers meet the requirements associated with the certification of critical embedded software. Distributed applications are being modelled with Event-B.

The OISAU project, also funded by the French Defence Administration, aims at defining the standard for the architecture of future autonomous systems for military applications (more commonly called robots or drones). The primary aim of this standard is to ensure interoperability of these systems within the community of systems of similar or different origins, referred to as the system of systems. This will enable designers and the like to define software and hardware architectures that are independent of the carrier (air, land, sea) and reusable. ClearSy is in charge of modelling the functional and non-functional needs of systems. Event-B makes it possible to model the behaviour of the system in its environment, to describe it as successive refinements of its properties and functions within various operational scenarios, and to validate the final generic model.

In the SafeCap project (Overcoming the railway capacity challenges without undermining rail network safety (safecap.cs.ncl.ac.uk)), Invensys Rail is helped by Newcastle University in its work on transferring the Event-B technology into inhouse industrial railway projects. This project focuses on developing modelling techniques and tools for improving railway capacity while ensuring that safety standards are maintained. The project team (Newcastle University, Swansea University, Invensys Rail) aim to integrate proof-based reasoning about time in state-based models, exemplified by Event-B and CSP-Prover, and to provide an open tool support for verifying timed systems. The toolset to be created is based on Rodin and uses a specially developed DSL to describe track layouts. SafeCap is supported by EPSRC UK and RSSB UK.

Since April 2010 the Åbo Akademi University team has been involved in the EU-funded ARTEMIS JU project RECOMP (Reduced Certification Costs Using Trusted Multi-core Platforms—(http://atc.ugr.es/recomp/). The project is industrially driven. The team is working on creating methodologies for component-based design of safety-critical real-time systems using Event-B targeting and on developing trusted multi-core platforms.

The Dependable Systems Forum (DSF) project involves several Japanese companies, namely NTT-Data, Fujitsu, Hitachi, NEC, Toshiba and SCSK. The DSF project applied several formal methods, including Event-B and Rodin, in an industrial development. One of the conclusions arrived at was that "Event-B/Rodin is definitely one of the industrial-strength formal methods today" (http://wiki.event-b.org/index.php/Industrial_Projects).

QNX Software Systems Limited, a leading vendor of operating systems, development tools, and professional services for connected embedded systems, has been applying Rodin tools to the design of software for a simple medical device. The aim is to use the evidence provided by the tool to support a safety case and help in the approval process.

14.5 Rodin Tools Ltd., Rodin Technical Committee and Tool Developers

The development and coordination efforts initiated in the course of DEPLOY operation will be continued in the following ways.

Rodin Tools Ltd. was set up by the Newcastle University team in 2012. It works with a community of members, and is in charge of organising the following activities around the Rodin platform:

- New platform and plug-in releases: new versions may be available to members as beta-releases.
- Bug fixing: with a priority service for members.
- Consultancy services: including plug-ins to order, with preferential rates for members.
- Training: the organisation will run training courses for all, with preferential rates for members. It will also advertise training offered by others.
- Organisation of workshops: including the Rodin Users and Developers workshops, and other events, with preferential rates for members.
- Participation in conferences: including organisation of tutorials and associated dissemination events.

This organisation will be funded by members' fees, set at various levels to enable subscription by individuals, PhD students, SMEs, large companies, STREP and IP projects. For membership details see http://www.rodintools.org/.

Rodin Tools Ltd. works together with the Rodin Technical Committee, which is in charge of defining the strategic development of the Rodin platform and the technical programmes of the Rodin Users and Developers workshops. The Committee is also responsible for running the event-b.org website (http://www.event-b.org/), which supports users and developers of the Rodin toolset. The source code and tool development is overseen by the Committee and will continue to be hosted on the SourceForge facilities. The two organisations continue to sustain and grow an active community of tool developers, ensuring that the new releases of the tools are regularly issued and the reported bugs are corrected.

14.6 How Companies Can Get Involved with the Technology

The DEPLOY project has established clear ways to learn about and install the technology, study and share the experience, obtain support for developing small and medium-size projects and organise training for various levels of expertise. All these steps are supported by various elements of the DEPLOY ecosystem, which ensures that new and already existing users of the technology are always supported in their work, the technology is maintained and there is a community of users sharing their experience.

Below are the main sources of information about the technology (see Chap. 13 for more information):

- Book: Abrial J.-R., Modeling in Event-B: System and Software Engineering. Cambridge University Press, May 2010.
- Material related to the book with some examples and slides (http://www.event-b.org/abook.html).
- The main DEPLOY site (http://www.deploy-project.eu/), which provides access to the project description and all project materials, including public deliverables and newsletters.
- The DEPLOY publication repository (http://deploy-eprints.ecs.soton.ac.uk/), which contains various training materials, models, tutorials, training courses and research papers and reports. This includes material developed and used during the three-day block course run in the first months of DEPLOY for the project deployment partners.
- The Event-B and Rodin Documentation wiki (http://wiki.event-b.org/index.php/Main_Page), the main portal for material related to the Event-B modelling method and the Rodin platform, including documentation for users of the platform and all of its plug-ins.
- Rodin user's handbook (http://handbook.event-b.org/), developed during the last year of DEPLOY; the final release was issued before the project end. The handbook is intended to help the user get acquainted with the tool platform and outlines the basics of modelling. It can also be used as a companion guide for experts. It will remain available after the project end.
- A repository of evidence of formal methods being adopted in industry (http://www.fm4industry.org), set up to provide answers to recurring queries of companies wishing to investigate the use of formal methods. Initially, it included only the evidence collected in DEPLOY (see Appendix B). Now the intention is for this repository to become widely used by organisations outside DEPLOY to collect evidence and share experience of the adoption of formal methods.

The event-b.org site provides pointers for downloading recent releases of the tools, and links with the dedicated SourceForge site (http://sourceforge.net/projects/rodin-b-sharp/). There is information about the procedures for bug reporting, and there are various mailing lists for discussing the experience of using the tools and receiving announcements about new releases.

To become more closely involved with the Rodin technology, have preferential access to information about its development and obtain training, companies and individuals need to work with Rodin Tools Ltd. (see above).

Finally, this book, too, is intended to serve the purpose of sharing the project experience and to help engineers who are deploying formal methods.

Acknowledgements I would like to acknowledge Thierry Lecomte's contribution of information about the projects that use Rodin and Event-B.

Chapter 15
Industrial Software Engineering and Formal Methods

Martyn Thomas and Alexander Romanovsky

Abstract In this chapter the editors consider the current state of professional software development in industry and commerce, the need for improvement, the role of formal methods and the opportunities and barriers that have been revealed by DEPLOY.

15.1 Introduction

The software industry is the biggest success of the past half-century. Aided hugely by the exponential improvements in computing power and the low cost of processors, storage, telecommunications and peripheral devices, software developers have created products and services that have revolutionised the way that almost every aspect of life is carried out. Most forecasts suggest that the rate of change will continue or even accelerate over the next half-century, as intelligent infrastructures, the *Internet of Things*, ubiquitous sensors and mobile devices underpin further transformations in everything from leisure to healthcare, financial services, transport and defence.

Some technical commentators have a less rosy view of the future. They recognise the enormous potential of a world that can fully exploit the power of software, but they point to the risks: the errors that lead to system crashes or security breaches; the complexity that robs most users of feeling fully in control of the devices that they own and on which they depend; the way in which major projects so often run far over budget and time and deliver less than was promised. These sceptics point out that software development, as practised by almost all commercial companies, is very far from being a mature engineering profession.

The very success of software has militated against this maturity. There have always been new business areas to target and new applications to develop and the

M. Thomas (✉)
Martyn Thomas Associates, London, UK
e-mail: martyn@thomas-associates.co.uk

A. Romanovsky
Newcastle University, Newcastle upon Tyne, UK
e-mail: alexander.romanovsky@ncl.ac.uk

A. Romanovsky, M. Thomas (eds.), *Industrial Deployment of System Engineering Methods*, DOI 10.1007/978-3-642-33170-1_15,
© Springer-Verlag Berlin Heidelberg 2013

early entrants have made easy money. As hardware costs have fallen, new opportunities have arisen and functionality has been more important in the marketplace than dependability. So functions that were manual have been automated and systems that were only accessible privately have been connected to the powerful and low-cost internet. One effect of the explosive growth in the exchange of digital documents around the world is that de facto standards have emerged, giving a small number of companies a huge share of the market, independently of the dependability of their products. The result is an infrastructure that contains very many insecure systems, and widespread use of software products that are so full of errors that each one has to check its manufacturer's website frequently to see whether there are critical updates to download. An industry has developed selling firewalls and software that try to provide protection against viruses, worms, keyloggers and other malware.

Yet security breaches occur daily. Many systems cannot be patched the instant a new software error is discovered, many computers have been subverted by malware and are now part of criminally controlled botnets, and many errors are exploited by criminals before they are known to the software manufacturers. There is a covert e-commerce industry from which you can buy tools to break into remote computers and other devices, or access credit card details and other information from successful thefts by others.

This situation is obviously unsustainable. The world cannot continue down a path that increasingly depends on the correct and secure behaviour of software until developers are able to routinely and cost-effectively produce software that can be shown to be correct and secure with high confidence.

Running tests on software can never deliver adequate confidence on its own, because tests examine its behaviour for a small number of specific input values and system states, whereas digital systems have many orders of magnitude more states and input values than could possibly be tested. Testing is successful for physical systems because (within well-understood limits) their behaviour can be predicted with certainty in the range of values that have been tested. If an aircraft wing or a bridge does not break when a force of X Newtons is applied, the engineers can be certain that it will not break when a force of any smaller number of Newtons is. Such laws are not applicable to software and so, in general, it is not valid to interpolate between the values tested in software.

A simple example may suffice. An integer expression $\frac{a}{b-c}$ can be tested for many combinations of values for a, b and c. If they are 32-bit integers, each can take about 4×10^9 possible values, so there are about 10^{28} possible combinations. This expression will overflow whenever b and c are equal, irrespective of the value of a. It is left to the interested reader to calculate how many tests with randomly selected values of a, b and c would be required to be certain that this error condition would be detected, and the probability of (say) one million tests with random values for a, b and c failing. Real software is, of course, vastly more complex than this example.

The solution, recognised over fifty years ago, is to utilise the mathematical nature of software and rely on analysis (supported by testing when this is the most practical way of verifying assumptions or validating requirements) to determine the behaviour of software-based systems. Mathematically based development methods make this possible, and Event-B is one example of such methods.

Throughout this book, the authors have reported the experience of deploying Event-B and its supporting tools on real software development tasks. This experience illuminates far more than the power and limitations of one formal method and its development environment. It shows us what needs to be done so that the software industry becomes a mature, engineering profession that confidently and successfully delivers safe, secure and cost-effective systems, accompanied by sufficient evidence to show that they are fully fit for purpose. We identify the main lessons that we draw from DEPLOY in the sections below, though many more insights will be found in the individual chapters of this book and in the evidence wiki described in Appendix B.

15.2 Software Engineering Must Be Soundly Based on Computer Science

Engineering is the application of science, embedded in mature development processes that incorporate best professional practice and the lessons drawn from past failures. No competent bridge designer will repeat the errors that led to the collapse of the Tacoma Narrows bridge, and it is an essential characteristic of professional engineering that disasters are studied and standards, models and methods then improved to avoid any repetition.

The core science for software engineers is, of course, computer science, and computer scientists have known for many decades that formal methods and formal analysis are the only ways to obtain strong evidence that software correctly implements its specification and has the required properties.

The software engineering profession will need to agree on what constitutes best practices, across most of the important activities that software engineers carry out routinely, and create mechanisms to improve these when a major failure shows that they are deficient or when a major success reveals a better way of doing things. The profession must also teach these practices to subsequent generations of engineers, so that each generation builds on the successes of the past and the profession as a whole moves forward.

The current lack of any agreement on best practices is evident. To give one basic example, much of the software industry chooses to implement software in programming languages that do not enforce type checking or bound checking on arrays, even though damage to the value of billions of dollars has already been caused by malware that exploits the errors and vulnerabilities that these simple changes would eliminate.

15.3 Managing the Requirements Gap

For a system to be fit for its purpose, it is not enough that the software (and hardware) correctly implement their specifications because, as Michael Jackson has eloquently explained in his books and papers, we are not usually much interested in

what happens at the boundary of the computer system; the real requirements are usually actions in the real world, related to the prevention of collisions between vehicles, the timely delivery of food, the consistent and reliable transmission of money, the successful maintenance of a patient's health, and so on. These requirements cannot be formally specified without abstracting away or making strong assumptions about many factors that may have a very significant influence on success, such as the impact of weather, the actions taken by people using this or some other system, or the response of a patient to the administration of a drug.

There is a gap between the complete set of things that can affect the outcome we desire, and the much smaller set of things that we can formally specify as the environment and desired behaviour of our system. All engineers face this problem, though the high complexity of software-based systems (for the reasons described earlier) makes software engineers especially aware of the difficulties. As Gmehlich and Jones explained in Chap. 3, approaches such as Jackson's *Problem Frames* go some way to managing this gap. However, a key point to make is that the inevitable gap between the *real requirements* and any formal specification is no reason to doubt the importance of formal development from specification to working software, which provides strong assurance that the software correctly implements its specification and that particular classes of error and vulnerability simply cannot exist in the delivered system.

15.4 Choosing the Right Tool for the Job

Software engineers have very many methods, languages and tools available to them and, as in any branch of engineering, it is important to be able to use a range of tools and to select the right tool for the task in hand. In DEPLOY, after some experiments, STS selected the ProB model checker as the appropriate tool for validating their complex and safety-critical network data. ProB not only has the power to do the analysis very quickly but, if it finds an error, is able to provide a specific counterexample that reveals exactly where the error will occur. STS described this experience in Chap. 4. We hope that the growing evidence repository that is part of the DEPLOY legacy (see Appendix B) will help software engineers locate the appropriate tools for their particular tasks.

There are many formal methods and tools at differing stages of maturity, designed to address the particular issues that arise in different sorts of system. Gmehlich and Jones explained in Chap. 3 that the "Duration Calculus" or the related work by Mahony and Hayes [1] could be used to address the real-time issues that Bosch and SSF found could not be expressed satisfactorily in Event-B. In Chap. 5, Ilić, Laibinis, Latvala, Troubitsyna and Varpaaniemi describe the work they carried out to explore ways of augmenting Event-B modelling with verification of real-time properties in the model-checking tool UPPAAL.

Some computer scientists have reservations about the use of formal methods that have different theoretical foundations for the same development. But software engineers are not pure scientists and, from an engineering standpoint, these reservations

need not cause us concern. The goal of an engineer is fitness for purpose and cost-effectiveness, not absolute scientific perfection. Good engineering involves judgement, and it can be good software engineering to use one formal method to specify, design and verify the functional correctness of a system and then to model its real-time behaviour and freedom from deadlocks using a different method and toolset. Software is very varied, and it currently seems unlikely that a formal method can be devised that fully supports formal verification of every property of interest. Other engineers have tool chests and select the right tool for each task, and so must we.

15.5 Engineering Is Teamwork

It is very rare that a system is so small and simple that it can be specified, designed and developed by one engineer working alone. Development methods therefore need to support division of the overall task into parts that can be worked on in parallel and later combined, and the tools need to provide support for this way of working, ensuring control over interfaces, management of versions and variants, and the ability to create frequent baseline integrations as well as to re-create earlier versions and variants at will.

The Rodin tools were reasonably mature at the start of DEPLOY, having been developed through earlier projects that had strong industrial involvement. Despite this, the industrial partners in DEPLOY quickly raised issues about the support that Event-B provided for breaking the development into parts that could easily be re-combined. These issues, and the ways in which they were ameliorated but not completely resolved, are described in Chap. 11 and elsewhere.

Teams of engineers are dynamic: people leave and join, moving between projects and needing to acquire fluency with any new methods and tools that they encounter. The documentation and training materials are required to meet the needs of engineers from different backgrounds and help them become proficient in the least possible time. The creation of really good documentation and training is difficult and time-consuming and, unfortunately, not valued by the metrics that determine research excellence and promotion opportunities for academics. This is a significant barrier to wider adoption of formal methods because, until there is a sizeable market for tools and formal methods, no industrial company will be able to recover the cost of producing high-quality documentation and training. In DEPLOY, we were fortunate to have academic partners who recognised the problem and responded to requests from the industrial partners by producing some excellent material (see, for example, the updated Rodin tutorial (http://handbook.event-b.org/).

The case for continued public investment in this area is strong, because it is important to society that software development should rapidly move from its current craft phase into a mature engineering profession. If companies are to adopt formal methods when developing important product lines, with product lifetimes that may extend for several decades, they need to be confident that the methods and tools will be supported for as long as they need them, and this requires a stable, diverse and profitable supply industry and the rest of the ecosystem described in Chap. 14.

15.6 How Much Proof Is Enough?

Proof is the part of formal development that software engineers find most difficult. Ideally, there will be no need for the engineers to prove anything manually because the tools will do all the proving automatically, but this ideal is rarely achieved.

In Chap. 3, Gmehlich and Jones showed how many proofs were required for the start-stop system model. Of 4,215 proofs, 3,787 were proven automatically; of the remainder, 400 took less than one minute each to prove manually, 12 took up to an hour, and 13 took over an hour. They concluded that it is, in principle, feasible to prove a system of the size of the start-stop system.

In Chap. 5, Ilić, Laibinis, Latvala, Troubitsyna, and Varpaaniemi give statistics for the proportions of automatic and manual proofs in the BepiColombo models developed by SSF.

The industrial partners and Associates found that the number of proofs that were automatic depended on the structure of the Event-B model. If the prover was failing to prove a large number of the proof obligations, restructuring the Event-B—not changing the meaning or the level of detail, but just the way that the model was expressed—often increased the degree of automatic proof considerably. This is unfortunate, because the engineers want to construct their formal models in the way that feels most natural, but it is not a major barrier and it is likely to improve significantly with future developments.

Is it necessary, however, or even desirable to prove the model fully? For the most critical systems, the engineers will want complete assurance; for many systems, however, it will be enough to prove key properties. For example, it may be acceptable for a financial system to crash occasionally, but it may be much worse if there are circumstances in which it creates or loses money, because then the whole database becomes corrupted and account balances cannot be trusted.

Even in safety-critical systems, it may be enough to prove that the system fails safely rather than that it never fails at all.

There are practical issues. Very few system developments are entirely new and incorporate no pre-existing software at all. Most systems will use commercial components (operating system, graphical user interface, browser, compiler libraries and so on). These, however, are unlikely to have formal specifications or to have been proved correct (and *proven-in-use* is an oxymoron, because the system states that are exercised will depend on the other software that is running, and in any case, the usage data and error history of the current version are rarely provided, are almost certainly inadequate to give strong evidence of correctness, and will become irrelevant as soon as anything in the component is changed or updated).

This means that the properties of the new system depend on some assumptions about the software (and hardware) environment just as they depend on assumptions about the physical world in which the system will be operating. Once again, a key point to make is that the unavoidable existence of unproven software and hardware in the system is no reason to doubt the importance of formal development for the new software, as it provides strong assurance that the software correctly implements its specification. It may also be important in determining liability in the event of a serious system failure.

One of us became convinced by his experience at Praxis PLC in the 1980s and 1990s that the greatest improvement in software quality came from introducing formal specifications into large, industrial and commercial software developments. In Chap. 7, Russo comes to much the same conclusion for the range of work undertaken by AeS.

Another practical issue relates to the information that the tools can give the engineer when a proof fails. Model checkers like ProB are excellent in this respect, but theorem provers will typically report in terms that are not expressed in the language of the application or the model and that the engineers may find obscure. SAP addressed this problem for their engineers by creating a domain-specific language behind which all the mechanics of proving could be hidden. SAP's success is reported in Chap. 6 by Wieczorek, Kozyura, Wei, Roth and Stefanescu.

15.7 Providing Evidence for Change

The introduction of formal methods into a software development project is likely to encounter more barriers than would a change in programming language or a semi-formal design notation. This is for two main reasons.

Firstly, formal notations such as Event-B look mathematical, and most professional software developers are unfamiliar with reading and manipulating mathematical symbols, and so are their managers.

Secondly, the essence of formal methods is their precision. Abstraction is used in order to make absolutely precise statements that omit implementation details, and this requires a change in the way that specifications and designs are developed and understood. Where a UML or IDEF diagram only give the general idea of a requirement, an Event-B model is precise, both in what it includes and what it excludes. That is its strength, and what makes it possible to do formal verification.

Software development groups therefore face the choice between relying on traditional development methods (which have been shown to produce software that contains 5–30 defects per thousand lines) and overcoming these barriers and moving to correct-by-construction methods that can deliver far higher quality with the same or better productivity.

Of course, engineers and their managers need strong evidence of the benefits to be willing to invest in formal methods, and the nature of the evidence needed is different for different people and different roles. The evidence wiki (see Appendix B) addresses the questions asked by different roles, and the evidence collected in DEPLOY.

We did not undertake scientific, controlled experiments as part of DEPLOY because it is impractical to control for all of the factors other than formal methods that can influence the outcome of a software development. These factors include domain knowledge, individual skill, previous knowledge of the application and possible designs, fluency in the chosen methods and many more. Russo explains the approach of one small company in Chap. 7.

One important practical issue should be highlighted: it is much easier for junior technical staff to embrace new methods and to become proficient than for their seniors. This is because junior staff are relatively recent graduates who do not have a major personal investment in the methods that are being replaced, whereas senior staff may draw much of their status in the team from their experience and fluency in the existing methods. It is therefore important to train the senior staff first, so that the rapid adoption by their juniors does not undermine their position and authority.

15.8 Conclusion

We believe that DEPLOY has shown the practicality of formal methods as the basis for a mature software engineering profession, and the survey reported by Fitzgerald, Bicarregui, Larsen and Woodcock in Chap. 10 shows that many companies around the world have already set off down this path. At the current pace of change, it will take too long for enough companies to become experienced users of formal methods and the result will be that either many of the systems that are currently envisaged will not be built, or they will be built at excessive costs and contain many dangerous errors and vulnerabilities.

Wieczorek, Kozyura, Wei, Roth and Stefanescu concluded from their experiences at SAP (Chap. 6) that "*Formal modelling and verification brings many quality assurance advantages for the development of business software. Design documents are complemented with software models that are accurate, executable, analysable, and can be used in deriving test cases directly linked to requirements. Formal validation and verification is not only used to prove correctness, but also to effectively find bugs, which is a nice alternative to traditional testing.*" We support this conclusion and extend it across all the application domains that the DEPLOY partners and Associates have explored. We have seen the future, and it works.

References

1. Mahony, B., Hayes, I.: Using continuous real functions to model timed histories. In: Proceedings of 6th Australian Software Engineering Conference (ASWEC91), Sydney (1991)

Appendix A
An Introduction to the Event-B Modelling Method

Thai Son Hoang

Abstract This appendix is a short introduction to the Event-B modelling method for discrete transition systems. Important mechanisms for the step-wise development of formal models, such as *context extension* and *machine refinement*, are discussed. Consistency of the models is presented in terms of proof obligations and illustrated with concrete examples.

A.1 Introduction

Event-B [2] is a modelling method for formalising and developing systems whose components can be modelled as discrete transition systems. An evolution of the (classical) B-method [1], Event-B is now centred around the general notion of *events*, which are also found in other formal methods such as Action Systems [4], TLA [6] and UNITY [5].

Event-B models are organised in terms of two basic constructs: *contexts* and *machines*. Contexts specify the static part of a model, whereas machines specify the dynamic part. The role of the contexts is to isolate the parameters of a formal model and their properties, which are assumed to hold for all instances. A machine encapsulates a transition system with the state specified by a set of variables and transitions modelled by a set of guarded events.

Event-B allows models to be developed gradually via mechanisms such as *context extension* and *machine refinement*. These techniques enable users to develop target systems from their abstract specifications, and subsequently introduce more implementation details. More importantly, properties that are proved at the abstract level are maintained through refinement, and hence are also guaranteed to be satisfied by later refinements. As a result, correctness proofs of systems are broken down and distributed amongst different levels of abstraction, which is easier to manage.

The rest of this appendix is structured as follows. We give a brief overview of the Event-B mathematical language in Sect. A.2. In Sect. A.3, we give an informal description of our running example. Subsequently, we show the basic constructs of

T.S. Hoang (✉)
Institute of Information Security, ETH Zurich, Zurich, Switzerland
e-mail: htson@inf.ethz.ch

A. Romanovsky, M. Thomas (eds.), *Industrial Deployment of System Engineering Methods*, DOI 10.1007/978-3-642-33170-1,
© Springer-Verlag Berlin Heidelberg 2013

Table A.1 Definitions

Construct	Definition
$r \in S \leftrightarrow T$	$r \in \mathbb{P}(S \times T)$
$f \in S \pfun T$	$f \in S \leftrightarrow T \wedge (\forall x, y_1, y_2 \cdot x \mapsto y_1 \in f \wedge x \mapsto y_2 \in f \Rightarrow y_1 = y_2)$
$f \in S \rightarrow T$	$f \in S \pfun T \wedge (\forall x \cdot x \in S \Rightarrow (\exists y \cdot x \mapsto y \in f))$

Event-B in Sects. A.4 (contexts) and A.5 (machines). We present the mechanisms for context extension in Sect. A.6 and machine refinement in Sect. A.7.

A.2 The Event-B Mathematical Language

The basis for the formal models in Event-B is first-order logic and a typed set theory. We are not going to give full details of the Event-B logic here. For more information, we refer the reader to [2, 8]. We present several main elements of the Event-B mathematical language that are important for understanding the Event-B models of the example below.

The first-order logic of Event-B contains standard operators such as *conjunction* ($\wedge$), *disjunction* ($\vee$), *implication* ($\Rightarrow$), *negation* ($\neg$), *equivalence* ($\Leftrightarrow$), *universal quantification* ($\forall$), and *existential quantification* ($\exists$). Two constants are defined, namely $\top$ and $\bot$, denoting *truth* and *falsity*, respectively.

A.2.1 Set Theory

An important part of the mathematical language is its set-theoretical notation, with the introduction of the membership predicate $E \in S$, meaning that expression E is a member of set S. A set expression can be a variable (depending on its type). Moreover, a set can be explicitly defined by listing its members (*set extension*), e.g. $\{E_1, \ldots, E_n\}$. Other basic set constructs include Cartesian product, power set, and set comprehension. Given two set expressions S and T, the *Cartesian product* of S and T, denoted $S \times T$, is the set of mappings (ordered pairs) $x \mapsto y$, where $x \in S$ and $y \in T$. The *power set* of S, denoted $\mathbb{P}(S)$, is the set of all subsets of S. Finally, given a list of variables x, a predicate P constraining x and an expression E depending on x, the *set comprehension* $\{x \cdot P \mid E\}$ is the set of elements E where P holds.

A key feature of the Event-B set-theoretical notation is the models of relations as sets of mappings. Different types of relations and functions are also defined as sets of mappings with different additional properties. Given two set expressions S and T, $S \leftrightarrow T$ denotes the set of all *binary relations* from S to T. Similarly, $S \pfun T$ denotes the set of all *partial functions* from S to T, and $S \rightarrow T$ denotes the set of all *total functions* from S to T. Definitions of these relations can be seen in Table A.1, expressed using set memberships.

Table A.2 Calculating well-definedness conditions using $\mathscr{L}$

Formula	Well-definedness condition
x	$\top$
$\neg P$	$\mathscr{L}(P)$
$P \wedge Q$	$\mathscr{L}(P) \wedge (P \Rightarrow \mathscr{L}(Q))$
$\forall x \cdot P$	$\forall x \cdot \mathscr{L}(P)$
$E_1 \div E_2$	$\mathscr{L}(E_1) \wedge \mathscr{L}(E_2) \wedge E_2 \neq 0$
$E_1 \leq E_2$	$\mathscr{L}(E_1) \wedge \mathscr{L}(E_2)$
$\mathrm{card}(E)$	$\mathscr{L}(E) \wedge \mathrm{finite}(E)$
$f(E)$	$\mathscr{L}(E) \wedge f \in S \nrightarrow T \wedge E \in \mathrm{dom}(f)$

Intuitively, a binary relation r from S to T is a set of mappings $x \mapsto y$, where $x \in S$ and $y \in T$. A partial function f from S to T is a binary relation from S to T, where each element x in S has *at most* one mapping to T. A total function f from S to T is a partial function from S to T, where each element x in S has *exactly* one mapping to T.

A.2.2 Types

Variables in Event-B are strongly typed. A type in Event-B can be built-in (e.g., BOOL, $\mathbb{Z}$) or user-defined. Moreover, given types T, T_1 and T_2, the Cartesian product $T_1 \times T_2$ and the power set $\mathbb{P}(T)$ are also types. In contrast with most strongly typed programming languages, the types of variables in Event-B are not presented when they are declared. Instead, they are inferred from constraining properties of variables. Typically, a property of the form $x \in E$, where E is of type $\mathbb{P}(T)$, allows us to infer that x has type T.

A.2.3 Well-Definedness

Event-B requires every formula to be *well defined* [7, 8]. Informally, one has to prove that partial functions (either predefined, e.g. division $\div$, or user-defined) are never evaluated outside of their domain. Ill-defined expressions (such as $x \div 0$) should be avoided. A syntactic operator $\mathscr{L}$ is used to map formulae to their corresponding well-definedness conditions. Table A.2 shows the definition of well-definedness condition using $\mathscr{L}$ for formulae in Event-B. Here x is a variable, P and Q are predicates, E, E_1, E_2 are expressions, and f is a binary relation from S to T. Moreover, $\mathrm{dom}(f)$ denotes the domain of f, i.e. the set of all elements in S that connect to an element in T.

Notice that by using $\mathscr{L}$, we assume a *well-definedness order* from left to right (e.g., for $P \wedge Q$) for formulae (this is similar to evaluating conditional statements, e.g. && or ||, in several programming languages).

A.2.4 Sequents

Event-B defines *proof obligations*, which must be proved to show that formal models fulfil their specified properties. Often these verification conditions are expressed in terms of *sequents*. A sequent $H \vdash G$ means that the *goal* G holds under the assumption of the set of *hypotheses* H. The obligations are discharged using certain inference rules, which we will not describe here. Instead, we will give informal justification of how proof obligations can be discharged. The purpose of presenting proof obligations within this appendix is to illustrate various conditions that need to be proved to maintain the consistency of the example.

A.3 Example. A Course Management System

The running example that we are going to use for illustrating Event-B is a course management system. We describe a requirements document of the system as follows. A club has some fixed *members*; amongst them are *instructors* and *participants*. Note that a member can be both an instructor and a participant.

ASM 1	Instructors are members of the club.

ASM 2	Participants are members of the club.

There are predefined *courses* that can be offered by a club. Each course is associated with exactly one fixed instructor.

ASM 3	There are predefined courses.

ASM 4	Each course is assigned to one fixed instructor.

A course is either *opened* or *closed* and is managed by the system.

REQ 5	A course is either *opened* or *closed*.

REQ 6	The system allows a closed course to be opened.

REQ 7	The system allows an opened course to be closed.

The number of opened courses is limited.

REQ 8	The number of opened courses cannot exceed a given limit.

Only when a course is opened, can participants *register* for the course. An important constraint for registration is that an instructor cannot attend his own courses.

REQ 9	Participants can only register for an opened course.

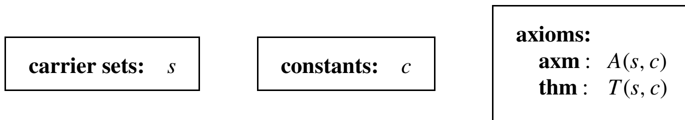
REQ 10	Instructors cannot attend their own courses.

In subsequent sections, we develop a formal model based on the above requirements document. In particular, we will refer to the above requirements in order to justify how they are formalised in the Event-B model.

A.4 Contexts

A context may contain *carrier sets*, *constants*, *axioms*, and *theorems*. Carrier sets are user-defined types. By convention, a carrier set s is *non-empty*, i.e., satisfying $s \neq \varnothing$, and *maximal*, i.e., satisfying $\forall x \cdot x \in s$. Constants c denote *static objects* within the development.[1] Axioms are *presumed properties* of carrier sets and constants. Theorems are *derived properties* of carrier sets and constants. Carrier sets and constants model the parameters of development. Moreover, axioms state parameter properties, assumed to hold for all their possible instances. A context C with carrier sets s, constants c, axioms $A(s, c)$, and theorems $T(s, c)$ can be presented as follows.

[1]When referring to carrier sets s and constants c, we usually allow for multiple carrier sets and constants, i.e., they may be "vectors".



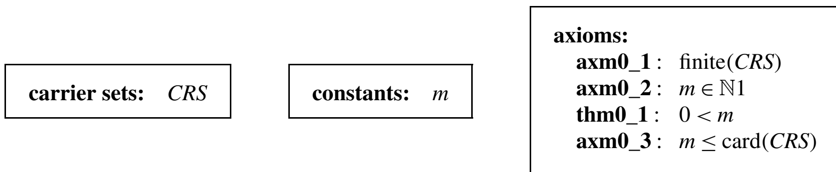
Note that we present axioms and theorems using different *labels*, i.e., **axm** and **thm**. Later on, we also use different labels for other modelling elements, such as invariants (**inv**), guards (**grd**) and actions (**act**).

Proof obligations are generated to ensure that the theorems are derivable from *previously* defined axioms. This is captured by the following proof obligation rule, called THM:

$$A(s, c) \vdash T(s, c). \quad \text{(THM)}$$

A.4.1 Example. Context coursesCtx

In this initial model, we focus on the opening and closing of courses by the system. As a result, our initial context coursesCtx contains a carrier set *CRS* denoting the set of courses that can be offered by the club (ASM 3). Moreover, coursesCtx includes a constant m denoting the maximum number of courses that the club can have at the same time (with respect to requirement REQ 8). The context coursesCtx is as follows.



Note that we label the axioms and theorems with prefixes denoting the role of the modelling elements, i.e., **axm** and **thm**, with some numbers. For example, **axm0_1** denotes the first (i.e., 1) axiom for the initial model (i.e., 0). We apply this systematic labelling throughout our development.

The assumptions on *CRS* and m are captured by the axioms and theorems as follows. Axiom **axm0_1** states that *CRS* is finite. Axiom **axm0_2** states that m is a member of the set of natural numbers (i.e., m is a natural number). Finally, axiom **axm0_3** states that m cannot exceed the number of possible courses that can be offered by the club, represented as card(*CRS*), the cardinality of *CRS*. A derived property of m is presented as theorem **thm0_1**.

thm0_1/THM A proof obligation is generated for **thm0_1** as follows. Notice that **axm0_3** does not appear in the set of hypotheses for the obligation, since it is declared after **thm0_1**. By convention, each proof obligation is labelled according to the element involved and the name of the proof obligation rule. Here **thm0_1**/THM indicates that it is a THM proof obligation for **thm0_1**.

$\begin{array}{l} \quad \mathrm{finite}(CRS) \\ \quad m \in \mathbb{N}1 \\ \vdash \\ \quad 0 < m \end{array}$	**thm0_1**/THM

The obligations can be trivially discharged since $\mathbb{N}1$ is the set of all positive natural numbers, i.e. $\{1, 2, \ldots\}$.

axm0_3/WD It is required to prove that **axm0_3** is well defined. The corresponding proof obligation is as follows.

$\begin{array}{l} \quad \mathrm{finite}(CRS) \\ \quad m \in \mathbb{N} \\ \quad 0 \leq m \\ \vdash \\ \quad \mathrm{finite}(CRS) \end{array}$	**thm0_1**/WD

Since the goal appears amongst the hypotheses, the proof obligation can be discharged trivially. Note that the order of appearance of the axioms is important. In particular, **axm0_1** needs to be declared before **axm0_3**.

A.5 Machines

Machines specify behavioural properties of Event-B models. In order to have access to information on context C, defined in Sect. A.4, machine M must connect with C. When machine M *sees* context C, it has access to C's carrier sets s and constants c, to refer to them when modelling, and C's axioms $A(s, c)$ and theorems $T(s, c)$, to use them as assumptions during proving. For clarity, in the following presentation we will not refer explicitly to the modelling elements of C. Note that in general a machine can see several contexts.

Machines M may contain *variables*, *invariants*, *theorems*, *events*, and a *variant*. Variables v define the *state* of a machine and are *constrained* by invariants $I(v)$. Theorems $R(v)$ are *additional properties* of v derivable from $I(v)$.

variables: v

invariants:	
inv :	$I(v)$
thm :	$R(v)$

A proof obligation (also called THM) is generated to prove that the theorem $R(v)$ is derivable from $I(v)$:

$$I(v) \vdash R(v). \qquad \text{(THM)}$$

Possible state changes are described by events (see Sect. A.5.1). The variant is used to prove convergence properties of events (see Sect. A.5.2).

A.5.1 Events

An event e can be represented by the term

$$\mathsf{e} \mathrel{\widehat{=}} \textbf{any}\ x\ \textbf{where}\ G(x, v)\ \textbf{then}\ Q(x, v)\ \textbf{end}, \tag{A.1}$$

where x stands for the event *parameters*,[2] $G(x, v)$ is the *guard* (the conjunction of one or more predicates) and $Q(x, v)$ is the *action*. The guard states the necessary condition under which an event may occur. The event is said to be *enabled* in a state, if there exists some value for its parameter x that makes its guard $G(x, v)$ hold in this state. The action describes how the state variables evolve when the event occurs. We use the short form

$$\mathsf{e} \mathrel{\widehat{=}} \textbf{when}\ G(v)\ \textbf{then}\ Q(v)\ \textbf{end} \tag{A.2}$$

when the event does not have any parameters, and we write

$$\mathsf{e} \mathrel{\widehat{=}} \textbf{begin}\ Q(v)\ \textbf{end} \tag{A.3}$$

when, in addition, the event guard always holds (i.e., equals $\top$). A dedicated event in the form of (A.3) is used for the *initialisation* event (usually represented by init). Note that events may be annotated with their convergence status, which we will look at in Sect. A.5.2.

The action of an event is composed of one or more *assignments* of the form

$$a := E(x, v) \tag{A.4}$$

or

$$a :\in E(x, v) \tag{A.5}$$

or

$$a :\mid P\big(x, v, a'\big), \tag{A.6}$$

where a is some of the variables contained in v, $E(x, v)$ is an expression, and $P(x, v, a')$ is a predicate. Note that the variables on the left-hand side of the assignments contained in an action must be disjoint. In (A.5), a must be a single variable, whereas it can be a vector of variables in (A.4) and (A.6). In particular, in (A.4), if a is a vector containing $n > 0$ variables, then E must also be a vector of expressions, one for each of the n variables. Assignments of the form (A.4) are *deterministic*, whereas assignments of the other two forms are *nondeterministic*. In (A.5), a is assigned an element of a set $E(x, v)$. (A.6) refers to P, which is a *before-after predicate* relating the values v (before the action) and a' (afterwards). (A.6) is also the

[2] When referring to variables v and parameters x, we usually allow for multiple variables and parameters, i.e., they may be “vectors”.

most general form of assignment and nondeterministically selects an after-state a' satisfying P and assigns it to a. Note that the before-after predicates for the other two forms are as expected; namely, $a' = E(x, v)$ and $a' \in E(x, v)$, respectively.

All assignments of an action $Q(x, v)$ occur simultaneously, which is expressed by conjoining their before-after predicates. Hence each event corresponds to a before-after predicate $\boldsymbol{Q}(x, v, v')$ established by conjoining all before-after predicates associated with each assignment and $b = b'$, where b is unchanged variables. Note that the initialisation init therefore corresponds to an *after predicate* $\boldsymbol{K}(v')$, since there are no states before initialisation.

A.5.1.1 Proof Obligations

Below we describe some important proof obligation rules for Event-B machines, namely, invariant establishment and preservation, and action feasibility.

Invariant Establishment and Preservation An essential feature of an Event-B machine M is its invariant $I(v)$. It shows properties that hold in every reachable state of the machine. Obviously, this does not hold a priori for any machines and invariants, and therefore must be proved. A common technique for proving an invariant property is to prove it by induction: (1) to prove that the property is established by the initialisation init (*invariant establishment*), and (2) to prove that the property is maintained whenever variables change their values (*invariant preservation*).

Invariant establishment states that any possible state after initialisation given by the after predicate $\boldsymbol{K}(v')$ must satisfy the invariant I. The proof obligation rule is as follows:

$$\boldsymbol{K}(v') \vdash I(v'). \tag{INV}$$

Invariant preservation makes it necessary to prove that every event occurrence *reestablishes* the invariants I. More precisely, for every event e, assuming the invariants I and e's guard G, we must prove that the invariants still hold in any possible state after the event execution given by the before-after predicate $\boldsymbol{Q}(x, v, v')$. The proof obligation rule is as follows:

$$I(v), G(x, v), \boldsymbol{Q}(x, v, v') \vdash I(v'). \tag{INV}$$

Note that in practice, by the property of conjunctivity, we can prove the establishment and preservation of each invariant separately.

Feasibility *Feasibility* states that the action of an event is always feasible whenever the event is enabled. In other words, there are always possible after values for the variables satisfying the before-after predicate. In practice, we prove feasibility for individual assignment of the event action. For deterministic assignments, feasibility holds trivially. The feasibility proof obligation generated for a nondeterministic assignment of the form $a :| P(x, v, a')$ is as follows:

$$I(v), G(x, v) \vdash \exists a' \cdot P(x, v, a'). \tag{FIS}$$

A.5.2 Event Convergence

A set of events can be proved to collectively converge. In other words, these events cannot take control forever and hence one of the other events eventually occurs. We call these events *convergent*. To prove this, one specifies a variant V which maps a state v to a natural number. One then proves that each convergent event strictly decreases V. More precisely, let e be a convergent event where v is the state before executing e and v' is the state after. Then for each such e, v, and v', one proves that $V(v') < V(v)$, additionally assuming all invariants and the e guard. Since the variant maps a state to a natural number, V induces a well-founded ordering on system states given by the strictly less than order ($<$) of their images under V. The following proof obligation rules apply to every convergent event where VAR ensures the decrement of the variant and NAT ensures that the variant is a natural number when the event is enabled:

$$I(v), G(x, v), \boldsymbol{Q}\big(x, v, v'\big) \vdash V\big(v'\big) < V(v) \tag{VAR}$$

$$I(v), G(x, v) \vdash V(v) \in \mathbb{N}. \tag{NAT}$$

Note that in some cases the convergence of some events can be shown only in a later refinement, but not immediately. In this case, their convergence is *anticipated* and we must prove that $V(v') \leq V(v)$, that is, these anticipated events do not increase the variant. Anticipated events must obey NAT and the following proof obligation rule, also called VAR:

$$I(v), G(x, v), \boldsymbol{Q}\big(x, v, v'\big) \vdash V\big(v'\big) \leq V(v). \tag{VAR}$$

As mentioned above, variant V is a natural number. Alternatively, V can be a *finite* set expression. In this case, for convergent events, one has to prove that it decreases the variant according to the strict subset inclusion $\subset$ ordering. For anticipated events, we ensure that these events do not increase the variant by proving that $V(v') \subseteq V(v)$. The proof obligation rule VAR is adapted accordingly.

For proving that the variant V is a finite set, the following proof obligation rule called FIN applies:

$$I(v) \vdash \text{finite}\big(V(v)\big). \tag{FIN}$$

Note that FIN needs to be proved once, i.e., it does not depend on the set of convergent and anticipated events (cf. NAT).

The convergence attribute of an event is denoted by the keyword **status** with three possible values: *convergent*, *anticipated*, and *ordinary* (for events which are neither convergent nor anticipated). Events are *ordinary* by default.

A.5.3 Deadlock-Freeness

A machine M is said to be *deadlocked* in some state if all of its events are disabled in that state. *Deadlock-freeness* for M ensures that there are always some enabled

events during the execution of M. Assume that M contains a set of n events e_i ($i \in 1, \ldots, n$) of the following form:

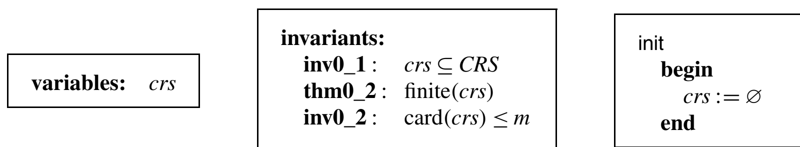
$$\mathsf{e_i} \;\widehat{=}\; \textbf{any}\ x_i\ \textbf{where}\ G_i(x_i, v)\ \textbf{then}\ Q_i(x_i, v)\ .\ \textbf{end}$$

The proof obligation rule for deadlock-freeness[3] is as follows:

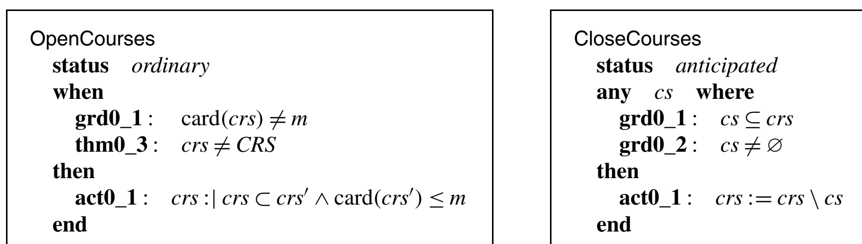
$$I(v) \vdash \big(\exists x_1 \cdot G_1(x_1, v)\big) \lor \cdots \lor \big(\exists x_n \cdot G_n(x_n, v)\big)\,. \qquad \text{(DLF)}$$

A.5.4 Example. Machine m0

We develop machine m0 of the initial model, focusing on courses opening and closing. This machine sees context coursesCtx as developed in Sect. A.4.1, and as a result has access to the carrier set *CRS* and constant *m*. We model the set of opened courses by a variable, namely *crs* (REQ 5). Invariant **inv0_1** states that it is a subset of available courses *CRS*. A consequence of this invariant and of axiom **axm0_1** is that *crs* is finite, and this is stated in m0 as theorem **thm0_2**. Requirement REQ 8 is directly captured by invariant **inv0_2**: the number of opened courses, i.e., card(crs) is bounded above by m. Initially, all courses are closed; hence *crs* is set to the empty set ($\varnothing$).



We model the opening and closing of courses using two events OpenCourses and CloseCourses as follows (REQ 6 and REQ 7).



We deliberately choose to model these events using different features of Event-B. In OpenCourses, we use a nondeterministic action to model the fact that some new courses are opened, i.e. $crs \subset crs'$, as long as the number of opened courses will not exceed its limit, i.e. $\mathrm{card}(crs') \leq m$. The guard of the event states that the current number of opened courses has not yet reached the limit.

[3]Typically, this is encoded as a theorem in the machine after all invariants.

CloseCourses models the set of courses that are going to be closed using parameter *cs*. It is a non-empty set of currently opened courses which is captured by CloseCourses' guard. The action is modelled in a straightforward way by removing *cs* from set *crs*.

We set the convergence status for OpenCourses and CloseCourses to be *ordinary* and *anticipated*, respectively. We postpone the reasoning about the convergence of CloseCourses till later refinements. Our intention is to prove that there can only be a finite number of occurrences of CloseCourses between any two OpenCourses events.

A.5.4.1 Proof Obligations

We present some of the obligations to illustrate what needs to be proved for the consistency of m0. We applied the proof obligation rules as shown earlier in this section. Notice that we can take the axioms and theorems of the seen context coursesCtx as hypotheses in the proof obligations. For clarity, we show only the parts of the hypotheses that are relevant for discharging the proof obligations. Moreover, we also show the proof obligations in their simplified forms, e.g. when event assignments are deterministic.

thm0_2/THM This obligation corresponds to the rule THM, in order to ensure that **thm0_2** is derivable from previously declared invariants.

$\ldots$ $\mathrm{finite}(CRS)$ $crs \subseteq CRS$ $\vdash$ $\mathrm{finite}(crs)$	**thm0_2**/THM

The proof obligation holds trivially since *crs* is a subset of a finite set, i.e., *CRS*.

init/inv0_2/INV This obligation ensures that the initialisation init establishes invariant **inv0_2**.

$\ldots$ $0 \leq m$ $\vdash$ $\mathrm{card}(\varnothing) \leq m$	init/**inv0_2**/INV

Since the cardinality of the empty set $\varnothing$ is 0, the proof obligation holds trivially.

OpenCourses/thm0_3/THM This obligation ensures that **thm0_3** is derivable from the invariants and the previously declared guards of OpenCourses.

$\ldots$ $m \leq \mathrm{card}(CRS)$ $crs \in \mathbb{P}(CRS)$ $\mathrm{card}(crs) \leq m$ $\mathrm{card}(crs) \neq m$ $\vdash$ $crs \neq CRS$	OpenCourses/**thm0_3**/THM

Informally, we can derive from the hypotheses that $\mathrm{card}(crs) < \mathrm{card}(CRS)$; hence crs must be different from CRS.

OpenCourses/act0_1/FIS This obligation corresponds to rule FIS and ensures that the nondeterministic assignment of OpenCourses is feasible when the event is enabled.

$\ldots$ $crs \neq CRS$ $\mathrm{card}(crs) \leq m$ $\mathrm{card}(crs) \neq m$ $\vdash$ $\exists crs' \cdot crs \subset crs' \wedge \mathrm{card}(crs') \leq m$	OpenCourses/**act0_1**/FIS

The reasoning about the proof obligation is as follows. Since crs is different from CRS, there exists an element c which is closed, i.e., not in crs. By adding c to the set of opened courses, we strictly increase the number of opened courses by 1. Moreover, the number of opened courses after executing the event is still within the limit since originally it is strictly below the limit.

CloseCourses/inv0_2/INV This obligation corresponds to rule INV and is simplified accordingly since the assignment is deterministic. The purpose of the obligation is to prove that CloseCourses maintains invariant **inv0_2**.

$\ldots$ $\mathrm{card}(crs) \leq m$ $\vdash$ $\mathrm{card}(crs \setminus cs) \leq m$	CloseCourses/**inv0_2**/INV

Since removing some courses cs from the set of opened courses crs can only reduce its number, the proof obligation can be trivially discharged.

DLF/THM The deadlock-freeness condition is encoded as theorem **DLF** of machine m0, which results in the following proof obligation.

$\ldots$ $0 < m$ $\vdash$ $(\mathrm{card}(crs) \neq m) \vee (\exists cs \cdot cs \subseteq crs \wedge cs \neq \varnothing)$	**DLF**/THM

We reason as follows. If $\text{card}(crs) \neq m$, the goal trivially holds. Otherwise, if $\text{card}(crs) = m$, since $m \neq 0$, $crs \neq \varnothing$. As a result, we can prove that $\exists cs \cdot cs \subseteq crs \land cs \neq \varnothing$ by instantiating cs with crs.

A.6 Context Extension

Context extension is a mechanism for introducing more static details into an Event-B development. A context can *extend* one or more contexts. When describing a context D as extending another context C, we call C and D the abstract and concrete context, respectively. By extending C, D "inherits" all the abstract elements of C, i.e., carrier sets, constants, axioms and theorems. This means that (1) a context extending D also implicitly extends C, and (2) a machine seeing D also implicitly sees C. As a result, proof obligation rule THM for D also has additional assumptions in the form of axioms and theorems from the abstract context C.

Subsequently, we present three new contexts that we use in the next refinement of our running example.

A.6.1 Context membersCtx

This is an initial context (i.e., it does not extend any other context) containing a carrier set *MEM*. *MEM* represents the set of club members, with an axiom stating that it is finite.

carrier sets: *MEM*

axioms:
axm1_1 : $\text{finite}(MEM)$

A.6.2 Context participantsCtx

This context extends the previously defined context membersCtx and is as follows.

constants: *PRTCPT*

axioms:
axm1_2 : $PRTCPT \subseteq MEM$
thm1_1 : $\text{finite}(PRTCPT)$

Constant *PRTCPT* denotes the set of participants that must be members of the club as specified in ASM 2 (**axm1_2**). Theorem **thm1_1** states that there can be only a finite number of participants, which gives rise to the following trivial proof obligation.

$\begin{array}{l} \text{finite}(MEM) \\ PRTCPT \subseteq MEM \\ \vdash \\ \text{finite}(PRTCPT) \end{array}$	**thm1_1**/THM

An important point is that axiom **axm1_1** of the abstract context membersCtx appears as a hypothesis in the proof obligation.

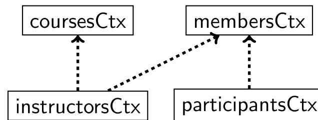
A.6.3 Context instructorsCtx

This context extends two contexts, coursesCtx and membersCtx, and introduces two constants, namely *INSTR* and *instrs*. *INSTR* models the set of instructors that are members of the club as specified by ASM 1 (**axm1_3**). Constant *instrs* models the relationship between courses and instructors and is constrained by **axm1_4**: it is a *total function* from *CRS* to *INSTR*, and hence directly formalises requirement ASM 4. Recall the definition of total function f from set S to set T: f is a relation from S to T where every element in S has exactly one mapping to some element in T.

constants: *INSTR*, *instrs*

axioms:
- **axm1_3** : $INSTR \subseteq MEM$
- **axm1_4** : $instrs \in CRS \rightarrow INSTR$

The hierarchy of context extensions for our example is summarised in the following diagram.



A.7 Machine Refinement

Machine refinement is a mechanism for introducing details about the dynamic properties of a model [2]. For more details on the theory of refinement, we refer the reader to the Action System formalism [4], which has inspired the development of Event-B. We present here the proof obligations defined in Event-B, related to refinement. When speaking about machine N refining another machine M, we refer to M as the *abstract* machine and to N as the *concrete* machine.

Despite the fact that the formal definition of Event-B refinement does not distinguish between *superposition refinement* and *data refinement*, we illustrate them in separate sections to show different aspects of the two. In superposition refinement, the abstract variables of M are retained in the concrete machine N, with possibly some additional concrete variables. In data refinement, the abstract variables v are

replaced by concrete variables w and, subsequently, the connections between M and N are represented by the relationship between v and w. In fact, more often, Event-B refinement is a mixture of both superposition and data refinement: some of the abstract variables are retained, while others are replaced by new concrete variables.

A.7.1 Superposition Refinement

As mentioned earlier, in superposition refinement, variables v of the abstract machine M are kept in the refinement, i.e. as part of the state of N. Moreover, N can have some additional variables w. The concrete invariants $J(v, w)$ specify the relationship between the old and new variables. Each abstract event e is refined by a concrete event f (later on we will relax this one-to-one constraint). Assume that the abstract event e and the concrete event f are as follows:

$$\mathsf{e} \mathrel{\widehat{=}} \textbf{any } x \textbf{ where } G(x, v) \textbf{ then } Q(x, v) \textbf{ end}$$

$$\mathsf{f} \mathrel{\widehat{=}} \textbf{any } x \textbf{ where } H(x, v, w) \textbf{ then } R(x, v, w) \textbf{ end}$$

We assume now that e and f have the same parameters x. The more general case, where the parameters are different, is presented in Sect. A.7.2.

Somewhat simplifying, we say that f refines e if the guard of f is stronger than that of e (*guard strengthening*), concrete invariants J are maintained by f, and abstract action Q simulates the concrete action R (*simulation*). These conditions are stated as the following proof obligation rules:

$$I(v), J(v, w), H(x, v, w) \vdash G(x, v) \qquad \text{(GRD)}$$

$$I(v), J(v, w), H(x, v, w), \boldsymbol{R}\big(x, v, w, v', w'\big) \vdash \boldsymbol{Q}\big(x, v, v'\big) \qquad \text{(SIM)}$$

$$I(v), J(v, w), H(x, v, w), \boldsymbol{R}\big(x, v, w, v', w'\big) \vdash J\big(v', w'\big) \qquad \text{(INV)}$$

In particular, if the guard and action of an abstract event are retained in the concrete event, the proof obligations GRD and SIM are trivial; hence we only need to consider INV for proving that the gluing invariants are reestablished.

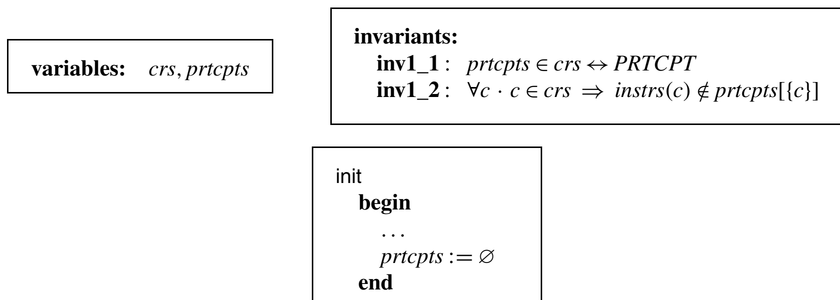
Proof obligations are generated to ensure that each assignment of concrete event f is feasible. In the case where the action of the abstract event is retained in f, we only need to prove the feasibility of any additional assignment.

In the course of refinement, *new events* are often introduced into a model. New events must be shown to refine the implicit abstract event SKIP, which does nothing, i.e., does not modify abstract variables v.

A.7.1.1 Machine m1

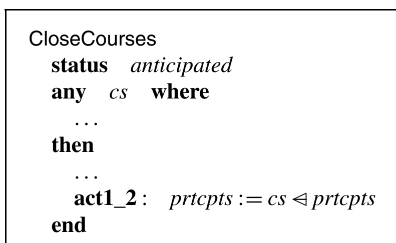
Machine m1 sees contexts instructorsCtx and participantsCtx. As a result, it implicitly sees coursesCtx and membersCtx. Variable *crs* is retained in this refinement.

An additional variable *prtcpts* representing information about course participants is introduced. Invariant **inv1_1** models *prtcpts* as a relation between the set of opened courses *crs* and the set of participants *PRTCPT*. Requirement REQ 10 is directly modelled by invariant **inv1_2**: for every opened course c, the instructor of this course, i.e., $instrs(c)$, is not amongst its participants, represented by $prtcpts[\{c\}]$.



variables: $crs, prtcpts$

invariants:
inv1_1 : $prtcpts \in crs \leftrightarrow PRTCPT$
inv1_2 : $\forall c \cdot c \in crs \Rightarrow instrs(c) \notin prtcpts[\{c\}]$



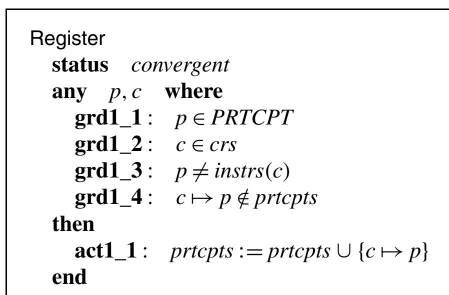
init
begin
...
$prtcpts := \varnothing$
end

Initially, there are no opened courses; hence *prtcpts* is assigned to be $\varnothing$.

The original abstract event OpenCourses stays unchanged in this refinement, while an additional assignment is added to CloseCourses to update *prtcpts* by removing the information about the set of closing courses *cs* from it.

CloseCourses
status *anticipated*
any cs **where**
...
then
...
act1_2 : $prtcpts := cs \lhd prtcpts$
end

A new event is added, namely Register, to model the registration of a participant p for an opened course c. The guard of the event ensures that p is not the instructor of the course (**grd1_3**) and is not yet registered for the course (**grd1_4**). The action of the event updates *prtcpts* accordingly by adding the mapping $c \mapsto p$ to it.



Register
status *convergent*
any p, c **where**
grd1_1 : $p \in PRTCPT$
grd1_2 : $c \in crs$
grd1_3 : $p \neq instrs(c)$
grd1_4 : $c \mapsto p \notin prtcpts$
then
act1_1 : $prtcpts := prtcpts \cup \{c \mapsto p\}$
end

We attempt to prove that Register is convergent and CloseCourses is anticipated using the following variant.

variant: $(crs \times PRTCPT) \setminus prtcpts$

The variant is a set of mappings; each links an opened course to a participant who has *not* registered for the respective course.

We present some of the important proof obligations for m1. Proof obligations GRD and SIM are trivial for events OpenCourses and CloseCourses. Consequently, we only need to consider INV for these old events.

CloseCourses/inv1_2/INV This obligation is to ensure that **inv1_2** is maintained by CloseCourses. The obligation is trivial, in particular, because, given that $c \notin cs$, $(cs \lhd prtcpts)[\{c\}]$ is the same as $prtcpts[\{c\}]$.

$\ldots$ $\forall c \cdot c \in crs \Rightarrow instrs(c) \notin prtcpts[\{c\}]$ $\vdash$ $\forall c \cdot c \in crs \setminus cs \Rightarrow instrs(c) \notin (cs \lhd prtcpts)[\{c\}]$	CloseCourses/**inv1_2**/INV

Register/inv1_1/INV This obligation is to guarantee that **inv1_1** is maintained by the new event Register.

$\ldots$ $prtcpts \in crs \leftrightarrow PRTCPT$ $p \in PRTCPT$ $c \in crs$ $\vdash$ $prtcpts \cup \{c \mapsto p\} \in crs \leftrightarrow PRTCPT$	Register/**inv1_1**/INV

FIN This obligation is to ensure that the declared variant used for proving convergence of events is finite (FIN). This is trivial, since the set of opened courses *crs* and the set of participants *PRTCPT* are both finite.

$\ldots$ $\text{finite}(crs)$ $\text{finite}(PRTCPT)$ $\vdash$ $\text{finite}((crs \times PRTCPT) \setminus prtcpts)$	FIN

CloseCourses/VAR This proof obligation corresponds to rule VAR, ensuring that anticipated event CloseCourses does not increase the variant.

$\ldots$ $\vdash$ $((crs \setminus cs) \times PRTCPT) \setminus (cs \lhd prtcpts)$ $\subseteq$ $(crs \times PRTCPT) \setminus prtcpts)$	CloseCourses/VAR

Register/VAR This proof obligation corresponds to rule VAR, ensuring that the convergent event Register decreases the variant. This is trivial since a new mapping $c \mapsto p$ is added to *prtcpts*, effectively increasing it, and hence strictly decreasing the variant.

$$
\begin{array}{l}
\quad \ldots \\
\quad c \mapsto p \notin prtcpts \\
\vdash \\
\quad (crs \times PRTCPT) \setminus (prtcpts \cup \{c \mapsto p\}) \\
\quad \subset \\
\quad (crs \times PRTCPT) \setminus prtcpts
\end{array}
\qquad \text{Register/VAR}
$$

A.7.2 Data Refinement

In data refinement, abstract variables v are removed and replaced by concrete variables w. The states of abstract machine M are related to the states of concrete machine N by *gluing invariants* $J(v, w)$. In Event-B, the gluing invariants J are declared as invariants of N and also contain the *local* concrete invariants, i.e., those constraining only concrete variables w.

Again, we assume a one-to-one correspondence between an abstract event e and a concrete event f. Let e and f be as follows:[4]

$$\mathsf{e} \;\widehat{=}\; \textbf{any}\ x\ \textbf{where}\ G(x, v)\ \textbf{then}\ Q(x, v)\ \textbf{end}$$

$$\mathsf{f} \;\widehat{=}\; \textbf{any}\ y\ \textbf{where}\ H(y, w)\ \textbf{then}\ R(y, w)\ \textbf{end}$$

As with superposition refinement, we can say that f refines e if the guard of f is stronger than the guard of e (*guard strengthening*), and the gluing invariants $J(v, w)$ establish a simulation of f by e (*simulation*). This condition is captured by the following proof obligation rule:

$$
\begin{array}{l}
\quad I(v) \\
\quad J(v, w) \\
\quad H(y, w) \\
\quad \boldsymbol{R}(y, w, w') \\
\vdash \\
\quad \exists x, v' \cdot G(x, v) \ \land\ \boldsymbol{Q}(x, v, v') \ \land\ J(v', w')
\end{array}
\tag{A.7}
$$

In order to simplify and split the above proof obligation, Event-B introduces the notion of "witnesses" for the abstract parameters x and the after value of the abstract variables v'. Witnesses are predicates of the form $W_1(x, y, v, w, w')$ (for x)

[4]Concrete events may be annotated with abstract events name and witnesses, which we will show later.

and $W_2(v', y, v, w, w')$ (for v'), which are required to be *feasible*. The corresponding proof obligations are as follows:

$$I(v), J(v, w), H(y, w), \boldsymbol{R}\big(y, w, w'\big) \vdash \exists x \cdot W_1\big(x, y, v, w, w'\big) \qquad \text{(WFIS)}$$

$$I(v), J(v, w), H(y, w), \boldsymbol{R}\big(y, w, w'\big) \vdash \exists v' \cdot W_2\big(v', y, v, w, w'\big) \qquad \text{(WFIS)}$$

Typically, witnesses are declared deterministically, i.e. in the form $x = \cdots$ or $v' = \cdots$. In these cases, witnesses are trivially feasible; hence the corresponding proof obligations are omitted.

Given the witnesses, the refinement proof obligation (A.7) is replaced by three different proof obligations as follows:

$$I(v), J(v, w), H(y, w), W_1\big(x, y, v, w, w'\big) \vdash G(x, v) \qquad \text{(GRD)}$$

$$I(v), J(v, w), H(y, w), \boldsymbol{R}\big(y, w, w'\big), W_1\big(x, y, v, w, w'\big), W_2\big(v', y, v, w, w'\big) \vdash \boldsymbol{Q}\big(x, v, v'\big) \qquad \text{(SIM)}$$

$$I(v), J(v, w), H(y, w), \boldsymbol{R}\big(y, w, w'\big), W_1\big(x, y, v, w, w'\big), W_2\big(v', y, v, w, w'\big) \vdash J\big(v', w'\big) \qquad \text{(INV)}$$

The concrete event f can be denoted by the abstract event e (using keyword **refines**) and the witnesses (using keyword **with**) as follows:

```
f
  refines  e
  any  y  where
    H(y, w)
  with
    x :  W1(x, y, v, w, w')
    v' : W2(v', y, v, w, w')
  then
    R(y, w)
  end
```

The action of the concrete event is required to be feasible. The corresponding proof obligation FIS is similar to the one presented for the abstract machine, with the exception that both abstract and gluing invariants can be assumed.

For newly introduced events, as with superposition refinement, they must be proved to refine the implicit abstract event SKIP, which is unguarded and does nothing, i.e. does not modify abstract variables v. In this case, GRD is trivial, since the abstract guard is $\top$. For SIM and INV, we omit references to W_1 (since there are no parameters for the abstract SKIP event). Moreover, the witness W_2 for v' is trivial: $v' = v$.

As mentioned earlier, in general, Event-B refinement is a mixture of both superposition and data refinement. Often, some (not all) abstract variables are retained

in the refinement, while the other abstract variables are replaced by new concrete variables. Similarly, some abstract parameters can be present in the concrete event, where other parameters are replaced by some new concrete ones. In general, we only need to give witnesses to disappearing variables and parameters.

The one-to-one correspondence between the abstract and concrete events can be relaxed. When an abstract event e is refined by more than one concrete event f, we say that the abstract event e is *split* and prove that each concrete f is a valid refinement of the abstract event. Conversely, several abstract events e can be refined by one concrete f. We say that these abstract events are *merged* together. A requirement for merging events is that the abstract events must have identical actions. When merging events, we need to prove that the guard of the concrete event is stronger than the disjunction of the guards of the abstract events.

The concrete machine N can be proved to be *relatively deadlock-free* with respect to the abstract machine M. It means that if M can continue in some state, so can N. Assume that M contains a set of n events e_i $(i \in 1, \ldots, n)$ of the following form:

$$\mathsf{e_i} \;\widehat{=}\; \textbf{any}\ x_i\ \textbf{where}\ G_i(x_i, v)\ \textbf{then}\ Q_i(x_i, v)\ \textbf{end}$$

Assume that N contains a set of m events f_j $(j \in 1, \ldots, m)$ of the following form:

$$\mathsf{f_j} \;\widehat{=}\; \textbf{any}\ y_i\ \textbf{where}\ H_i(y_i, w)\ \textbf{then}\ R_i(y_i, w)\ \textbf{end}$$

The proof obligation rule for relative deadlock-freeness[5] is as follows:

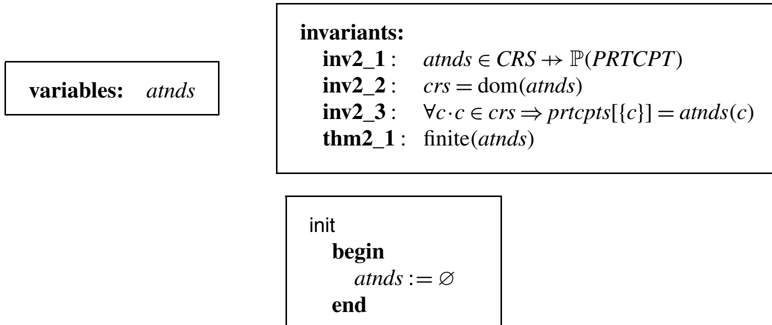
$$\begin{array}{ll} & I(v) \\ & J(v, w) \\ & \big(\exists x_1 \cdot G_1(x_1, v)\big) \;\vee\; \cdots \;\vee\; \big(\exists x_n \cdot G_n(x_n, v)\big) \\ \vdash & \\ & \big(\exists y_1 \cdot H_1(y_1, w)\big) \;\vee\; \cdots \;\vee\; \big(\exists y_m \cdot H_m(y_m, w)\big) \end{array} \qquad \text{(REL_DLF)}$$

A.7.2.1 Machine m2

We perform a data refinement by replacing abstract variables *crs* and *prtcpts* by a new concrete variable *atnds*. This machine does not explicitly model any requirements from Sect. A.3: it implicitly inherits requirements from previous abstract machines. As stated in invariant **inv2_1**, *atnds* is a *partial function* from *CRS* to some set of participants (i.e., member of $\mathbb{P}(PRTCPT)$). Invariants **inv2_2** and **inv2_3** act as gluing invariants, linking abstract variables *crs* and *prtcpts* with concrete variable *atnds*. Invariant **inv2_2** specifies that *crs* is the domain *atnds*. Invariant **inv2_3** states that for every opened course c, the set of partic-

[5]Typically, this is encoded as a theorem in N after declaration of all invariants.

ipants attending that course represented abstractly as $prtcpts[\{c\}]$ is the same as $atnds(c)$.



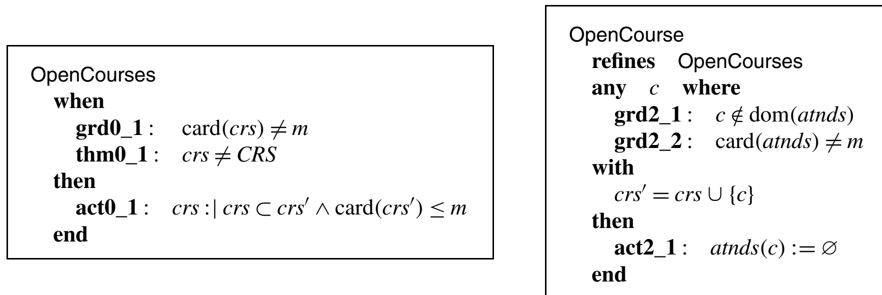
variables: $atnds$

invariants:
inv2_1 : $atnds \in CRS \rightarrow\!\!\!\!\!+ \mathbb{P}(PRTCPT)$
inv2_2 : $crs = \mathrm{dom}(atnds)$
inv2_3 : $\forall c \cdot c \in crs \Rightarrow prtcpts[\{c\}] = atnds(c)$
thm2_1 : $\mathrm{finite}(atnds)$

init
begin
$atnds := \varnothing$
end

We illustrate our data refinement by the following example. Assume that the available courses CRS are $\{c_1, c_2, c_3\}$, with c_1 and c_2 being opened, i.e., $crs = \{c_1, c_2\}$. Assume that c_1 has no participants, and p_1 and p_2 are attending c_2. Abstract variable $prtcpts$ hence contains two mappings as follows: $\{c_2 \mapsto p_1, c_2 \mapsto p_2\}$. The same information can be represented by the concrete variable $atnds$ as follows: $\{c_1 \mapsto \varnothing, c_2 \mapsto \{p_1, p_2\}\}$.

We refine the events using data refinement as follows. Event OpenCourses is refined by OpenCourse, where one course (instead of possibly several courses) is opened at a time. The course that is opening is represented by the concrete parameter c.



OpenCourses
when
grd0_1 : $\mathrm{card}(crs) \neq m$
thm0_1 : $crs \neq CRS$
then
act0_1 : $crs :| crs \subset crs' \wedge \mathrm{card}(crs') \leq m$
end

OpenCourse
refines OpenCourses
any c **where**
grd2_1 : $c \notin \mathrm{dom}(atnds)$
grd2_2 : $\mathrm{card}(atnds) \neq m$
with
$crs' = crs \cup \{c\}$
then
act2_1 : $atnds(c) := \varnothing$
end

The concrete guards ensure that c is a closed course and the number of opened courses ($\mathrm{card}(atnds)$) has not reached the limit m. The action of OpenCourse sets the initial participants for the newly opened course c to be the empty set. In order to prove the refinement relationship between OpenCourse and OpenCourses, we need to give the witness for the after value of the disappearing variable crs'. In this case, it is specified as $crs' = crs \cup \{c\}$, by adding the newly opened course c to the original set of opened courses crs.

Abstract event CloseCourses is refined by concrete event CloseCourse, where one course c (instead of possibly several courses cs) is closed at a time. The guard and action of concrete event CloseCourse are as expected.

CloseCourses
  status *anticipated*
  any cs **where**
    grd0_1 : $cs \subseteq crs$
    grd0_2 : $cs \neq \varnothing$
  then
    act0_1 : $crs := crs \setminus cs$
    act2 : $prtcpts := cs \lhd prtcpts$
  end

CloseCourse
  refines CloseCourses
  status *convergent*
  any c **where**
    grd2_1 : $c \in \mathrm{dom}(atnds)$
  with
    $cs = \{c\}$
  then
    act2_1 : $atnds := \{c\} \lhd atnds$
  end

We need to give the witness for the disappearing abstract parameter cs. It is specified straightforwardly as $cs = \{c\}$. Notice also that we change the convergence status of CloseCourse from *anticipated* to *convergent*. We use the following variant to prove that CloseCourse is convergent.

variant: $\mathrm{card}(atnds)$

The variant represents the number of mappings in $atnds$, and since it is a partial function, it is also the same as the number of elements in its domain, i.e. $\mathrm{card}(atnds) = \mathrm{card}(\mathrm{dom}(atnds))$. As a result, the variant represents the number of opened courses.

Event Register is refined as follows:[6] references to crs and $prtcpts$ in guard and action are replaced by references to $atnds$.

(abs_)Register
  any p, c **where**
    grd1_1 : $p \in PRTCPT$
    grd1_2 : $c \in crs$
    grd1_3 : $p \neq instrs(c)$
    grd1_4 : $c \mapsto p \notin prtcpts$
  then
    act1_1 : $prtcpts := prtcpts \cup \{c \mapsto p\}$
  end

(cnc_)Register
  refines Register
  any p, c **where**
    grd2_1 : $p \in PRTCPT$
    grd2_2 : $c \in \mathrm{dom}(attendees)$
    grd2_3 : $p \neq instrs(c)$
    grd2_4 : $p \notin atnds(c)$
  then
    act2_1 : $atnds(c) := atnds(c) \cup \{p\}$
  end

We now show some proof obligations for proving the refinement of m1 by m2.

OpenCourse/act0_1/SIM This proof obligation corresponds to rule SIM, ensuring that the action **act0_1** of abstract event OpenCourses can simulate the action of concrete event OpenCourse. Notice the use of the witness for crs' as a hypothesis in the obligation.

[6] We use prefixes (abs_) and (cnc_) to denote the abstract and concrete versions of the event, respectively.

$\ldots$ $atnds \in CRS \twoheadrightarrow \mathbb{P}(PRTCPT)$ $crs = \mathrm{dom}(attendees)$ $c \notin \mathrm{dom}(attendees)$ $\mathrm{card}(attendees) \neq m$ $crs' = crs \cup \{c\}$ $\vdash$ $crs \subset crs' \wedge \mathrm{card}(crs') \leq m$	OpenCourse/**act0_1**/SIM

CloseCourse/grd0_1/GRD This proof obligation corresponds to rule GRD, ensuring that the guard of concrete event CloseCourse is stronger than the abstract guard **grd0_1** of abstract event CloseCourses. Note the use of the witness for *cs* as a hypothesis in the obligation.

$\ldots$ $crs = \mathrm{dom}(atnds)$ $c \in \mathrm{dom}(atnds)$ $cs = \{c\}$ $\vdash$ $cs \subseteq crs$	CloseCourse/**grd0_1**/GRD

CloseCourse/NAT This proof obligation corresponds to rule NAT on page 220; it ensures that the variant is a natural number when CloseCourse is enabled.

$\ldots$ $\mathrm{finite}(atnds)$ $\vdash$ $\mathrm{card}(atnds) \in \mathbb{N}$	CloseCourse/NAT

CloseCourse/VAR This proof obligation corresponds to rule VAR on page 220; it ensures that the variant is strictly decreased by CloseCourse. The obligation is trivial since the variant represents the number of opened courses and CloseCourse closes one of them.

$\ldots$ $atnds \in CRS \twoheadrightarrow PRTCPT$ $c \in \mathrm{dom}(atnds)$ $\vdash$ $\mathrm{card}(\{c\} \lhd atnds) < \mathrm{card}(atnds)$	CloseCourse/VAR

A.8 Summary of the Development

The summary of the hierarchy of the development is illustrated in Fig. A.1.

Table A.3 summarises how assumptions and requirements are taken into account in our formal development. Note that the last refinement, m2, does not explicitly

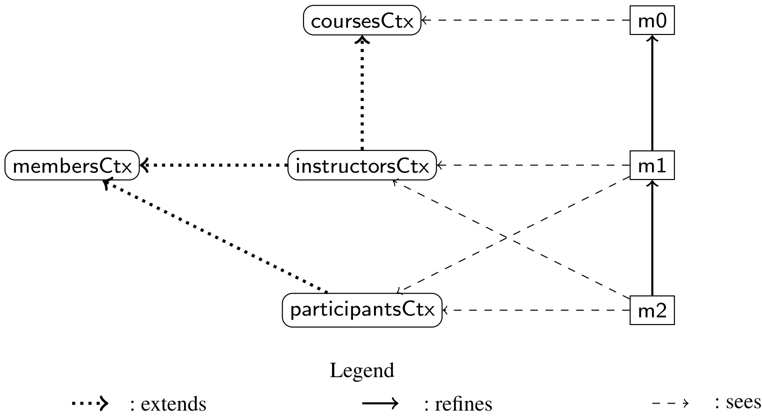


Fig. A.1 Development hierarchy

Table A.3 Requirements Tracing

Requirement	Models	Requirement	Models
ASM 1	instructorsCtx	REQ 6	m0
ASM 2	participantsCtx	REQ 7	m0
ASM 3	coursesCtx	REQ 8	m0
ASM 4	instructorsCtx	REQ 9	m1
REQ 5	m0	REQ 10	m1

take into account any requirements. Indeed, the requirements are implicitly *inherited* from the abstract machines through the refinement relationship.

The development is formalised and proved using the supporting Rodin platform[7] for Event-B [3] and is available online.[8] The summary of the proof statistics for the

Table A.4 Proof statistics

Constructs	Proof obligations	Automatic (%)	Manual (%)
coursesCtx	2	2 (100 %)	0 (0 %)
membersCtx	0	0 (N/A)	0 (N/A)
instructorsCtx	0	0 (N/A)	0 (N/A)
participantsCtx	1	1 (100 %)	0 (0 %)
m0	11	8 (73 %)	3 (27 %)
m1	14	13 (93 %)	1 (7 %)
m2	29	26 (90 %)	3 (10 %)
Total	57	50 (88 %)	7 (12 %)

[7] At the time of writing, we use Rodin version 2.4.0.

[8] http://deploy-eprints.ecs.soton.ac.uk/371/.

development is shown in Table A.4. Around 50 % of the proof obligations appear in m2, where we perform data refinement. Typically, data refinement involves radical changes to developments, since when replacing abstract variables with concrete variables, it is also necessary to adapt the events accordingly.

References

1. Abrial, J.-R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, Cambridge (1996)
2. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
3. Abrial, J.-R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: An open toolset for modelling and reasoning in Event-B. Int. J. Softw. Tools Technol. Transf. 12(6), 447–466 (2010)
4. Back, R.-J.: Refinement calculus II: Parallel and reactive programs. In: de Bakker, J.W., de Roever, W.P., Rozenberg, G. (eds.) Stepwise Refinement of Distributed Systems, Mook, The Netherlands, May 1989. Lecture Notes in Computer Science, vol. 430, pp. 67–93. Springer, Berlin (1990)
5. Chandy, K., Misra, J.: Parallel Program Design: A Foundation. Addison-Wesley, Reading (1989)
6. Lamport, L.: The temporal logic of actions. Trans. Program. Lang. Syst. 6(3), 872–923 (1994)
7. Mehta, F.: Proofs for the working engineer. PhD thesis, ETH Zurich (2008)
8. Schmalz, M.: The logic of Event-B. Technical Report 698, Institute of Information Security, ETH Zurich (October 2010). http://www.inf.ethz.ch/research/disstechreps/techreports/show?serial=698

Appendix B
Evidence-Based Assistance for the Adoption of Formal Methods in Industry

Jean-Christophe Deprez, Christophe Ponsard, and Renaud De Landtsheer

Abstract Managing change in industrial development processes is often a challenge. It involves many levels of the organisation: top management, who need to understand the benefits and impacts of change, project managers, who need to understand new processes and adapt their planning and monitoring practices, and engineers, who need to be trained in the use of new methods and tools. This is especially true for the introduction of Formal Methods (FM), which may significantly impact software development lifecycle processes. Industries that seek to adopt formal methods often need assistance in understanding their benefits, what formal methods can be used at what phase of the development lifecycle and what strategy should be used for a successful deployment of formal methods. During the DEPLOY project, an entire segment of work was dedicated to collecting important material to use as evidence, to provide answers to recurring concerns of companies wishing to investigate the use of formal methods. This appendix presents the overall approach used to collect and structure DEPLOY material as an evidence repository, notably, using success stories and Frequently Asked Questions (FAQs). These are fully published on www.fm4industry.org. A sample of success stories and FAQs answers are given, along with explanations on how companies in typical industry situations can navigate the evidence repository to find information relevant to their contexts. Finally, we also discuss how an industry engaged in an adoption process can set up its own internal evidence based on its specific context and how it can share its experience without compromising potentially confidential information.

J.-C. Deprez (✉) · C. Ponsard · R. De Landtsheer
CETIC, Rue des Frères Wright 29/3, Charleroi, Belgium
e-mail: jcd@cetic.be

C. Ponsard
e-mail: cp@cetic.be

R. De Landtsheer
e-mail: rdl@cetic.be

A. Romanovsky, M. Thomas (eds.), *Industrial Deployment of System Engineering Methods*, DOI 10.1007/978-3-642-33170-1,
© Springer-Verlag Berlin Heidelberg 2013

B.1 Introduction

Industry projects heavily rely on quantitative data when assessing project progress or product quality. However, in the early stage of transferring research results to industry, qualitative information can add very important value. A method based on measurement was proposed during the first year of the DEPLOY project, but it was rejected by Industry partners. Their main reasons for refusing an assessment based on measurements were confidentiality and a belief that numbers would not provide the right type of information to present a convincing argument for promoting formal methods within their organisation. In particular, DEPLOY industry partners recognised that quantitative methods are more appropriate for streamlined development projects, where a development approach is mastered by all project participants. On the other hand, in research transfer, where industry participants must understand new concepts, qualitative information is much more appropriate and convincing. DEPLOY industry partners also stressed the need to present evidence material in a form convenient for industry, such as well-structured Frequently Asked Questions (FAQs).

General surveys, such as that reported in [10] and a later extended one included in this book, confirm the need to provide evidence to support the adoption process by companies considering the use of Formal Methods (FM). **The goal of this appendix is to describe an approach to presenting and exploiting information related to FM adoption by industry**. The starting point was the concept of evidence. However, this needed to be further analysed in order to define how evidence could best provide information about formal methods to industry and in particular, about formal methods explored during the DEPLOY project. While one of the purposes for collecting evidence was to help industry partners of the DEPLOY project in their adoption process, the main target audience is organisations not involved in DEPLOY.

The sections of this appendix are organised as follows. First, a discussion elaborates on what constitutes evidence of formal method use in industry. From this, a general method for collecting and presenting evidence material is proposed. Second, different approaches are presented to searching for evidence material useful in one's context; in particular, different scenarios covering various industry sectors and different staff roles are presented. Then, some concrete exploitation scenarios of the material available online are highlighted. Finally, methods and tools for evidence collection and presentation within an organisation are proposed, including ways to be used by the organisation to share its experience with others.

B.2 A Method for Collecting and Presenting Evidence

Success stories have been shown to be an efficient way of collecting evidence of successful industry adoption. They often take the form of white papers. In some cases, they are published in industry tracks of conferences. Such publications rarely

Table B.1 A specific "success story" template

Name	Short meaningful identification to use as the title of the story.
Keywords	List of keywords helping in the classification of the success story.
Short description	Description (in a few lines) of what the story is about and what claims are supported.
Source	Organisation that contributed to the work of this story and a contextual description of the company's structure and research and innovation process.
Benefits	Positive impacts that can be inferred from the success story. (Note: Themes/questions from the FAQs highlight the important points on which impact are sought; hence one should review them when listing the benefits highlighted in the success story.)
Limitations	Limitations identified (not definitive if the story is ongoing): these are potential show stoppers and barriers to overcome.
Elaboration	Complete description of the work done to document the success story.
Consolidation	Additional arguments reported in the scientific literature corroborating the success story, including citations of scientific publications presenting these arguments.
Further work	Specific work still to be done by DEPLOY partners to make the story a success.
References	List of publications or reports related to the success story.
Related FAQs	List of questions in the FAQs to which this success story provided material.

allow sufficient space. Consequently, success stories often lack details of the very specific contexts in which they took place; for example, little information is usually provided about the overall innovation cycle of the enterprise involved in the success story, or on how researchers and production engineers collaborated during the transfer project.

In conclusion, traditional **success stories** are an interesting base for building an evidence repository of industry adoption, but they must be augmented to provide readers with additional contextual information as well as links to related efforts. Consequently, a specific "success story" template was designed to enhance and systematise the presentation of success stories emerging from the DEPLOY project. This template is presented in Table B.1.

The template above contains information on the context. In particular, the Source section suggests that an organisation should describe its structure and its research and innovation process. There can be large amounts of this context information. Hence it is recommended that the Source field should only provide the organisation's name and a link to another document that describes the full company context. This approach also has the added benefit that an enterprise with several success stories may often be linked to the same context document.

Success stories provide evidence of formal method transfer and usage in industry from a very specific viewpoint. By their very specific nature, success stories

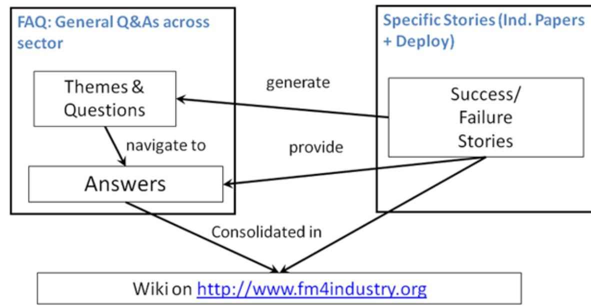


Fig. B.1 Organisation of the Evidence repository

do not, however, address high-level cross-concerns of various industry sectors or, at least, they do not present information in such a way that cross concerns can be easily deduced. This means that success stories should be complemented by another technique for presenting topics of general concern to many readers from various industries. During the DEPLOY project, researchers and industry partners proposed structuring this cross-cutting information in the **Frequently Asked Questions (FAQs) format**. Industry partners are accustomed to this type of format, where generic questions are answered in a fairly limited space. During DEPLOY, the average length targeted for answers was about one page as illustrated in the next section. The resulting structure of our evidence repository is shown in Fig. B.1.

At this point, it is necessary to identify a fairly exhaustive list of common concerns from different industry sectors. For this purpose, many documents provided by the DEPLOY industry partners as well as by the academic researchers and the tool providers were analysed. The high-level topics identified are:

- The impact on an organisation with regards to training scope and resourcing
- The impact on the software/system development process
  - The impact on the quality of product (and its work product) developed using formal methods,
  - The capability to exploit formal models at various stages of the development process,
  - The capability to perform reuse across development projects when formal methods are used, including reuse of formal and proven artefacts,
  - The capability to phase the learning of a formal method in an organisation and eventually limit the number of those who are expected to understand and become an expert in a formal method,
  - The capability to phase the migration to the formal method use incrementally (given the existence of products initially developed without using formal methods)
- Known strengths and weaknesses of tools associated with a formal method as well as the quality of support by tool providers
- External factors advocating take-up (from competition, standards bodies, laws) of formal methods
- General concerns related to formal methods in industry

These various themes validated by the DEPLOY industry and academic partners provide the initial categories for which the FAQs needs to be formulated.

Pieces of evidence are labelled **Industry Roles** to specifically target the relevant types of audience and increase the impact of answers. The organisations involved in DEPLOY were quite different, ranging from the technological SME with highly polyvalent roles to large companies with a specific R&D unit and very well-defined roles. Based on a careful study of the innovation cycle of the involved companies and on our experience in technology transfer, it was, however, possible to identify a set of key roles common to the DEPLOY partners, namely, high-level managers, project managers, engineers and technical analysts, and quality assurance staff. For a transfer project to be successful, many of these people must be involved in and supportive of the proposed transfer of research results. Thus, to obtain clear and concise answers, each FAQ should always explicitly target a single role. This does not necessarily mean that other roles will not be interested in the question, but rather that the named role is the primary target; hence the answer is provided to match this primary role. For instance, the wording of an answer targeted at engineers is different from that targeted at high-level management. Thus, if a high-level manager is interested in reading a FAQ answer primarily targeting engineers, he will understand that the answer is presented from the engineer's viewpoint. Making the targeted audience explicit by stating its roles definitely helps the reader.

In the context of transferring formal method engineering techniques to industry, the following roles were identified:

- High-Level Managers, who make strategic decisions and consider their financial impact
- Project and QA Managers, who supervise active users of FM (in production or R&D), plan projects and perform safety analysis and more traditional QA activities
- Engineers and Analysts, who actively use FM
- QA Practitioners, who must understand documents involving FM notation but do not need to develop the capabilities to produce them.

The FAQs approach suggests identifying questions of interest to each role for each theme. The important subtlety is that a question in the FAQs must be:

- Generic enough to interest enough readers and not be the sole concern of a single sector or, even worse, a single company.
- Specific enough so it is fairly easy to understand what results from an industry pilot should be proposed to what questions from the FAQs.

At the presentation level, the most adequate and commonly used medium is an Internet wiki, which gives wide access to the material. Wiki technology enables feedback from the people accessing the material, from comments to more active contributions, with a degree of control or moderation that can be adjusted. There are many Open Source wiki implementations, and we finally opted for Mediawiki [8], which is made very powerful by numerous extension plug-ins. Figure B.2 shows the homepage, accessible at http://www.fm4industry.org.

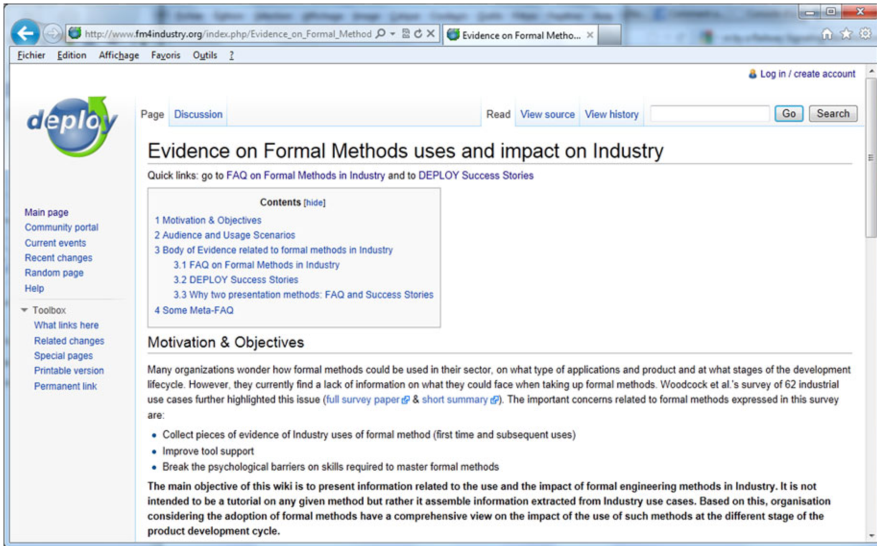


Fig. B.2 Organisation of the Evidence repository

B.3 Selected Highlights of FAQs and Success Stories

In this section we present selected FAQs and success stories to give some concrete and well-selected examples that will help the reader with the industrial background, interested in FM, to figure out what kind of information s/he can find and how it can help him/her. Those examples will serve to illustrate exploitation and production scenarios presented in the next sections, so the appendix is self-contained.

B.3.1 FAQ I: What Is the Position of Standards Regarding Formal Methods in My Industry Segment?

Theme: External Factor Pushing for Formal Method Adoption (ExFac)
Role: High-level Manager

B.3.1.1 Overview of Reference Standards for Safety-Critical Systems

The landscape of standards includes both generic and domain-specific standards. Figure B.3 shows a sectorial classification. A number of sectors have derived their standards from the generic IEC61508 standard while others have their own stand-alone standard.

- IEC 61511:2003 addresses industrial processes
- IEC 61513:2001 addresses the nuclear industry

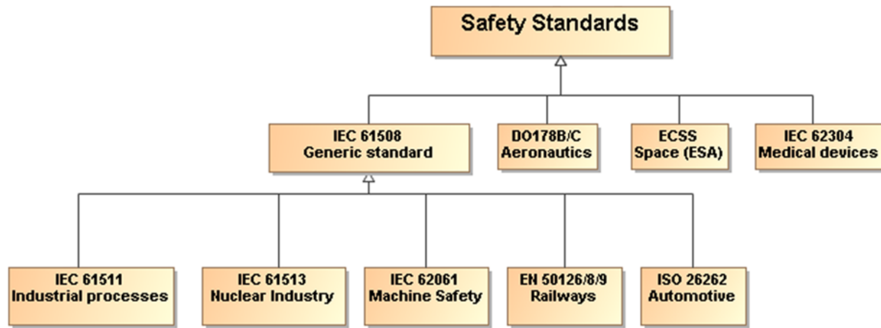


Fig. B.3 Sectorial classification of safety standards

- CE 62061:2005 addresses machine safety
- CENELEC/EN 50126/50128/50129 targets the railway sector
- ISO 26262:2011 addresses the automotive sector
- IEC 62304:2006 addresses the medical sector
- DO-178 addresses the aeronautic sector
- ECSS addresses the space sector (ESA)

In the aeronautic and space sectors, there are standards not directly linked to IEC61508.

- DO-178 (1992) is for the aeronautic sector
- ECSS is for the space sector

Figure B.3 shows a wide variety of standards. In addition to sector-specific constraints, there are also a number of other reasons for such a variety of standards, including historical (uncoordinated work, national level) and market reasons (market protection) [3]. Moreover, standards evolve, hopefully towards better integration, so they are revised (the publication year is often appended to their numerical identifier, as in EN 61508:2002) or made more specialised (as denoted by "B" in DO-178B).

Beyond this variety, standards exhibit common aspects:

- **definition of safety assurance levels:** all standards are based on a risk-oriented approach to their safety functions, classifying them into a number of dependability classes, typically by specifying PFD (Probability of Failure on Demand) and RRF (Risk Reduction Factor) figures. This classification varies across standards; for example, in IEC 61508, the classification ranges from SIL1, which is the least dependable, to SIL4, which is the most dependable. The DO-178B has a letter-based classification ranging from level "E", the least dependable, to level "A", which is the most dependable.
- **prescription level:** standards can be prescriptive (with an obligation to demonstrate compliance) or just give recommendations. However, in practice, deviating from recommendations can be difficult as this may require non-trivial work to justify it and to convince a certification body that is used to the well-established

recommended practices. In this way, the introduction of formal methods depends on whether they are simply mentioned, recommended or highly recommended.

- **scope:** this can address the software, the hardware or the complex system as a whole. It can also focus on specific parts of the development process (such as the development lifecycle, quality management, safety assessment, or system certification).

B.3.1.2 Standards and Formal Methods: An Overview

Generic IEC61508 Based on the SILs defined earlier, IEC 61508 classifies methodologies, techniques and activities as "not recommended", "recommended", "highly recommended" and so on. Among the "highly recommended" techniques for SIL4 are inspection and reviewing, using an independent test team and use of formal methods.

A single "highly recommended" technique is not enough to ensure the required safety, reliability and correctness: the standard requires a carefully chosen combination of appropriate techniques. Achieving this in a project requires a dedicated system engineering process as described in [6] to be defined.

Railway Sector The EN 50128 guidelines, issued by the European Committee for Electrotechnical Standardisation (CENELEC), address the development of "Software for Railway Control and Protection Systems", and constitute the main reference for railway signalling equipment manufacturers in Europe, with their use spreading to other continents and other sectors of the railway industry (as well as other safety-related industries).

In EN 50128, Formal Methods/Proofs are explicitly identified as relevant techniques/measures for software requirements specification, software architecture, software design, implementation, verification and testing and data preparation techniques. More precisely, they are "recommended" for SILs 1 and 2 and "Highly Recommended" for SILs 3 and 4. In particular, the Formal Methods cited include CCS, CSP, HOL, LOTOS, OBJ, Temporal Logic, VDM, Z and B. In the ongoing 2011 revision of the standard, additional constraints are put on tools, especially code/data generation tools, with respect to the need for a specification and for evidence that the implementations comply with it.

Despite this, and success stories such as that of the automated metro line 14 in Paris (METEOR) [2], formal methods have not spread across the entire railway signalling industry, where much software is still written and tested in traditional ways (with the testing effort usually adding up to more than 50 % of the development effort). This lack of adoption is due to the investments needed to build up a formal methods culture, and to the high costs of commercial support tools. Moreover, equipment can be made to conform to CENELEC without applying formal methods [1].

However, EN 50128 requires that the bug detection and bug fixing activities include reviews of early phases. This costs more in terms of time and money. Consequently, companies are becoming interested in applying formal methods at the specification and design phases as it is the only solution that transfers the effort to the design team.

Another barrier is that, although mentioned in the standard, the certification bodies might not be familiar with Formal Methods, as most of the systems they certify follow a test-based approach. The first time a certification body is asked to certify a system whose justification relies on such a formal argument, they might require additional information to be convinced. Typically, there needs to be an indication of the level of expertise that an auditor should possess in order to perform certification for a company which bases its development on formal methods. Once certification is acquired, each new system can easily follow the same path. Siemens has gone through this process with the B-method, and it is now accepted by the certification bodies, so that subsequent projects have become easier to certify. In the specific market of metro lines, the B-method has even become the standard required by the market.

Aeronautic Sector DO-178B was published a while ago, in 1992. It does not recommend or propose a specific development process or methodology. The certification approach is to demonstrate compliance of the process and produced artefacts with a set of goals related to a number of lifecycle-related processes (planning, development, V&V, configuration and so on). As mentioned above, DO-178B ranks the software category by five dependability classes: from A (most dependable: catastrophic effect) to E (least dependable: no effect). The scope of the application of formal methods is the verification of software artefacts. In DO-178B, verification can also be achieved though a combination of review, analysis and testing activities.

With regard to Formal Methods, DO-178B categorises them as "alternative methods" because, at the time the document was produced (1992), they were assessed as inadequately mature. However, they can be used "as long as they can be demonstrated to address the goals of the standard, and their usage is adequately planned and described" [9].

A new revision DO-178C was published in January 2012. It is explicitly addressing formal methods to complement dynamic testing; they can be used selectively or as the primary source of evidence.

Automotive Sector The sector-specific standard is ISO 26262 (derived from IEC 61508). It recommends formal methods, but semi-formal methods are highly recommended. In such ISO standards, many development methods can be "recommended", but only one can be "highly recommended" (above all others). At the beginning of the debate on ISO 26262, it was suggested that formal methods should be highly recommended. After intensive lobbying from large automotive companies, which are convinced that formal methods will incur a significant cost overhead, semi-formal methods have been given the "highly recommended" status, while formal methods are just recommended.

B.3.1.3 Conclusion: Standards, Formal Methods and Possible Strategies

Some standards recommend or even highly recommend formal methods, but very few standards describe how to manage formal development. It is therefore difficult to prove that the development process complies with the standards. Certification authorities need to be convinced about this issue.

As formal methods are less widespread and certification authorities are less familiar with them than with the classical development methods, there is more work to be done than with other methods, especially the first time, even though formal methods can provide better arguments. However, the investment can be worth the effort, as shown by the Siemens case for B.

B.3.2 FAQ II: Are the Tools Associated with a Particular Formalism Backed by Responsive, Robust and Enduring Organisations?

Theme: Known Strengths and Weaknesses of Tools and Tool Providers (TOOL)
Role: High-Level Manager

B.3.2.1 Is There Guarantee of Long-Term Tool Availability and Support?

Industry projects may last tens of years from the development to the decommissioning of a system. It is therefore crucial for industry to ensure proper support throughout the complete project lifetime, including its retirement. Tools can be distributed under Open Source or Proprietary Licences. Each model comes with its own risk of disappearance (bankruptcy for proprietary code, community dissolution for Open Source). Given the niche market, securing the support is a nontrivial task (e.g., escrow for proprietary code, direct community involvement or support for Open Source).

B.3.2.2 Is the Tool Reliable?

Closed source reliability is a matter of trust that can be provided by a certification scheme for example. Concerns have been raised about whether Open Source tools can achieve higher reliability [4]. However, the large number of industrial-strength open source tools available nowadays, e.g. PVS and NuSMV suggests that they can. One reason for this is that they will potentially be peer-reviewed by many; another is that distributed development requires better defined and carefully designed interfaces at the design level. Furthermore, extensive test suites are often available for Open Source tools.

B.3.2.3 Is the Tool Scalable?

The ability to scale up depends on different factors. Tool-induced limitations may be due to the underlying formal technology, implementation problems (e.g., some bottleneck in a processing chain) or simply usability (e.g., limitations in managing large pieces of models). To assess scalability, references, feedback and reviews provide initial information that is useful for completely ruling out tools inadequate for industry. A second step is to challenge the tool on realistic case studies in various industry sectors, as the way models are built can also impact the ability to scale up. Open Source tools may be more likely to be unable to scale up, especially if they are still at the R&D stage. However, there are also highly scalable Open Source tools in the FM area (e.g., SPIN and NuSMV model-checkers, ACL2 and Isabelle theorem provers).

B.3.2.4 Is the Tool Usable?

It is important that tools facilitate various tasks when building or modifying a model, carrying out validation and verification activities, supporting teamwork, and so on. Commercial tools generally have better usability because special attention is devoted to this aspect, whereas Open Source tools tend to focus more on core functionality and efficiency, sometimes with only a command line interface.

B.3.2.5 Does the Tool Integrate Well in Industry Tool Chains?

The ability to integrate into existing industrial tool chains is fundamental. This requires that well-documented data formats are defined, and that APIs and binaries on specific OSs are available. This is an area where Open Source tool developers usually outperforms proprietary ones. Furthermore, Open Source developers often adopt open, standard data formats. By contrast, intense competition frequently pushes proprietary tool developers to keep internal data formats hidden.

B.3.2.6 What Is the Impact of My Tool Regarding Certification?

Using a formal tool in the design flow (i.e., at design time) can have an impact on the certification process, especially if the tool is generating production artefacts such as source code for systems requiring higher integrity levels. Evidence of correctness of the output produced by these tools has to be provided by various means: redundant implementation, extensive test coverage, and specific verification activities. Another supporting success story, the ProB tool used by Siemens and developed by the University of Düsseldorf is undergoing a qualification process for the railways EN-50128 standard.

B.3.3 Success Story I: Productivity Improvement of Data Consistency in Transportation Models

Table B.2 summarises a success story from the transportation sector.

Table B.2 Productivity improvement of data consistency in transportation models

Name	Productivity Improvement of Data Consistency in Transportation Models.
Keywords	B-Methods, assumption validation, model-checking, ProB tool.
Short description	The correct and safe behaviour of Metro models developed by Siemens relies on assumptions made about the topology of each specific site. Those assumptions have to be validated against a number of properties required of the model. Until now this task was achieved through custom rules in Atelier B, which was not very efficient at proving them automatically: it required a lot of manual work, about one person-month for a typical metro project. As an alternative to a proof-based approach, model checking with the ProB model-checker was successfully applied to this task. A dramatic productivity improvement was experienced on a real project (San Juan metro line): the same faults could be discovered completely automatically and within a few minutes of execution time, whereas the previous approach required about one man-month of effort.
Source	Siemens, University of Düsseldorf.
Benefits	• Dramatic productivity improvement for the problem of data validation • Better diagnostics of failed property using output
Limitations	• Specific to transportation domain; extensibility to other sectors to be studied • The ProB Tool must undergo qualification/certification (ongoing)
Elaboration	The success story is fully described in the paper [7] and its qualification report is an internal deliverable of the DEPLOY project. Below we highlight the main arguments and lessons learned: • **Full automation and quick response time.** Checking the properties takes 4.15 seconds, and checking the assertions takes 1,017.7 seconds (i.e., roughly 17 minutes) using ProB 1.3.0-final.4 on a MacBook Pro with a 2.33 GHz Core2 Duo. Manually it takes a month for one person to solve the same problem (San Juan line). • **Easier diagnostics.** Another advantage of model checking over proof is that it provides a counterexample that will be of greater help in understanding the cause than the information found in a failed proof. Furthermore, a specific visualisation tool was developed to make it easier to understand a counterexample. • **Confidence in the results.** An important issue is the qualification of the tool in order to have full confidence in the results produced. With respect to this, the model-checking approach implemented has some form of redundancy: a property is expressed in a positive and a negative form. The produced result should be consistent; otherwise, this indicates a problem. The problem could be located at lower levels; in fact, two bugs were discovered in the Prolog interpreter. The absence of problem is, however, no guarantee of correctness and does not remove the need for tool validation.

Table B.2 (Continued)

Elaboration	• **Qualification process.** In order to be used as a correctness argument in the development process of a qualified product, the tool must be qualified itself. Without such qualification, the tool might be used as a debugger is used by developers, to speed up development, and to conduct manual verification only when the developer is confident that the verification will succeed. The ProB tool has undergone a qualification process. The qualification argument is mainly based on testing various parts of the tool, and on implementing internal sanity checking mechanisms to check that the tool is consistent with itself. Two testing approaches were deployed in parallel: global end-to-end testing of industrial cases and unit testing of specific internal modules. Part of unit testing involved automatically generated test cases. Furthermore, an independent tool was used to double-check the results of some unit tests. An additional lesson for academics is the need to cover the full language used in industry while prototyping. As industry will not normally adapt its models to suit an academic tool, it is necessary to consider extending the tool at the industrialisation phase. The amount of work involved should be evaluated early enough in the R&D process. The ProB tool is already a success as Siemens plans to use it to replace Atelier B for formal data validation. This would require the validation of ProB for SIL4 use, which is being investigated.
Consolidation	Model checking has been tried in the past in the railways domain to validate some interlocking properties. Specific properties (robustness and locality) allowed optimisation of the general CTL algorithm. The case was only validated on a small railway interlocking involving two stations [5]. A CSP approach was also investigated together with the FDR model-checker [Wint02]. Useful counterexamples related to safety requirements could be produced using some examples. However, a conclusion was reached that languages based on process algebra, such as CSP, were not adequate for modelling this domain (especially for the control table). They resulted in models that were difficult for practitioners to understand or validate.
Related FAQs	**of Interest to Project and QA Managers** • **QI-PQAM-2:** How is the productivity of the various stages of the system development cycle affected when formal engineering methods are used? • **ExFac-HM-1:** What is the position of standards regarding formal methods in my industry segment (e.g., are they enforcing them, highly recommending, etc.)? **of Interest to Engineers and Analysts:** • **TOOL-EA-2** Do tools automate all tedious tasks?

B.4 Typical Scenarios Exploiting Shared Evidence Material

A few customary situations faced by professionals with different roles in industrial software-intensive system development are presented, along with the kinds of specific questions they often have about formal methods and their usage in actual in-

dustrial development. Subsequently, a brief explanation shows how they can browse the evidence repository to find elements of answers to their questions.

Scenario I I am a high-level manager who has heard that others in the sector have successfully used formal methods. My staff has no prior experience of them. My question is: "How should I proceed in the evidence repository to better understand if FM could be used in my company, too?"

Scenario II I am an engineer in an SME. Our development team has successfully used advanced static analysis tools during the debugging and testing phases of product development. However, in a recent project, important requirements remained implicit until the customer validation phase. Subsequently, a significant overhaul of the architecture was needed to handle these. Static analysis was clearly no help. My question is: "How should I proceed in the evidence repository to better understand how our development team could start using formal methods earlier in the development lifecycle to increase the chances of identifying all important requirements after the design phase?"

Scenario III I am a QA engineer. I saw a presentation by another company at a conference, which showed QA and safety engineers from other fields using tools to generate test cases for various kinds of coverage. My question is: "Does your repository contain information on test generation formalisms, approaches and tools and their eventual applicability to my sector?"

Scenario IV I am a product (line) manager. I have heard of several success stories on the use of formal methods but my company already has well-tested components that fulfil our quality requirements. Our project mostly consists in configuring these existing components and integrating them. My question is: "Does your evidence repository demonstrate the use of formal methods to ease component integration, even if these components were not developed using formal methods?"

These situations traditionally faced by industry illustrate how one can navigate the evidence repository to find FAQs and success stories containing information relevant to one's context. For each of the scenarios above, a suggested navigation approach will be explained. In general, there are only two ways to start navigating the evidence repository. The top-down approach starts from the FAQs while the bottom-up one starts from success stories. Most readers will come up with a general question and therefore find the top-down approach to be more appropriate. However, in some cases, one may learn about a specific success story at a conference or while reading an article, for example, in Scenario IV. In this case, it may then be more

appropriate to look for this or a similar success story on the evidence repository wiki. Below, the two navigation approaches are briefly discussed and then applied to each of the above scenarios.

I—Top-Down Navigation Approach When one has a general question about formal methods in specific organisational context, it is assumed that this is a rather non-focused exploratory search. In this case, the most optimal way of looking for relevant information is as follows.

- Identifying the themes in the FAQs that are relevant to the question.
- For each theme of interest, reviewing the questions specific to one's role to identify a subset of relevant questions.
- While reading answers, identifying pointers to specific reference papers and success stories; then deepening the search by reading success stories in the evidence repository or the referenced scientific literature.

It is important to note that questions are sometimes worded to highlight broad cross-industry concerns. Such questions can be answered from quite different viewpoints, taking into account different sectors, different types of formal methods and even different phases of the lifecycle. Thus, when first reading the FAQ, readers must understand that the answer may focus not only on their topic of interest. This broad coverage for some questions is provided deliberately, so that the reader will find more general answers than was initially anticipated. Nevertheless, readers will be free concentrate on a particular part of the answer or to explore other information provided in the answer. An answer is normally subdivided into subsections to make this easier: for example, the answer that considers differences in the use of formal methods at different phases of the development lifecycle will be explicitly broken down with one subsection per phase of the lifecycle.

In other words, when initiating a search from the FAQs, the reader must keep an open mind and not search for direct answers to only his/her context. Obtaining tailored answers would require specific studies that can only be achieved through a dedicated subcontract that goes far beyond the evidence repository.

II—Bottom-Up Navigation Approach The reader may start an investigation by considering a success story that closely relates to his/her specific need. Although s/he may be tempted to stop the investigation at that point and to start implementing the approach described in the success story, appropriate caution should be taken. First, his/her specific context is likely not to match the context in which the success story developed. Second, continuing the investigation may highlight new possibilities not initially anticipated by the reader. It is therefore worth spending more time navigating from the success story to its context description. To search for a success story and identify those potentially similar to one read or heard about elsewhere, a review is needed of the complete title list of success stories, which are currently organised by industry sectors. At the moment, the list is still short enough for full browsing to work; it will be subclassified when the number of success stories exceeds 30 or 40. Success stories in the evidence repository include cross-references

to FAQs where they are cited. It is then possible to navigate to an FAQ linked to the success story and identify other potential questions of interest with the same FAQ theme. Finally, success stories are classified by sectors; thus, it also makes sense to review the success stories related to his/her sector.

Scenario I In this scenario, a high-level manager starts with a very broad enquiry. Thus, the manager should use a top-down approach for navigating the evidence repository, starting with the selection of themes. His/her initial concern relates to the lack of current expertise on formal methods. Consequently, the whole theme on training would provide relevant information. Furthermore, the high-level manager should be informed about the different means for including formal methods in the development lifecycle. In particular, it may be possible that the organisation's context makes it possible to restrict the application of formal methods to a small team of experts rather than having to train the entire engineering staff. In addition, early in an enquiry, the high-level manager is likely to be interested in the general topic theme. Thus, in this context, the high-level manager would be very interested in the following FAQ themes and questions:

- Impact on an organisation with regard to training scope and resourcing
  - **TSP-HM-1** What is the cost or effort needed to train engineers/analysts to use a new formalism given their previous experience with formal engineering methods?
- Understanding the impact on the Software/System Development Process The capability of phasing the learning of a formal method in an organisation and eventually of limiting the number of those who must understand and become an expert in a formal method
  - **CIF-HM-1** Is it possible to phase the learning and use of a formal method? (In other words, can the introduction of a formal method be done gradually or must the entire development staff understand (and master) the formal method at once?)
  - **CIF-HM-2** How do organisational procedures used in various system development lifecycle processes need to be adapted when formal methods are introduced?
  - **CIF-PQAM-1** Can the use of a formal method be hidden from the majority of development and management teams, except for a few selected experts who will use it (perhaps even without other team members knowing)?
  - **CIF-PQAM-2** What are the risks of hiding the use of formal methods and what are the strategies for mitigating them?

- The capability of phasing the migration to the use of a formal method (given the existence of products initially developed without using formal methods)
  - **MF-HM-1** Is it possible to iteratively migrate an existing system to the formal method use?
- General topics of concerns related to formal methods in industry
  - **G-HM-3** What are formal methods?
  - **G-HM-1** What are the main benefits and risks of using formal methods in product development?
  - **G-HM-2** Why have formal methods failed to achieve a breakthrough in the market for such a long time?

Scenario II In this scenario, an engineer states that the development team already has some initial expertise in using advanced static code analysis tools, which usually perform formal verification on the code at the implementation and test phases. These tools help with verification (Am I building the product right?) but not with validation (Am I building the right product?). The engineer therefore wonders how using formal methods at the requirement and design phase could help. Although the engineer's concern seems quite specific, it is still appropriate to perform top-down navigation. From the engineer's perspective, it seems clear that the overall theme of "Understanding the impact on the Software/System Development Process" is of interest. While s/he may then be drawn to other themes, in the list below we identify only themes and questions related to his initial interest.

- The impact on the quality of product (and its work product) developed using formal methods,
  - **QI-PQAM-1** What impact does the use of formal engineering methods have on identifying issues at each phase of the development cycle? (Note: while reading the answer to this FAQ, the engineer will find a link to a success story on how formalising requirements helped improve their quality, through identifying ambiguity in informal requirements among other things)
  - **QI-PQAM-2** How is the productivity of the various stages of the system development cycle affected when formal engineering methods are used (as compared to when non-formal development methods are used)?
  - **QI-EA-1/** How have various formal methods (and modelling patterns in these formal methods) developed to increase various quality attributes

of systems/products such as safety, reliability, guarantees regarding real-time properties, concurrency, availability and maintainability?

- The capability to exploit formal models at various stages of the development process,
  - **EM-EA-3** Is there evidence of formal methods being used to improve the quality of requirements and design documents expressed in less formal notations such as UML?
  - **EM-PQAM-2** Does the use of a formal engineering model have any beneficial effect on requirements and design traceability?
  - **EM-PQAM-3** Is there any guidance on the cost/benefit trade-off in using different validation techniques (testing, model checking, proof) between formal and non-formal methods as well as between formal methods?
  - **EM-EA-2** What formal methods offer modelling languages, tools or theories suited to a particular domain, which may be a vertical (Automotive) or a horizontal domain(System Security)?
  - **EM-QAP-1** Does the use of formal engineering methods help in designing tests?
  - **EM-QAP-2** Does the use of formal engineering methods help provide test oracles?
- The capability to phase the learning of a formal method in an organisation and eventually to limit the number of those who must understand and become an expert in a formal method
  - **CIF-HM-1** Is it possible to phase the learning and use of a formal method? (In other words, can the introduction of a formal method be done gradually or must the entire development staff understand (and master) the formal method at once?)
  - **CIF-HM-2** How do organisational procedures used in various system development lifecycle processes need to be adapted when formal methods are introduced?
  - **CIF-PQAM-1** Can the use of a formal method be hidden from the majority of development and management teams except for a few selected experts who will use it (perhaps even without other team members knowing)?
  - **CIF-PQAM-2** What are the risks of hiding the use of formal methods and what are the strategies for mitigating them?
  - **CIF-EA-1** How far can the use of a formal method be automated?
- The capability to phase the migration to the use of a formal method (given the existence of products initially developed without using formal methods)
  - **MF-PQAM-1** Does an iterative migration to the use of a formal method on an existing system require complete rephrasing of requirements, reengineering of models, rewriting of test plans and so on?

– **MF-EA-1** Is there an efficient staged approach to introducing the use of a formal method iteratively on an existing system (e.g., is there a preferred order of introducing the use of several formal methods?)?

According to the FAQs, a significant proportion of the questions above concern high-level and project managers. However, in this case the questions are highlighted as being "of interest" to an engineer. This should not cause a problem. Indeed, the classification based on roles indicated that the question will usually be of interest to a given role; hence, the answer is adapted to that role. This means that, for example, an answer targeted at a high-level manager will not include technical content. In other words, an engineer reading a question/answer entry for a high-level manager or even a project manager should understand that the answer is provided from the viewpoint of a manager.

Scenario III In this scenario, the QA engineer starts from a particular success story. Hence, this is one of the rare cases where bottom-up navigation is probably preferable. The best case scenario is when the success story is already part of the evidence repository. In this case, the safety engineer can simply read it in the wiki and follow the links to relevant FAQs and their answers. However, if the success story has not appeared in the evidence repository yet, the engineer is first advised to send a email to the fm4industry editorial board to indicate that an interesting success story has been published. Second, the engineer can then browse the current list of success stories, possibly to find a success story similar to the one read or heard about. In this case, after a quick review of various story titles, the QA engineer would discover that there is one which presents the benefits of model-based testing automation. A pointer makes it possible to navigate from the success story to a specific FAQ. Subsequently, if interested, the engineer can continue with the bottom-up browsing of the FAQs.

Success story of Interest:

- In the Business Information Domain—Benefits of Model-Based Test Automation

From this success story, a link to the following FAQs is provided:

- **EM-QAP-1** Is it possible to take advantage of formal models to automate a part of the QA tasks?

Subsequently, the QA engineers may find other FAQs in the same category "Exploiting Models (EM)" interesting:

- **EM-QAP-2** Does the use of formal engineering methods help provide test oracles?
- **EM-PQAM-3** Is there any guidance on the cost/benefit trade-off in using different validation techniques between various formal methods as well as between formal and non-formal methods?

After having read the complete material, the QA engineer should feel more confident about the applicability of formal methods in his/her organisation. S/he may then be interested in reading additional FAQs to better understand how to proceed, in particular, FAQs on training and on controlling the deployment of formal methods in an organisation.

Scenario IV In this scenario, a product line manager indicates that his/her company will not switch to an approach involving a complete reformulation of specifications using formal methods, but the organisation is willing to investigate the use of formal methods for specifying the integration of existing components. Although the manager mentions success stories, they are not specific; hence s/he should probably initiate his/her search using the top-down navigation. Unfortunately, FAQs are not worded so as to make it easy for the manager to directly map the themes to his/her concern with the use of formal methods for integration. However, by analysing the themes a little deeper, the manager would probably become interested in the themes "Control Impact of Formalism (CIF)", "Migration to a Formalism (MF)" and possibly "Exploiting Models (EM)".

Reading the various questions in these 3 themes, the manager could probably identify the following FAQs as potentially pertinent:

- Question ID: MF-PQAM-1. Does an iterative migration to the use of a formal method on an existing system require complete rephrasing of requirements, reengineering of models, rewriting of test plans and so on?
- Question ID: MF-HM-1. Is it possible to migrate an existing system to the use of a formal method iteratively?
- Question ID: CIF-HM-1. Is it possible to phase the learning and the use of a formal method? (In other words, can the introduction of a formal method be done gradually or must the entire development staff understand (and master) the formal method at once?)

Although none of the questions above are directly related to the manager's concern, the fact that they mention iterative migration and gradually increasing learning indicates that formal methods do not need to be applied to the entire system at once. Thus, the manager may wonder if certain answers address the topic of formal specification of system integration as one of the ways of migrating to the use of formal

methods. Unfortunately, none of these questions have yet been answered in the literature or in the course of DEPLOY.

In such a situation, as a last resort, the manager would be advised to browse success stories to identify any similarities with his concerns. As a result, the manager may find that "Adoption Eased by Using the Formal Models Behind Domain Specific Notations" could somehow provide guidance, since the integration language can be viewed as a specific notation for system integration purposes.

The success story does not provide a perfect answer to the manager but seems nonetheless to indicate that a high-level language, in this example, a Business Process Modelling language, can be used and extended to perform formal verification. Although it requires an open mind and a leap in thinking, the manager could draw a parallel between two high-level languages: the one in the success story is used for expressing a Business Process, while in his case, this kind of language is used for system integration. If the manager can make this observation, he should realise that it may be possible to attach a formal method to the traditional system integration language used at the company. Consequently, the success story provides a hint to the manager that a formal method can potentially be used for specifying system integration. Furthermore, it may be possible to hide the use of the formalism from system integrators, who will continue to use the system integration languages they know.

Making such a leap in reasoning may not be possible for the manager. In this case, it is in his/her direct interest to communicate with the fm4industry board, who can guide readers and help them correctly interpret the material in the evidence repository.

B.5 Setting Up Private Evidence Collection and Selective Public Sharing

Setting up an evidence repository for a research project such as DEPLOY is useful for a company, even if this is different from what it may normally aim to do. In particular, for an organisation to make a sound decision on its innovation cycle, it is important to spend time sorting and normalising information collected from trials. As pointed out at the beginning of this appendix, information on innovation should also be qualitative rather than uniquely quantitative. The approach used in the course of DEPLOY to obtain the needed information is based on the FAQs and the enhanced success stories previously described. These two instruments provide appropriate means to gather information for any research transfer project. However, this entails having to identify relevant themes and questions to the target topic. In this section, we assume that an organisation may be willing to roll out an evidence repository as part of its engagement with formal methods. In this case it can directly use the FAQ themes and questions from DEPLOY.

Below are the important aspects of creating an evidence repository:

- Having a duplicate evidence repository, with one of them kept private for storing work and progress and the other one used to publish for the target audience (for example, to make information available to a few departments, a whole enterprise or even the world)
- Nominating a trusted editorial board to vouch for the independence and the overall quality of the information provided. It may even be possible to require a particular turnover of the editorial board over time (e.g., by new nominations or elections)
- Controlling the editing of published evidence material
- Facilitating commenting and obtaining feedback on work in progress as well as on the published evidence material
- Allocating the necessary resources for evidence collection, sorting and transformation

During DEPLOY, two distinct wiki (based on a Redmine forge) were used to address the various aspects above. The first one requires a login and a password for read and write privileges, and was useful for keeping potentially confidential information under control. The second one requires appropriate credentials for write authorisation, but is viewable by the public. Material was moved from one to the other in a controlled way. For an even easier setup, the Mediawiki tool is now used for the final public repository. It provides the appropriate features for controlling read/write access, and it also includes a discussion board with message threading for commenting and providing feedback.

Clearly, organisations will initially prefer to keep most information private at the organisation or at an even more restricted level. However, once the information has matured, it is usual for well-crafted pieces of evidence to be used as advertising. For example, publishing information about training can indirectly show competitors that an organisation is promoting the adoption of formal methods, whereas publishing data on exploiting formal models for various purposes and on reusing formal models will certainly display a growing expertise. This is exactly the type of information sought for publication on the www.fm4industry.org site. However, the caveat is that the information must be validated and approved by the editorial board of fm4industry, which incidentally may evolve over time. The board's basic validation principle is as follows: the presentation of a peer-reviewed article at a workshop or a conference is sufficient guarantee for acceptance. It is nonetheless possible that other information may meet sufficient standards to be accepted; for example, a white paper published by an organisation as part of an EU research project supported by an academic partner may be accepted for publication.

The main goal of fm4industry.org is to promote the use of formal methods in industry and become an almost self-sustainable site to which external parties would make contributions of their own will. However, to guarantee the validity and quality of information, a minimal threshold for acceptance of new publications must be enforced.

References

1. Bacherini, S., Fantechi, A., Tempestini, M., Zingoni, N.: A story about formal methods adoption by a railway signaling manufacturer. In: Proceedings of the 14th International Symposium on Formal Methods, Hamilton, Canada, August 21–27, 2006, pp. 179–189. Springer, Berlin (2006)
2. Behm, P., Benoit, P., Faivre, A., Meynadier, J.M.: Meteor: A successful application of B in a large project. In: Proceedings of Formal Methods 1999. Lecture Notes in Computer Science, vol. 1708, pp. 369–387. Springer, Berlin (1999)
3. Bozzano, M., Villafiorita, A.: Design and Safety Assessment of Critical Systems. CRC Press (Taylor and Francis), an Auerbach Book, Boca Raton (2010)
4. Craigen, D.: Formal methods adoption: What's working, what's not! In: Proceedings of the 5th and 6th International SPIN Workshops on Theoretical and Practical Aspects of SPIN Model Checking, pp. 77–91. Springer, London (1999)
5. Eisner, C.: Using symbolic CTL model checking to verify the railway stations of Hoorn-Kersenboogerd and Heerhugowaard. Int. J. Softw. Tools Technol. Transf. 4(1), 107–124 (2002)
6. Kars, P.: Formal methods in the design of a storm surge barrier control system. In: Lectures on Embedded Systems, European Educational Forum, School on Embedded Systems, pp. 353–367. Springer, London (1998)
7. Leuschel, M., Falampin, J., Fritz, F., Plagge, D.: Automated property verification for large scale B models. In: Proceedings of the 2nd World Congress on Formal Methods, FM '09, pp. 708–723. Springer, Berlin (2009)
8. Mediawiki. http://www.mediawiki.org
9. RTCA/DO-178B. Software considerations in airborne systems and equipment certification (1992)
10. Woodcock, J., Larsen, P.G., Bicarregui, J., Fitzgerald, J.S.: Formal methods: Practice and experience. ACM Comput. Surv. **41**(4), 1–36 (2009)