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

DEPLOY Deliverable D52

D15.5 Final Dissemination/Exploitation Report

Thierry Lecomte (ClearSy)
Alexander Romanovsky (Newcastle)
Michael Butler (Southampton)
Elena Troubitsyna (Aabo Akademi)

Public Document

April 30, 2012

http://www.deploy-project.eu

Contents

1 Introduction

This document reports the project achievements in dissemination and exploitation. It is updated every year, completed with progress made, and delivered at month M12 (D14), M24 (D27), M36 (D37), and M48 (D52). This particular document is the final (M48) report.

2 Main Achievements in Dissemination and Exploitation

This chapter presents DEPLOY fourth year dissemination and exploitation achievements.

2.1 DEPLOY Interest Group (DIG)

The DEPLOY Interest Group (DIG) is a community that has been of a paramount importance for the project, as its members have specifically declared their interest and support. Hence the overall dissemination/exploitation activity has been centred around the DEPLOY Interest Group, gathering companies, universities, and individuals interested in the Rodin platform. The DIG has had a privileged access to information, such as bi- annual newsletter, dedicated hands-on sessions, etc.

DIG members may:
- join the group. A simple (electronic) letter of intent is sufficient. Joining the DIG is free of charge;
- provide feedback on the platform and related plug-ins, by using the platform and sharing experience and expectation;
- provide complementary case studies and examples covering similar or new application domains;
- attend dedicated trainings and hands-on sessions, organized specifically for the DIG upon request.

Special attention was given to the DIG members: dedicated means were allocated to help the DIG members to get educated and experienced with the Rodin tools.

To increase membership in the DIG, our strategy has been threefold:

- invite Rodin project followers to join the DIG,
- send personal invitations to join,
- promote the DIG at each dissemination event.

This has been coordinated with the organization of industrial days, local actions of partners, etc., when possible. Communication has been ensured by a dedicated mailing list, a newsletter, and industry days. DIG members have been personally invited to all our dissemination events.

In order to populate the DIG with relevant users, we have initiated a survey ("We need to know who you are !") where people have the opportunity to register to the DIG and to the newsletter as well. More than 200 answers have been collected by the end of the project, indicating that the typical user is from academia, working on Windows and doing research with Rodin.

Current DEPLOY Interest Group members is composed of 67 members.

2.2 DEPLOY Associates

The DEPLOY Associates (*DAs*) is a group created late 2009, gathering privileged industrial experimenters of the DEPLOY tools and methodology. The main goal of this group has been to ensure a broad dissemination of the results of the project (tools, methodology, documents, etc.) by:

- experimenting with new case-studies, possibly from domains not yet addressed by the DEPLOY project
- ensuring that adequate training is delivered to the *DA* personnel in charge of the case-study, in order to obtain comparable results among *DAs*
- collecting feedback (metrics, models, conclusions, etc.) from *DA*, in order to improve project deliverables and to demonstrate the extent to which they are applicable to industry.

The DEPLOY Associates have received specific and dedicated help from the DEPLOY project (training, consultancy, etc.).

Three DEPLOY Associates were been selected and invited to work with the project:

Automação E Systémas – Sao Paulo (Brazil)
AeS is a SME specialized in the design and development of embedded systems. Outside the project, AeS and ClearSy are collaborating on the deployment of platform screen-door control/command systems in the metro of Sao Paulo. AeS has investigated the use of Rodin for safety-critical systems through a number of case studies:

- a methodological WRSPM approach to a B formalization in an industrial setting;
- LADDER to B;
- using the B formal method in the process of traditional software development for critical systems;
- a UML-based method for Event-B refinement;

- changing the way to vital verification.

Critical Software Technologies – Southampton (U.K.)
CST, part of the Critical Software Group, is specialized in the development, verification and validation of software. As a DEPLOY Associate, CST applied the Rodin tools and methods for the verification and validation of avionics and satellite software. Following an on-site training delivered by Southampton in 2010, CST performed the requirements analysis phase through an integrated flight secondary display case study.

XMOS ltd – Bristol (U.K.)
XMOS is a "fabless" semiconductor company that develops multi-core, multi-threaded processors targeted at embedded systems markets. Some concepts found in XMOS technology are part of the transputer technology developed by that company in the 1980s. XMOS processor technology is general-purpose and has therefore been exploited in a range of different markets, including audio, display, communications, robotics and amateur innovation. Event-B and Rodin based techniques have been applied for Instruction Set Architecture (ISA) analysis, by constructing a formal model of the ISA of the XCore microprocessor.

Appendix A provides a comprehensive report on the activities and achievements of the DAs.

2.3 Events

DEPLOY results were presented at several occasions, listed in the table below.

July 15th - 16th 2011 Bangalore (India). Workshop Refinement Based Development of Software Systems using Event-B
September 12nd – 15th 2011 Newcastle (UK). 11th International Workshop on Automated Verification of Critical Systems (AVoCS)
September 22nd 2011. Naples (Italy). Conference SAFECOMP-2011
September 28th – 29th 2011 Brussels (Belgium). Conference "Internet of Services 2011: Collaboration meeting for FP7 projects"
September 29th– 30th 2011 Geneva (Switzerland). SERENE 2011: 3rd International Workshop on Software Engineering for Resilient Systems ICFEM 2011: 13th International Conference on Formal Engineering Methods 26th—28th October 2011, Durham, United Kingdom
November 28th 2011 Tokyo (Japan). 56th GRACE Seminar on Advanced Software Science and Engineering
February 7th – 9th 2012Bristol (UK). 20th Safety-critical Systems Symposium
February 27th – March 1st 2012 Fontainebleau (France). DEPLOY Federated Event

Below we provide more information for the most important ones.

Workshop Refinement Based of Software Systems using Event- B
(Bangalore, 15-16 July 2011)
Members of the DEPLOY Project made a major contribution to this workshop in

Bangalore with the main speakers being Michael Butler and Colin Snook from the University of Southampton. This workshop provided an intensive introduction to Event-B and the Rodin platform. It covered modelling, refinement and proof in Event-B and UML-B. Other speakers were S. Ramesh and Manoranjan Satpathy from General Motors R&D Bangalore. Several representatives from Indian industry attended including GM, nuclear and space industries as well as several Indian Universities. The workshop was organised jointly by IISC Bangalore (Deepak D'Souza) and GM R&D (P. Sampath, M. Sathpathy and S. Ramesh).

AVoCS 2011 *(Newcastle, 12-14 September, 2011)*
11th International Workshop on Automated Verification of Critical Systems was organized with the help of DEPLOY and co-chaired by DEPLOY members from Newcastle and Dusseldorf. The PC included many project members, the topics of the workshop were directly related to the topics of the project (http://conferences.ncl.ac.uk/AVoCS2011/). Seven papers from DEPLOY were accepted and presented. The proceedings of this event are submitted to the EC as project deliverable D51.

DEPLOY Federated Event *(Fontainebleau, France, 27 February – 1 March 2012)*
The DEPLOY Federated event, hosted by IUT Sénart-Fontainebleau with the help of the LACL laboratory, took place in the south of Paris. It was the occasion to demonstrate a complete picture of the current status of the Rodin platform, the on-going research and industrial use, inside and outside the DEPLOY project. This was the final dissemination event of DEPLOY. On each day of this event we had about 74-78 participants.

The first day was devoted to tutorials on various Rodin plugins and their use, and covered the following topics: the theory plugin, integration of external provers and solvers in Rodin, customisation of proof tactics, adding plugins into ProB. The morning consisted of presentations on each of these topics, while the afternoon consisted of several hands-on small group master-class sessions in parallel.

The second and third days were devoted to workshop-style talks on experiences with Rodin usage and on new plug-ins for Rodin. There were 30 presentations from the DEPLOY partners and outside of DEPLOY. There was plenty of lively interaction between participants during and between the talks. This has been a successful event that demonstrated that the community is active and growing.

The fourth day, the Industry Day, was devoted to reporting on DEPLOY achievements and to discussing the industrial use of Rodin and Event-B. We had talks from the deployment partners, the three DAs, 2 talks from ClearSy and Systerel about their use of Rodin in industrial projects outside DEPLOY, more presentations from DEPLOY about the project objectives and outcomes and on evidence, and about a new company Formal Minds recently created in Dusseldorf.

The full programme of the DEPLOY Federated Event is included as an appendix (Appendix B) to this report. The Rodin workshop proceedings and Industry Day slides are available online:

http://wiki.event-b.org/index.php/Rodin_Workshop_2012
http://www.bmethod.com/php/federated-event-2012-en.php

2012 Safety Critical Systems Symposium *(Bristol, UK, 7-9 February 2012)*
DEPLOY provided an exhibition stand on the Rodin tools at the 2012 Safety Critical Systems Symposium. Delegates were provided with a memory stick containing an installation of Rodin along with tutorial material. There was strong interest in Rodin from industrialists working on the safety-critical domain.

The following event is planned after DEPLOY: Michael Butler will be teaching at the 2012 Marktoberdorf Summer School on Engineering Dependable Software Systems: http://asimod.in.tum.de/ The lectures will be on "Abstraction, Refinement and Decomposition for Systems Engineering". The lectures will address the key role played by formal modelling and verification in systems engineering. The lectures will use the Event-B formal modelling language and the associated Rodin toolset for Event-B.

2.4 Electronic Dissemination

All materials related to DEPLOY and the Rodin platform are made electronically available:

- Platform and plug-ins source code
- Project deliverables, papers, and manuals
- Teaching material
- Models (including case-study description)

Websites. These are the websites are related to the DEPLOY project:

- the official project site, hosted by ClearSy and reachable at http://www.deploy- project.eu. It contains useful information about the project, its objectives. This site nicely integrates three other websites, hosted by Southampton University
- the DEPLOY repository (http://deploy-eprints.ecs.soton.ac.uk/), containing all the project deliverables, publications, tutorials, models, etc. External stakeholders are invited to contribute to the DEPLOY repository.
- the Event B site (http://www.event-b.org/) gathering information on the Rodin platform and its plugins.
- The wiki website (http://wiki.event-b.org), providing documentation for users and developers of the Rodin toolset.
- the developer site, hosted by sourceforge and reachable at http://rodin-b-sharp.sourceforge.net/.
- A repository of evidence for adopting FMs in industry (http://www.fm4industry.org) has been developed in the project for providing answers to recurring concerns of companies wanting to investigate the usage of formal methods. It initially included the evidence collected in DEPLOY.

Publications. The following articles (the publication list is on the website) having been published in 2011-12 and are available on the publications website:

1. Ponsard, Christophe and Deprez, Jean-Christophe and Delandtsheer, Renaud (2012) Is my Formal Method Tool Ready for the Industry? In: AVOCS 2011, 12-14 September 2011, Newcastle, UK.

2. Dinca, Ionut and Ipate, Florentin and Mierla, Laurentiu and Stefanescu, Alin (2012) Learn and Test for Event-B - a Rodin Plugin. In: ABZ'12 Conference, June 19-21, 2012, Pisa, Italy. (In Press)

3. Diaconescu, Denisa and Leustean, Ioana and Petre, Luigia and Sere, Kaisa and Stefanescu, Gheorghe (2012) Refinement-Preserving Translation from Event-B to Register-Voice Interactive Systems. In: 9th International Conference on Integrated Formal Methods (iFM'12), June 19-21, 2012, Pisa. (In Press)

4. Edmunds, Andrew and Rezazadeh, Abdolbaghi and Butler, Michael (2012) Formal modelling for Ada implementations: Tasking Event-B. In: Ada-Europe 2012: 17th International Conference on Reliable Software Technologies, Stockholm. (In Press).

5. Deprez, Jean-Christophe and Ponsard, Christophe (2012) An Collaborative FAQ Approach for Collecting Evidence on Formal Method Industrial Usage. In: DEPLOY Federated Event (Industry Day).

6. Ponsard, Christophe and Flamand, Jacques and Deprez, Jean-Christophe (2012) Assessment of the Evolution of the Rodin Open Source platform. In: Rodin User and Developer Workshop 2012, 28-28 February 2012, Fontainebleau.

7. Ipate, Florentin and Dinca, Ionut and Stefanescu, Alin (2012) Model learning and test generation using cover automata. IEEE Transactions on Software

Engineering . (Submitted)

8. Hayes, Ian J. and Burns, Alan and Dongol, Brijesh and Jones, Cliff B. (2012) Comparing Models of Nondeterministic Expression Evaluation. The Computer Journal, (Submitted)

9. Hayes, Ian J. and Jones, Cliff B. and Colvin, Robert J. (2012) Refining rely-guarantee thinking. Formal Aspects of Computing, (Submitted)

10. Iliasov, Alexei (2012) Augmenting formal development with use case reasoning. In: Proc. of the 17th International Conference on Reliable Software Technologies (Ada-Europe 2012). Stockholm, Sweden. June 11- 15, 2012. Springer.

11. Dinca, Ionut (2011) Multi-Objective Test Suite Optimization for Event-B Models. In: ICIEIS'11, Springer CCIS Series vol. 251 , 11-14 November 2011, Malaysia.

12. Lopatkin, Ilya and Iliasov, Alexei and Romanovsky, Alexander and Prokhorova, Yuliya and Troubitsyna, Elena (2011) Patterns for Representing FMEA in Formal Specification of Control Systems. In: The 13th IEEE International High Assurance Systems Engineering Symposium, Boca Raton, FL, November 10-12, 2011, Boca Raton, USA.

13. Varpaaniemi, Kimmo (2011) 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). Documentation. Space Systems Finland Ltd. (Unpublished)

14. Bryans, Jeremy W. (2011) Developing a Consensus Algorithm Using Stepwise Refinement. Proceedings of the 13th international conference on Formal methods and software engineering, LNCS (6991). pp. 553-568.

15. Bendisposto, Jens and Jones, Cliff and Leuschel, Michael and Romanovsky, Alexander (2011) Proceedings of the 11th workshop on Automated Verification of Critical Systems. Newcastle University.

16. Edmunds, Andrew and Rezazadeh, Abdolbaghi and Butler, Michael (2011) From Event-B Models to Code: Sensing, Actuating, and the Environment. In: SBMF2011, Sept 2011, Sao Paulo, Brazil.

17. Hayes, Ian J. and Burns, Alan and Dongol, Brijesh and Jones, Cliff B. (2011) Comparing Models of Nondeterministic Expression Evaluation. Technical Report. School of Computing Science, University of Newcastle.

18. Tarasyuk, Anton and Troubitsyna, Elena and Laibinis, Linas (2011) Quantitative Verification of System Safety in Event-B. In: SERENE 2011 Lecture Notes in Computer Science 6968, Springer, 2011.

19. Deprez, Jean-Christophe and Ponsard, Christophe and Fitzgerald, John S. (2011) A FAQ Approach for Collecting Evidence on Formal Method Industrial Usage. In: FM2011 / Industry Day, 20-24 June 2011, Limerick (Ireland).

20. Ponsard, Christophe and Deprez, Jean-Christophe (2011) Collaborative Building of an Open Evidence Repository to Drive the Adoption of Formal Engineering Methods. In: FM2011, 20-24 June 2011, Limerick (Ireland).

21. Boström, Pontus and Degerlund, Fredrik and Sere, Kaisa and Waldén, Marina (2011) Concurrent Scheduling of Event-B Models. In: 15th International Refinement Workshop (associated with Formal Methods 2011), 20th June 2011, Limerick, Ireland.

22. Ponsard, Christophe and Devroey, Xavier (2011) Generating High-Level Event-B System Models from KAOS Requirements Models. In: InforSID 2011, 24-26 May 2011, Lille (France).

23. Bryans, Jeremy W. and Fitzgerald, John S. and McCutcheon, Tom (2011) Refinement-based techniques in the analysis of information flow policies for dynamic virtual organisations. In: PRO-VE 2011 - 12th IFIP Working Conference on VIRTUAL ENTERPRISES, 17-19 October, Sao Paulo, Brazil. (Submitted)

24. Stefanescu, Alin and Ipate, Florentin and Lefticaru, Raluca and Tudose, Cristina (2011) Towards Search-based Testing for Event-B Models. In: 4th International Workshop on Search-Based Software Testing, 21 Mar 2011, Berlin.

25. Edmunds, Andrew and Butler, Michael (2011) Tasking Event-B: An Extension to Event-B for Generating Concurrent Code. In: PLACES 2011. (In Press)

26. Edmunds, Andrew and Butler, Michael (2011) Tasking Event-B: An Extension to Event-B for Generating Concurrent Code. In: PLACES 2011.

27. Fathabadi, Asieh Salehi and Rezazadeh, Abdolbaghi and Butler, Michael (2011) Applying Atomicity and Model Decomposition to a Space Craft System in Event-B. In: Third NASA FORMAL METHODS SYMPOSIUM. (In Press)

28. Silva, Renato and Pascal, Carine and Hoang, Thai Son and Butler, Michael (2011) Decomposition Tool for Event-B. Software: Practice and Experience, 41 (2). pp. 199-208.

29. Dinca, Ionut and Stefanescu, Alin and Ipate, Florentin and Lefticaru, Raluca and Tudose, Cristina (2011) Test Data Generation for Event-B Models using Genetic Algorithms. In: 2nd International Conference on Software Engineering and Computer Systems (ICSECS'11), June 27-29, 2011, Malaysia.

30. Grotsev, Denis and Iliasov, Alexei and Romanovsky, Alexander (2011) Formal Stepwise Development of Scalable and Reliable Multiagent Systems. In: Dependability and Computer Engineering: Concepts for Software-Intensive Systems. IGI Global. ISBN ISBN13: 9781609607470

31. Iliasov, Alexei (2011) Generation of certifiably correct programs from formal models. In: 1st Int. Workshop on Software Certification. At the 22nd Int. Symposium on Software Reliability Engineering (ISSRE 2011), November 30, 2011, Hiroshima, Japan.

32. Iliasov, Alexei (2011) Use case scenarios as verification conditions: Event-B/Flow approach. In: Software Engineering for Resilient Systems, Proc. of 3rd International Workshop. September 29-30, 2011 Geneva, Switzerland. LNCS (6968). Springer, pp. 9-23. ISBN 978-3-642-24123-9

33. Iliasov, Alexei and Laibinis, Linas and Troubitsyna, Elena and Romanovsky, Alexander (2011) Correct-by-Construction Development of Fault Tolerant Systems (Tutorial at FM 2011). [Teaching Resource]

34. Iliasov, Alexei and Laibinis, Linas and Troubitsyna, Elena and Romanovsky, Alexander (2011) Formal Derivation of a Distributed Program in Event B. In: Proc of ICFEM 2011: 13th International Conference on Formal Engineering Methods. 26th—28th October 2011, Durham, United Kingdom. Springer.

35. Iliasov, Alexei and Romanovsky, Alexander (2011) Scaling Event-B to industrial applications: the role of formal design decomposition (Industry Paper). In: 22nd IEEE International Symposium on Software Reliability Engineering, Nov 29 - Dec 2, 2011, Hiroshima, Japan.

36. Jastram, Michael (2011) ProR - Eine Softwareplattform fur Requirements Engineering. Softwaretechnik-Trends, 31 (1).

37. Jastram, Michael and Graf, Andreas (2011) Requirement Traceability in Topcased with the Requirements Interchange Format (RIF/ReqIF). In: FIRST TOPCASED DAYS TOULOUSE 2011, 2 Feb - 4 Feb 2011, Toulouse, France.

38. Jastram, Michael and Graf, Andreas (2011) Requirements Modeling Framework. Eclipse Magazin, 6.11 .

39. Jastram, Michael and Graf, Andreas (2011) Requirements, Traceability and DSLs in Eclipse with the Requirements Interchange Format (RIF/ReqIF). Dagstuhl-Workshop MBEES: Model-Based Development of Embedded Systems. fortiss GmbH, München, Germany.

40. Jones, Cliff B. and Pierce, Ken G. (2011) Elucidating concurrent algorithms via layers of abstraction and reification. Formal Aspects of Computing, 23 (3). pp. 289-306.

41. Laibinis, Linas and Troubitsyna, Elena and Iliasov, Alexei and Romanovsky, Alexander (2011) Formal Approach to Ensuring Interoperability of Mobile Agents. In: Handbook of Research on Mobile Software Engineering: Design Implementation and Emergent Applications. IGI Global. ISBN ISBN13: 9781615206551

42. Lopatkin, Ilya and Iliasov, Alexei and Romanovsky, Alexander (2011) Rigorous Development of Dependable Systems using Fault Tolerance Views. In: The 22nd annual International Symposium on Software Reliability Engineering (ISSRE 2011), Nov 29 - Dec 2, 2011, Hiroshima, Japan.

43. Mazzara, Manuel (2011) On Methods for the Formal Specification of Fault Tolerant Systems. In: DEPEND 2011.

44. Mazzara, Manuel and Dragoni, Nicola and Zhou, Mu (2011) Dependable Workflow Reconfiguration in WS-BPEL. In: NODES 2011.

45. Mazzara, Manuel and Marraffa, Antonio and Biselli, Luca and Chiarabini, Luca (2011) Polidoxa: a Sinergic Approach of a Social Network and a Search Engine to Offer Trustworthy News. In: INTRUSO 2011.

46. Schmalz, Matthias (2011) Term Rewriting in Logics of Partial Functions. Proceedings of ICFEM 2011 . (In Press)

47. Su, Wen and Abrial, Jean-Raymond and Huang, Runlei and Zhu, Huibiao (2011) From Requirements to Development: Methodology and Example. In: ICFEM 2011, Durham, UK. (In Press)

48. Tarasyuk, Anton and Troubitsyna, Elena and Laibinis, Linas (2011) Quantitative Reasoning about Dependability in Event-B: Probabilistic Model Checking Approach. In: Dependability and Computer Engineering: Concepts for Software-Intensive Systems. IGI GLobal.

49. Yeganefard, Sanaz and Butler, Michael (2011) Structuring functional requirements of control systems to facilitate refinement-based formalisation. In special issue: Automated Verification of Critical Systems 2011, Electronic Communications of the EASST, 46.

50. Prokhorova, Yuliya, Laibinis, Linas, Troubitsyna, Elena, Varpaaniemi, Kimmo, Latvala, Timo (2011). Derivation and Formal Verification of a Mode Logic for Layered Control Systems. In: Tran Dan Thu, Karl Leung (Eds.), *Proceedings of the 18th Asia-Pacific Software Engineering Conference (APSEC 2011)*, 49-56, IEEE Conference Publishing Services (CPS), December 2011.

51. Tarasyuk, Anton, Troubitsyna, Elena and Laibinis, Linas (2012). Formal Modelling and Verification of Service-Oriented Systems in Probabilistic Event-B. In Proc. of *International Conference on Integrated Formal Methods, IFM 2012*, Lecture Notes for Computer Science, Springer, June 2012.

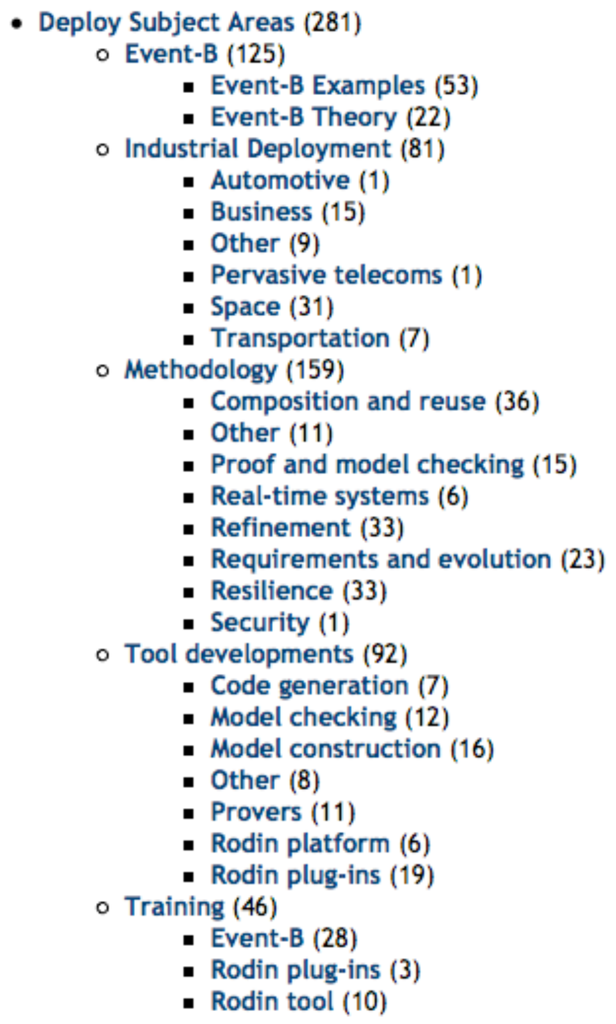
52.Pereverzeva, Inna, Troubitsyna, Elena and Laibinis, Linas (2012). Formal Goal-Oriented Development of Resilient MAS in Event-B. In Proc. of *Ada-Europe 2012 -- 17th International Conference on Reliable Software Technologies*. Lecture Notes in Computer Science, Springer, June 2012.

53.Pereverzeva, Inna, Troubitsyna, Elena and Laibinis, Linas (2012). Formal Development of Critical Multi-Agent Systems: A Refinement Approach. In Proc. of *European Dependable Computing Conference*, May 2012. IEEE Computer Press.

54. Yeganefard, Sanaz and Butler, Michael (2012) Control systems: phenomena and structuring functional requirement documents. In 17th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2012). Paris, FR, 18 - 20 July 2012.

55. Iliasov, Alexei and Troubitsyna, Elena and Laibinis, Linas and Romanovsky, Alexander and Varpaaniemi, Kimmo and Ilic, Dubravka and Latvala, Timo (2012) *Developing Mode-Rich Satellite Software by Refinement in Event-B*. Science of Computer Programming. Accepted. In press.

The DEPLOY repository. The DEPLOY repository is composed of several subject areas (event-B language, industrial deployment, methodology, tool developments, and training). A snapshot of the resources currently available is given below:



- Deploy Subject Areas (281)
  - Event-B (125)
    - Event-B Examples (53)
    - Event-B Theory (22)
  - Industrial Deployment (81)
    - Automotive (1)
    - Business (15)
    - Other (9)
    - Pervasive telecoms (1)
    - Space (31)
    - Transportation (7)
  - Methodology (159)
    - Composition and reuse (36)
    - Other (11)
    - Proof and model checking (15)
    - Real-time systems (6)
    - Refinement (33)
    - Requirements and evolution (23)
    - Resilience (33)
    - Security (1)
  - Tool developments (92)
    - Code generation (7)
    - Model checking (12)
    - Model construction (16)
    - Other (8)
    - Provers (11)
    - Rodin platform (6)
    - Rodin plug-ins (19)
  - Training (46)
    - Event-B (28)
    - Rodin plug-ins (3)
    - Rodin tool (10)

Metrics. Statistics are collected in the project to evaluate Rodin platform and the DEPLOY project's popularity. The measurement of DEPLOY websites hits from foreign IP addresses will provide an estimate of the awareness and the interest concerning DEPLOY in both the industry and academic worlds. Reverse links are used to improve our Google score, thus improving our visibility on the Net.

DEPLOY websites statistics (number of monthly unique visits) are given below (for the 48 months of the project):

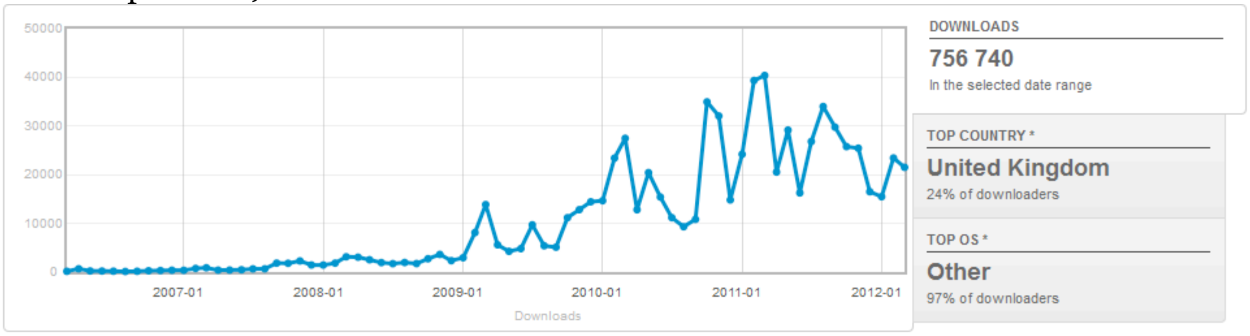
	01	02	03	04	05	06	07	08	09	10	11	12
Publications	62	383	359	410	411	364	N/A	N/A	N/A	N/A	N/A	N/A
Event-B.org	38	144	477	600	486	488	488	489	554	586	654	373
Wiki Event-B.org	0	0	12	11	10	146	384	722	540	927	1483	827
Deploy-project.eu	2586	4741	5936	6365	7741	7770	8618	5988	3164	3561	4338	4262

	13	14	15	16	17	18	19	20	21	22	23	24
Publications	1592	803	940	871	827	748	826	852	1010	1117	1059	919
Event-B.org	756	763	1102	839	795	750	887	631	765	939	968	850
Wiki Event-B.org	1326	1470	1979	1577	1543	1198	1565	1340	1609	1728	1702	1654
Deploy-project.eu	4698	4375	4703	4012	4436	4733	5474	4493	5240	6492	6576	6078

	25	26	27	28	29	30	31	32	33	34	35	36
Publications	1007	863	922	496	320	636	743	638	869	1038	1164	938
Event-B.org	907	955	1188	891	1005	897	1019	912	1029	1112	1015	904
Wiki Event-B.org	1652	1540	1865	1483	1666	1502	1483	1525	1936	2131	2083	1837
Deploy-project.eu	9955	8649	10547	10948	10087	10729	14240	11484	13958	11302	10693	11358

	37	38	39	40	41	42	43	44	45	46	47	48
Publications	1123	1234	840	575	609	511	539	537	707	793	727	651
Event-B.org	1136	1428	1378	1176	980	806	768	770	1038	1281	1142	1070
Wiki Event-B.org	2321	2656	4212	5499	3717	2102	2709	1923	2521	2797	2775	2576
Deploy-project.eu	11812	13598	14683	32770	3700	28468	2888	36516	29756	41661	61776	48674

Sourceforge statistics (number of downloads for all files, since the beginning of the Rodin platform) are listed below:



DOWNLOADS
756 740
In the selected date range

TOP COUNTRY *
United Kingdom
24% of downloaders

TOP OS *
Other
97% of downloaders

Sourceforge is not providing any more detailed statistics. So it is not possible to distinguish operating systems and files (platform, plugins).

Platform exploitation. A company, called Rodin Tools Ltd (http://www.rodintools.org), has been created recently as a Not for Profit Company taking over responsibility for the Rodin toolset at the end of DEPLOY. The Company consists of:

- a Strategy Committee of external advisers to look at 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.

A dedicated workshop was organized in February 2012, during the DEPLOY Federated Event, aimed at provided support to external developers. A one-day tutorial was set up at that occasion.

Newsletter. DEPLOY publishes a newsletter every 6 months, providing a clear view on:

- what is going on in the project,
- what its current status is, and
- what are the next steps.

All WPs are contributing to the newsletter, which is sent to persons having registered on the website (200 so far). All issues are archived on the website and can be downloaded anonymously.

Project brochure. A leaflet, presenting the project, was created at the beginning of the project and has by now been distributed at many conferences attended by DEPLOY partners.

Training materials. In relation with WP10 Technology Transfer, teaching material including:

- tutorials,
- large examples, entirely loaded on the platform, accompanied by extensive explanations are available to the community, targeting practitioners (engineers, etc.), teachers, researchers, etc. through the DEPLOY publications website.

New resources made available during 2011 are:

- Varpaaniemi, Kimmo (2011) Event-B Projects DSAOCSSv002 and DSAOCSSV003 with Special Files for ProB Classic. [Rodin Archive]
- Iliasov, Alexei and Laibinis, Linas and Troubitsyna, Elena and Romanovsky, Alexander (2011) Correct-by-Construction Development of Fault Tolerant Systems (Tutorial at FM 2011). [Teaching Resource]

The Rodin User's Handbook (http://handbook.event-b.org) has been released in addition to the Event-B wiki (http://wiki.event-b.org). The story behind this book is as follows. The executive team of the DEPLOY project recognized the importance of good documentation of the tools for succeeding is reaching a wider audience. In a meeting at ETH in 2010, the team established, amongst other things, that "it is clear that the current documentation would not provide sufficient guidance, say, for an engineer in an automotive company to start using the tools without significant support". It then commissioned the creation of better Rodin documentation. This handbook is the result of this effort.

2.5 Collaboration with ICT SSAI&E projects

DEPLOY sets up co-operation activities with other ICT projects under the WP2007/2008 objective "Service and Software Architectures, Infrastructure and Engineering", in order to exploit synergies between other projects and to increase the impact of the ICT initiative.

This topic is covered by the "Collaboration Plan" deliverable – see deliverable D50.

Appendix A. Reports from the DEPLOY Associates

This Appendix includes three chapters written for the Springer book to be published at the end of the project (*Industrial deployment of system engineering methods providing high dependability and productivity. A. Romanovsky, M. Thomas (Eds). Springer. 2012).*

These three chapters provide a comprehensive description of the work carried out by the DAs.

Chapter 7
Formal Methods as an Improvement Tool

Aryldo G Russo Jr

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

7.1 History

AeS Group is a Brazilian company, created in 1991. At that time it was working in the building automation field. In 1998, AeS became involved in the railway field, when the first Brazilian General Door Control System (GDC) for Rolling stock doors was developed. The aim of this equipment was to be an interface between train requests (operator request, signalling requests, etc.) and the door system itself. This system was safety critical, as the major operation of this equipment was to send an open command to the doors in each cabin. From that time, several different safety critical or safety related systems were developed by AeS, such as, ”dead man control”, parts of brake system, speed limit control, etc.

Due to the advances in technology, many safety functions that were handled by hardware are now the responsibility of embedded software. Some standards can 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 (higher level, higher safety), the so called Safety Integrity Levels - SIL, and above level 2, the use of formal methods is required or suggested to achieve a certain level of completeness, robustness, and safety, that grows as the level grows. The goal of using formal methods is to

Aryldo G Russo Jr
AeS Group, R Domingos Barbieri 298 Brazil, e-mail: agrj@aes.com.br

produce an unambiguous and consistent specification that is as complete, error-free and with as few contradictions as possible, and which is simple to verify.

In 2006, AeS began studying Formal Methods to decide which method to use, where to apply, how to apply, etc. later on, 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

Before going through the studies, a brief description of the AeS structure, projects, team, etc. is needed to introduce a context of our applications.

- Our Research and Development team is composed, basically, from 10 engineers, none of them with a formal method background, and one R&D manager (an engineer with some formal background)
- During the time of the DEPLOY project the team received a whole week training covering the Rodin environment, refinement and UML-B.
- as AeS is inserted 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 application that requires higher levels of safety (SIL3 and SIL4)

7.3 Case studies

This section presents a chronological description of AeS's case studies, trying to show what was the problem, what we have applied and what were the results. 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 at the point we thought they would be more effective, avoiding extra costs and efforts. Nevertheless, we think that at some time in the future a completely formal development process might be used and evaluated in comparison with the traditional approach. This will be our final case study but, unfortunately it was not finished at the time of writing this chapter.

7.3.1 *Early 2007 - The B Method and railway domain - breaking the wall*

During our search of formal methods (that began in 2006), we discovered out that the B method had gain 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 about the subject and no introductory guide was available. Digging deeper, we discovered that, at the beginning of 2007, in Besancon, France, a conference about B would occur. From the industry point of view, conferences are the place where you go to learn how to use tools, techniques, methods, etc. so, that would 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. It was also, our first contact with people committed to using the B/Event-B methods. It was the first contact with the Rodin tool. It was this conference that led to us becoming a DEPLOY associate partner, and to be using successfully 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 to do it manually, but studying the formalization process and the notion of abstraction and refinement helped to change the way of thinking and to improve the understanding of the specification.

As a result of this first small attempt, I can report two small (in terms of size) but important achievements:

- we found natural language specification inconsistencies: we formalized a small portion of a door system specification, and, at that point our objective was to formalize exactly what was written in that specification. In doing that we reached a point where the formal abstractions could not be proved, and analysing the 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 formalize the existing specification of our pilot project. A small system that should stop the train movement in case of human problems with the operator. this was a single sheet only specification, and as in the former case, we formalized exactly what we had and we discovered that there were a lot of missing information. This led us to create several assumptions that had to be validated with the customer specialist to be able to finish the specification.

At the end, in both attempts, we realized that we could use formalization 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 reflect the industrial needs, I prepared a brief comparison of some existing tools. I have restricted this comparison to some tools that I knew better and that have been used in my application field, that is, railways applications. Those tools are: Atelier B, Rodin and SCADE. This study was performed 3 years ago, so differences may be found if it were performed again.

7.3.3.1 Methodology

The ”Oracle” I used to determine the classification of each tool in each category was my personal feeling since a more detailed research was not performed so far, but even so, in the last 3 years I could collect some comments from people I have been training, so I hope it may be helpful.

It is also prudent to state the maturity differences among these tools. While SCADE and Atelier B have been in the market for a long time, Rodin was about to be released in its first official version (version 1.0) which means that the first two have already received much feedback from their industrial users helping them to change the directions when users were not satisfied (in the Atelier B case, after a lot of complaints about the user interface, the developers completely changed the GUI), while the last one had no time yet to receive or to implement completely such feedbacks.

The comparison methodology was based on three aspects, as follows:

- *capability:* how these tools can satisfy project constraints
- *usability:* the difficulty the user faces when trying to use the tool
- *adaptation to the current development process :* I mean, how well the tool fits in the process without causing too many changes to the way it was performed so far

To make a classification of these aspects I used a simple ranking method, as follows:

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

I present the results in Table 7.1.

7.3.3.2 Chart comparison

To justify those results I can say that:

- Atelier B

Aspect	**capability**	**usability**	**adaptation**	***Results***
Atelier B	2	1	2	*5*
Rodin	2	2	1	*5*
SCADE	2	3	3	*8*

Table 7.1 Comparsion table

- the capability to solve the project constraints is not so bad, but you do need to know a lot of the formal language and constructs to be able to have easy proof obligations.
- although, the version 4 of Atelier B supplies much better usability, all comments I have so far are based on the previous version where the lack of a good User Interface made its usage 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 during the middle phases (refinements) are some times, painful, it can not receive the higher grade.

• Rodin

- as it is not so different from Atelier B, similar results are seen. The capability to solve the project constraints is not so bad, but you do need to know a lot of the formal language and constructs to be able 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 a lack of text editor that could help more experienced person and speed up the specification process lowers its classification.
- as, at the time of this evaluation, there were no possibilities of decomposition, and the ability to help only in the system specification phase, turn it in a difficult tool to be used in the current process. These features have now been added.

• SCADE

- because SCADE is based on a different concept, where formal methods are behind the scenes, it has a great capability to deal with project constraints, but you still need some formal 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 captivate the user. A lot of things can be done based on templates and patterns, which is a great help.
- besides the capability to go from the specification to the code, it has also some other complementary tools that help you in important auxiliary tasks in the project, such as requirement management, traceability, etc.

7.3.4 Late 2009 - writing a formal specification - user point of view

B [1] 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 [2], the authors have presented a traceability between KAOS requirements and B. Some authors are investigating the use of the Problem Frames approach [3] as a possible response. A mixed solution using natural language and UML-B has been proposed by [4]. However, these approaches use non-standard artefacts for requirement specifications, which we consider a disencentive for convincing designers to adopt formal methods since they must spend time to learn them.

Use cases [5] can be considered as the *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 a natural language, which makes use cases readable for most end-users. In [6] the author has presented an approach for building B specifications from use-case models. This approach is similar to ours, but his project has focused 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 not correct since each use case can have many transactions according to Jacobson [7].

Another relevant point about uses cases is the possibility to derive test cases. A recent and important development model is the so-called Test Driven Development[8], 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 an approach for starting B specifications from use and test cases. Use cases 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. Until now, formal methods are commonly only introduced in the design phase where the requirements are already well defined, but this is not the case on industrial projects.

The method proposed here would be useful to be used during early phases, to help in both directions:

- to the top, helping the requirements elicitation (what might be combined with other techniques and methods, like Jackson's Problem Frames)
- to the bottom, helping the creation of the first abstract formal model, in the sense that it can support the first definitions.

For mapping use cases to traditional B specifications we propose that use case scenario sentences must be written using a controlled natural language (CNL) described according our use case transaction definition, which is based on Ochodek's transaction model [9].

Definition 1. A transaction is a shortest sequence of actor's and system actions, which starts from the actor's request and finishes with the system response. The system validation and system executive actions must also occur within the starting and ending action. The pattern for a transaction 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. 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 are good at telling the sequence of events. We have mapped the use case actor as subject, a set of actions predicate synonyms (for example validate, verify etc. would be grouped together) as verb and the rest of the sentence as object.

7.3.4.2 Results

We used an approach for mapping requirements to B (Event-B) models through use and test cases in a pragmatic way. We were not interested (at this moment) 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 for starting B specifications.

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

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

7.3.5 Early 2010 - A Methodological Approach to a B Formalization

Developing a formal specification of a system from the informal functional requirements of a customer remains an open issue in the software engineering world. Good requirements, in the sense of well written and easy to understand, are essential[3] but not sufficient for achieving this goal.

We propose a framework to structure and organize 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, beyond any doubt; so, we will consider the customer feedback once we have the model constructed and the inconsistencies and lack of information are clearly pointed out and proved. Second, this work is part of a larger project to develop a method for systematically organizing the requirements of a system having only as input natural language documents. As a side effect of this work, the formal model could help the customer to rewrite the natural language document to resolve such 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 [3], or KAOS [10]. This work employs an adaptation of the WRSPM approach [14], 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 base of the method explored in this work.

In order to achieve a model that we can verify and validate, that points out doubtless inconsistencies, redundancies and lack of information, the B method is applied over the specification from WRSPM. Once a formal model is constructed, one can use animation tools like ProB [13] to show this to the customer, which would be much easier to understand than mathematical proof obligations. It is also in this process that we can verify the advantages of the methodology and identify the main complications of 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 we propose to generate the WRSPM-based model consists in taking the customer requirements (expressed in natural language) one by one. This way we incrementally produce the specification **S** in terms of *phenomena*, *invariants* and *operations*. We break requirements into **W**, **R**, and establish traceability 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 Figure 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 *phenomena* that has been identified for the first time we define first invariants regarding to its type and next the invariants and operations relevant to its state. Once all *phenomena* of that single requirement are identified and (the new ones) typed and invariant/operation related, we break the

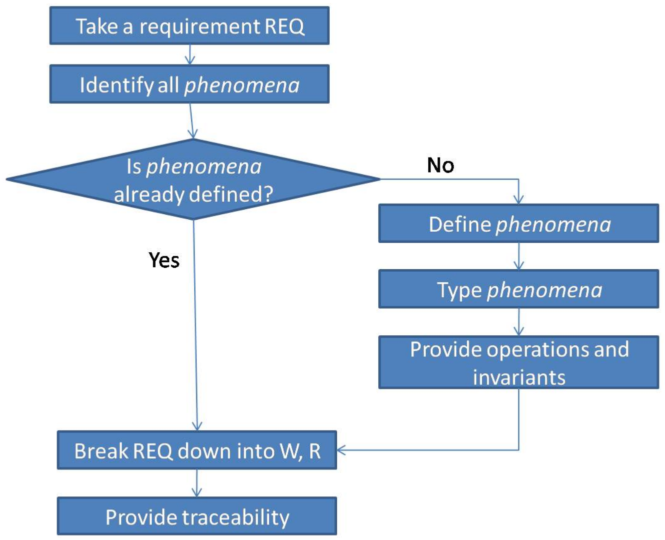


Fig. 7.1 Flowchart illustrating the processing of requirements.

text of the requirement into information of the environment (**W**) and constraints of 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 e.g. the B method). This is the starting point of a series of refinements, the last one resulting in **P** (in some programming language), to be executed in some execution environment **M**. In the process, dependency links shall 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 to have their formal modelling (the abstract model) done in just one step, and must have it done in successive steps. That is where enters the idea of *horizontal refinements* (or superposition).

The idea consists in starting with an abstract model that only considers part of the requirements, and introducing new requirements through successive refinements, adding new variables, strengthening pre-conditions, adding invariants, actions or operations.

The person doing the modelling process has to explore the requirements document (in the methodology presented in this paper, the WRSPM model) and gradually take from it the elements to be formalized, taking care to correctly define the architecture that supports the incremental addition of information and to define a good order to add this requirements, so abstraction disparities are not faced (like a very *concrete* requirement being added in a highly abstract level). Another issue is to guarantee completeness with the previous model (WRSPM): the horizontal refinements process must last 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 added what we considered enough to the abstract model (to be further expanded with the horizontal refinements), this model was analysed using the animations tool ProB [13], which provides features like model checking and execution step-by-step to achieve some state though the operations of 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 be 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 be to refine horizontally the abstract model of the system, since we only considered part of the requirements in the process of construction of the model, so the new requirements must be introduced through refinements. That was the point where 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, in a practical example: The *machine System* of the abstract model cannot include the *refinement CGP_r*, that cannot include the *refinement Opening_r*, and so on. In addition, one cannot add a new operation though refinements: it is necessary to go back to the abstract model, define the whole signature of the operation (with the variables to be received and returned) and so do the refinement step. To have the abstract model able to be horizontally refined one would have to do an analysis through *all* the requirements to define at least the signature of *all* the operations that are going to be performed in *all* the machines. Therefore, so that we could apply horizontal refinements, we decided to use another approach to build the model and do the refinements, using the Event-B [12] formal methodology and its extensions that enables the decomposition [11] process.

In Event-B we are allowed to add events (analogous to operations in B) through refinements, so we can indeed *expand* the model. The key point to the horizontal refinements in Event-B is, however, the *decomposition*. There is no **INCLUDES** abstraction, there is no communication between different *machines*, but one can introduce it 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 communication (through shared events) between the CGP and the environment, and to model the complexity division (through shared variables) between the CGP and its nested components. Once the 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 and approach of a graphical tool prototype based on formal methods to validate track topologies and train movement conditions. This tool is Eclipse-based and compatible with the Rodin platform. Its main features are summed up as the graphical simulation of railways specifications and the train movement properties validation. The tool is called VeRaSiS, and uses Event-B to formalize, 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, like 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 happen, 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 in charge of, among other tasks, dictating the train movement in a specific direction and secured condition. In order to guarantee these conditions, the track topology and the movement conditions are usually manually-translated into a set of properties that are verified and validated through the expert eyes of the validator. This process chain is not sufficiently reliable to guarantee that errors will not creep into the writing process, errors and/or inconsistencies that are very difficult to pinpoint during the validation phase. Nowadays, the superior effectiveness of formal methods over manual validation in ensuring safety in critical systems has been clearly demonstrated. Yet, despite the plethora of research and tools developed to facilitate the use of formal methods, mainstream developers still have a certain reluctance to use those methods. In order to help those developers overcome such a resistance, we developed VeRaSiS.

In the mid-80s, 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. The line 14 of the Paris Metro is one of the most exemplary outgrowth of this project. Currently, the line deals with more than 45 000 passenger per day 5 and since, 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 arduousness to master the use of these methods by mainstream software developers is still a problem. The use of formal methods in the railway domain depends mainly on the requirements type. This focuses on the train movement. More exactly, it takes a close look at the Boolean equation validation issue.

7.3.6.1 The Boolean equations validation problem

Boolean equations or Boolean logic is a logical 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 define the safe movement of a train from one location to another. A set of Boolean equations (called Boolean Equation Library - BEL) has to be generated for all paths in the model designed.

In industrial projects, those equations are generated manually and validated either manually or with formal tools. Besides the time spent to construct and validate these equations, sometimes errors or inconsistencies can still remain after the manual validation has been done, a probability which can, at worst, lead to accidents or, at best, if detected on time (that is, during the test phase), force developers to rework the equations from scratch.

7.3.6.2 Using Formal methods to validate Boolean equations

A preliminary analysis has been 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 was already validated manually. First, the example was translated into a classical B machine. A parser that converts the Boolean equations to B [5] notation has been developed in order to automatise the translation. This machine has been run in ProB and, it gave a counterexample showing a small error which was not detected in the first validation, two speed limits could be selected, non deterministically, at the same time. This case study has been taken at random, which means that probably there are other examples that contain errors and that were validated manually. This study has, firstly, shown companies the need to replace the manual validation by a 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. The VeRaSiS plug-in provides three main features: firstly, a graphical simulation of use cases, secondly, a fully automatic Boolean equations generation, and finally validation of the Boolean equations. The VeRaSiS plug-in comes up with a graphical interface allowing users to design railways systems. The BEL is automatically generated. After this, a formal tool is used to validate it. Error messages are translated into straightforward messages in order to shield users from the formal language. Even so, to be able to be used in an industrial environment, a minimum understanding of formalism is needed.

7.3.6.4 Results

We have presented a new formal tool to model railway specifications and validate trains movement properties. This Eclipse plug-in is designed for industrial needs and developed in order to meet the requirement that mainstream users can develop formal railway models while remaining as far as possible from difficult formal specifications. The VeRaSiS plug-in replaces the traditional Boolean equations 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 has two important conclusions. Firstly, the fact that we have to optimize the algorithm for generating the BEL from the graphical model and the translation template from BEL to Event-B. Secondly, the need for further work: normally the translation in one to one, as one BEL becomes one event, however as the complexity grows, introducing new kinds of BEL, this rule might not be valid anymore, which implies that there are further studies that need to be done. We also need to generate a set of rules for each kind of BELs that need to be created in a real project, as in the case study we selected only one (or two) kinds of BEL, so that we could validate the concept, but all the required equations have to be generated in an operational model. Additional case studies would certainly find other limitations of the VeRaSiS concepts. Our primary goal is to have a full first version of the VeRaSiS plugin in order to prove its functionality.

7.4 Conclusions

Some words should be said about the introduction of FM in the development process, mainly, in small companies (as in big ones, it seems to get easy as you could just hire some specialists and put them to work together with the development team). Until now, even after some training, we did not find a good way to change completely our development process. We do have some trained people that are able to understand and use, with some restrictions, Event-B and Rodin. 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 thought) it would be of some help to improve the product quality and dependability. In a perfect world, the approach of parallel development (FM and traditional) would be ideal to convince people of its power.

Using formal methods informally does really help in changing the way of thinking, meaning that, even if not all the process is formalized, the minds involved in the development are following a refinement process.

So far we use formal methods in different ways, like:

1. Verify gaps and mistakes in the natural language requirements. We model the specified system (only the abstract model) and use 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 was developed satisfies what was required. We have not found a way to do that 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, UML-B, etc.). It is a distributed system and different parts of the ”standard” system might be used in different customer projects - it is a sort of product line. We have not found any way to re-use parts of the specification, only to reuse the whole specification making any necessary modifications. Even with the Rodin decomposition tool, it is not clear that we would be able to re-use parts of the specification.
3. We think that the use of formal methods increases dependability of the system, though we lack strong evidence for that. 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 was not yet possible to support by measurement, but I can say for sure (based on my professional feelings), is that the adaption 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 was not fully achieved. This helped us to increase the quality of the final product and to drastically decrease the time dedicated to final tests.

References

1. Abrial, J.R.: The B-book: assigning programs to meanings. Cambridge University Press, New York, NY, USA (1996)

2. Ponsard, C., Dieul, E.: From requirements models to formal specifications in B. ReMo2V CEUR Workshop Proceedings **241** (2006)
3. Jackson, M.: Problem Frames: analyzing and structuring software development problems. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2001)
4. Jastram, M., Leuschel, M., Bendisposto, J., Jr, A.R.: Mapping requirements to b models. DEPLOY Deliverable - Unpublished manuscript (2009)
5. Jacobson, I.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley Professional (June 1992)
6. Ledang, H. Automated Software Engineering Conference
7. Jacobson, I.: Object-oriented development in an industrial environment. In: OOPSLA '87: Conference proceedings on Object-oriented programming systems, languages and applications, New York, NY, USA, ACM (1987) 183–191
8. Beck, K.: Test Driven Development: By Example. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002)
9. Ochodek, M., Nawrocki, J.: Automatic transactions identification in use cases. (2008) 55–68
10. Darimont, R. and van Lamsweerde, A.: Formal refinement patterns for goal-driven requirements elaboration. In: Proceedings of the 4th ACM SIGSOFT symposium on Foundations, 1996.
11. Butler, M. and Hallerstede, S.: The Rodin Formal Modelling Tool. `http://deploy-eprints.ecs.soton.ac.uk/4/1/eventb.pdf`
12. J-R. Abrial. Modeling in Event-B: System and Software Engineering. Cambridge University Press, May 2010.
13. ProB. Animator and model checker, February 2012.
14. Gunter, C. and Gunter, E. and Jackson, M. and Zave, P. A Reference Model for Requirements and Specifications. IEEE SOFTWARE, 41, 2000.

Chapter 8
Critical Software Technologies experience with Formal Methods

Alex Hill, Jose Reis,Paulo Carvalho

Abstract Critical Software Technologies(CSWT) participated in the DEPLOY project as an Associate and the chosen case study was the Integrated Secondary Flight Display (ISFD) used onboard commercial and military aircraft. The chosen case study was used to explore and learn both Event-B method and the toolset Rodin. This chapter relates the experience of CSWT with Event-B in the context of DEPLOY, describing the advantages of the formal method for the domain safety and mission critical systems and lessons learned. The conclusions achieved show that Event-B has great potential in the safety and mission critical systems domain. The performance of the verification and validation process defined in 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 those properties through controlled experiments are seen as the right route to reduce 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 been not 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 applicable to CSWT can be characterised by 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

Jose Reis
Critical Software Technologies Ltd,2 Venture Road,Southampton Science Park, Chilworth, Southampton - SO16 7NP,United Kingdom e-mail: `jreis@critical-software.co.uk`

- 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 depending on the criticality of the software
- define what activities are required for each criticality level

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 impact on the quality of the outputs, cost and schedule of the project. Taking DO-178 for example, the standard identifies specific criticality levels for the software. Level-A is the most critical level and E the lowest one. 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 anomalous behaviour within a system classified as Level-C would result 'only' in a major failure condition of the aircraft e.g. 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's operation or 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 has been invoked at least once, every decision has taken all possible outcomes and each condition in a decision has been shown to independently affect that decision's outcome. Avionics software classified as Level C requires 100% statement coverage, which means that every statement in the program has been exercised. In terms of costs the following table illustrates the delta cost and schedule of DO-178 per criticality level:

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%

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; where most are not integrated. One of the downsides of having an heterogeneous development environment is that the effort required to propagate a change up and down the documentation hierarchy can be high and cost the project a significant amount of money. It is also very error prone, because it is very difficult to guarantee the 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 as per 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. Effectively, systems engineering programmes are constrained by commercial pressures which lead to 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 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 ISFD 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 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 ISFD unit

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

8.2.1 Project Team Structure

The team working in the project was structured as follows:

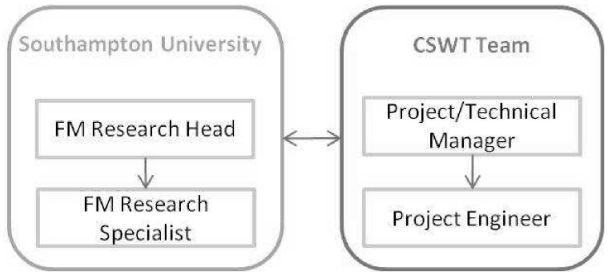


Fig. 8.1 Team composition

The CSWT team had no previous experience with Event-B or Rodin, and so the team went through four training sessions provided by Southampton University.

These were half day sessions followed by workshops with Southampton University reviewing the models produced by CSWT and to provide further training on specific topics such as model decomposition. The training sessions proved to be very useful and demonstrated with 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 a Product Description Specification.

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

- Phase I
  - Included the generation of a requirements document from scratch using existing knowledge of the system
  - Allowed experimentation with the RODIN tool and the creation of some very basic models
- Phase II
  - Requirements specified in the previous phase were re-factored
  - Generated new Event-B model covering System Power On/Off Events and Attitude Alignment events
- Phase III
  - Refactored requirements to cover Display properties and 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
    - Created first refinement to address range checking and attitude alignment

The first phases did not follow a consistent methodology because the input specification was not structured to a suitable level in as much as 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 the figure below:

The class diagram proved to be quite useful when reasoning about the main sets in the model and the relationship between sets. 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 easily be translated into sets and relationships in the Event-B model.

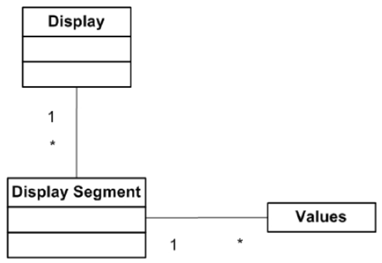


Fig. 8.2 Example class diagram used

An extract of the context model created is provided hereafter:

CONTEXT deployISFDV01_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 were modelled:

```
EVENTS
Event    $changeSpeed \widehat{=}$
   begin
      act01 :  $aircraft\_speed :\in \mathbb{N}$
   end
Event    $SystemPowerOn \widehat{=}$
   when
      guard01 :  $system\_status = OFF$
      guard02 :  $aircraft\_speed < 1$
   then
      act01 :  $system\_status := ON$
      act02 :  $current\_mode := Nominal$
   end
Event    $AttitudeAlignment \widehat{=}$
   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 :  $current\_mode = Nominal \lor current\_mode = ILS$
   then
      act3 :  $f\_attitude\_alignment := TRUE$
   end
Event    $PressAlignmentButton \widehat{=}$
   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 and that the consequence is for proofs to be more difficult to generate and for the model to become more complex. A cookbook developed previously by Southampton University Electronics and Computer Science group [3] aiming at developing a systematic process for modelling and refining a control problem system by distinguishing environment, controller and command phenomena, proved to be a valuable method of separating concerns and dividing the problem into smaller problems susceptible to a structured approach. The cookbook [3] identifies a pattern which helps separating the controller from monitoring and operator actions. Using this pattern the Event-B model identified events related with the monitoring of sensors values, the outcome of a set of values in the display unit, leaving at very abstract level sensors. Events related with operator actions such as changing display mode were also captured. Each of the three main types of events, i.e. 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 the cookbook [3] 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.

The case study developed in the context of DEPLOY provided a good basis to understand how Event-B can help strengthen requirements. The proof generation mechanism flags up when there is a problem with the requirements, and the model checker confirms that the model has been constructed correctly. Future work is being conducted to further explore the usage of Event-B to develop Cyber-physical systems[1] [1] 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 the drawing of conclusions on 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 known successful applications of formal methods on real cases such as the Paris metro, A380 avionics and the UK National Air Traffic Services (NATS) - Interim Future Area Control Tool Support (iFACTS) set high expectations. CSWT investment was carefully controlled and the model produced was relatively simple as shown in table 8.2. Notwithstanding this, the results achieved were encouraging and CSWT is already working on two additional projects where the Event-B method is being used.

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

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

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

- Requirements strengthening : RODIN and the automatic proof generation mechanism highlight when there is a problem with the system requirements. Thinking about the problem in terms of sets and invariants pushes the system engineer to further think about the problem and the solution 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 breakdown of the system and guarantee the 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

There are still a number of characteristics that are important for CSWT which have potential and which may be improved:

- Traceability between requirements and formal model: traceability between requirements specified in a natural language and the 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 flowed down to the design model.
- Test case generation from 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 [1]).
- Team-based formal development : the parallelisation of activities in a design model is required to enable efficient usage of project resources and to meet stringent programme timescales. For instance if a classical approach is adopted the system will be broken down into smaller parts and each part will be allocated to different teams. Each team can then work on its allocated part and various synchronisations points can be defined to better assure the interfaces between the various system parts. CSWT sees refinement and decomposition as essential features to facilitate team working, but several levels of refinement may be required before specific sections of the formal model can be allocated to different team members. From a process perspective it is important to understand how to refine models to enable a quick transition to an environment where engineers can work concurrently on models. 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

The experience of CSWT 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 parts are only 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. Jean Abrial Raymond [4] presents in his book several examples of how this can be achieved. For instance, in the context of the case study CSWT decided to exclude from the modelling sensors behaviour, numerical processing done within the ISFD and display brightness adjustments, so that effectively everything that is outside the display unit is seen as environment and is modelled at an abstract level.
- 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 the set theory and to think about the problem in terms of sets from the start. Class diagrams like the one depicted in Figure 8.2 and the relationships between the 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. The experience in DEPLOY showed that working closely together with a group of professors with several years of experience helped with the grasping of 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 usage of the approach. Jean Abrial Raymond [4] presents in his book several real scenarios where Event-B is used.

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

References

1. *Advanced Design and Verification Environment for Cyber-physical System Engineering*, urlhttp://www.advance-ict.eu/index.html
2. Hilderman V (2009). *DO-178B Costs Versus Benefits.* High Rely.
3. Sanaz Yeganefard, Michael Butler,Abdolbaghi Rezazadeh. *Evaluation of a Guideline by Formal Modelling of Cruise Control System in Event-B*, ECS, University of Southampton, 2010.
4. Jean-Raymond Abrial.*Modeling in Event-B, System and Software Engineering*,Cambridge University Press,2010.

Chapter 9
Experiences Deploying Event-B in an 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 such as 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 runtime 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 the Event-B notation, using the Rodin toolset. This project applied and extended the Event-B and Rodin based techniques for ISA analysis, developed at the University of Bristol, 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, forms 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 uncontrol-

Stephen Wright
University of Bristol, Department of Computer Science, MVB, Woodland Road, Bristol BS8 1UB, UK, e-mail: stephen.wright@bris.ac.uk

Kerstin Eder
University of Bristol, Department of Computer Science, MVB, Woodland Road, Bristol BS8 1UB, UK, e-mail: Kerstin.Eder@bristol.ac.uk

lable 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, being distinct 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 behavior 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, in particular, to exhaustively cover all the cases. Formal Methods are best suited to overcome problems of exactly this nature. The difficulty in understanding has also been exacerbated with the appearance of Reduced Instruction Set Computer (RISC) machines, in which the instruction set is designed purely for efficiency, with no thought to easing the burden of a programmer attempting to write a program or understand the rationale of a knotty instruction sequence. Also, as with many things in life, computers spend most of their time being 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, i.e. handling the occasions when things do go wrong.

In this chapter we introduce the XMOS XCore commercial microprocessor, and describe a project 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, and 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 re-used 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 8. The ISA contains a range of typical instructions such as branching and jumping, register-to-register calculations and memory access, but also provides support for efficient synchronized 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 run-time 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 alphabetical order of its assembler mnemonic, and containing a mixture of natural language description and semi-formal pseudo-code 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 parametrization 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 tools are commonly available with most tools, but these are targeted at correction of high-level programming mistakes, and themselves rely on the ISA being fully understood. A published ISA specification is the primary reference source for engineers and computer scientists developing and optimizing the compiler tools. More directly, functions may be coded in the device's 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 use of an ISA without its proprietary micro-architecture, leaving 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 ensures 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 behaviors, such as early instruction pre-fetching, or scheduler mode changes.

9.4 The XCore Formalization Project

9.4.1 Project Objective

Our primary objective for the XCore ISA formalization project was to provide a rigorous re-statement of XMOS's existing specification. In particular, we were seeking to identify errors or omissions in the document. These could be direct mis-statement of behavior or formats, omission of possible conditions, and ambiguity 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 organizations soon after the completion of their studies, in order to transfer and exploit their hard-won, and hard-funded, new ideas into industry. The XCore ISA formalization project was an exact fit to 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" [10]. The KTS benefited from the close links between XMOS and the University. XMOS hosted the project from October 2010 for one year. As part of the original proposal we planned a specific set of tasks and objectives, and then reported on progress throughout the 12 months. We describe these tasks next, and their achievement during the project.

9.4.3 Project Tasks

9.4.3.1 Architecture and Tool Chain Familiarization

The Architecture and Tool Chain Familiarization task covered initial familiarization with all aspects of the XCore, with particular emphasis on the ISA and XMOS's tool support for it [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 XMOS's real and software-emulated devices. We fully completed this task, gaining full understanding of all the instructions and their interactions, ready for their capture into 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 re-ran the test executables created for the initial familiarization 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 Tool Set Review

Due to the fast pace of development in the field, during the Rodin Tool Set Review task we researched all the developments 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 fully completed: a detailed review of the available Rodin tool-set 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 due to being insufficiently 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; to develop 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 a combination of using the Rodin text editor to allow rapid copy-and-modify development, and 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, would require proof guidance and result 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 combination of the existing “Relevance Filter” plug-in and 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 Modeling Framework Update

With these groundwork tasks completed, we were ready to begin: the Generic Modeling Framework Update task sought to customize 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 partially completed, being performed for only the core instructions.

9.4.3.7 XCore ISA Formalization

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

MIDAS experience emphasized the importance of VM generation and test, and we planned the B2C [9] 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, a better solution was found: we fulfilled the task's intent with the enhancement of the B2C tool and the development of more concise Event-B constructs, allowing a VM to be generated directly from the main formal specification. Although these are described separately, in reality generation and test of the VM went hand-in-hand with model construction and expansion of the test-suite 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 covering the generation of a natural language document derived from the formal specification, describing the ISA behavior under all conditions. This was not achieved in the time available, although numerous corrections to the existing specification were identified and reported via XMOS's computerized reporting system.

9.4.4 Project Results

Detailed documentation describing the structure of the formal model, and instructions for its download, build and execution was fully completed 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, automatic proof discharge). Equally typical were the unforeseen "showstopper" problems that could in isolation cause the project to fail completely (e.g. 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 through to implementations. VDM and B-Method support similar notions of action definition, although B-Method syntax gives an appearance closer to 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 stress 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 the incremental decomposition of system behavior 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, both in the notation itself and its supporting tools. To address the latter, Event-B was developed closely alongside a supporting open-source tool-set, Rodin. Rodin has a modular architecture based on the Eclipse framework with clearly defined Application Programming Interfaces (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 to 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, both in the notation itself and its supporting tools. Regarding the former, Event-B does not strictly define a notation, but 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 notation. The Rodin tool-set 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 the addition of syntactic sugar and the automation of repetitive procedures. The tool may 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 its 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 provision of locally produced plug-ins, reducing risk to development. This is an essential feature in an industrial setting.

9.6 Formal Specification

Modelling in Event-B typically stars 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 reached containing all details required for coding or, ideally, automatic code generation. In contrast to this "top-down" modelling methodology, which results in a hierarchy of increasingly more complex models, the XCore ISA is "flat", i.e. the complete specification is presented at the full level of detail. This left 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, including both valid and invalid. Instruction groups are constructed by the successive partitioning of sub-sets of *Inst* based on common features, e.g. control flow instructions, instructions which access the internal storage, load/store instructions, etc. This gives rise to a hierarchy of abstraction layers. These sub-sets are then employed in the guards of events describing the possible outcomes of execution of a particular instruction group. An event describing the successful execution of the instruction is initially created by 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", in which instructions are grouped according to their general properties, and "horizontally", in which events are incrementally enriched via the Event-B "extends" mechanism.

The XCore ISA implements 176 operations, some of which have multiple encodings, resulting in 209 possible instructions. Construction of each 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 ISAs functionality 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. Firstly, direct errors were found, where a detail is unambiguously incorrect, and a compiled executable would never execute correctly under the specified behavior. An example of this was the transposition of instruction fields in a format descriptor. Secondly, ambiguities were found which could be open to misinterpretation and potentially lead to differences in behavior between different interpretations, an example being modulo arithmetic being used for address-offset calculation but not 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 thus 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 modeling of other ISAs and test-case coverage projects in 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 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 "head-full". 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 address-offset calculation being signaled by 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 made in 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 Rodin tool's ability to cache and reapply proofs after being trivially touched was of some help, but many changes (such as renaming of variables) would break many proofs such that full reproof was needed. For a project requiring over 1700 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 as the basis of new developments (for example, development of the B2C tool was initiated with study of 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 that 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 consist of API evolution rendering plug-ins incompatible, through to the need for ongoing review of available tools and their maturity, in order to maintain productivity improvements.

Event-B models and proofs could only be fully inspected via the Rodin tool, This presented a problem: the company 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 into the 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 files. Thus, if the formal model were to be integrated into this design flow, Rodin architecture would provide easy integration via generation of XML input files in the existing format. This would be achieved by development of an appropriate plug-in: a relatively straightforward process.

9.9 Industrial Perspective

Although 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 project "Deploy", and XMOS's own open-source website. This approach has stimulated comment and questions in the forum of XMOS's tech-savvy customer developers. 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 resource is 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, and contrasts 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 team 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 well occupied with simply picking up the existing tools and methodology. Cultural and practical issues such as this cannot be dismissed, as staff turnover and re-division of responsibilities within a company are a common occurrence.

XMOS's existing verification methods 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 actual silicon. As is to be expected for such a methodology, the company has several functional models of the ISA already, and any attempt to introduce another will need to demonstrate significant advantage with respect to 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. As well as its technical role, the use of formal model based simulation is also important in this respect. It fulfills the role of existing emulators and thereby instills 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, for which resilience to multiple simultaneous system failures are a major factor in the design process, but are not 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 a connection on a mobile phone: 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 with their associated risks and costs in time and resource. 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 aid understanding to construct well validated specifications. Theorem proving is usually assumed to be used for static verification of equivalence between this flat specification and a conventional implementation: the use of theorem proving to enforce correctness within the specification, via techniques such as well-definedness and invariant proving, is not common knowledge. Formal Methods are also often assumed to replace all or most testing: the dynamic testing of formal models and their use for test generation is not common knowledge. This was perhaps demonstrated during the initial definition of the project goals: the industrialists were interested in a simple unambiguous statement of the existing specification, whereas the academics where 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 tasks such as editing (by enabling sub-division of models as well as large file handling), building, proving (e.g. automatic proof discharge for large hypothesis sets) and work allocation (e.g. parallelization of work and proving to allow team working). For example, editing and build performance should be comparable to that of conventional integrated development tools, and automatic proof discharge needs to remain at the approximately 95% achieved for small models. These capabilities are essential in any tool's core platform in order to provide 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 to achieve these goals. Automated presentation of a model's event and refinement inter-relations in graphical and linked-document form (such as XML) will also allow efficient reviewing via standard desktop tools.

Finally, reliable support is needed for a successful industrial tool. Academic and open-source tools are generally provisioned 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 down-time 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 organizations that exist to support (and fairly profit from) the open-source Gnu toolset and 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 or, still rather rarely, theorem proving. Model checking, sometimes referred to as property checking, is now used quite routinely in the microelectronics design industry, and more recently also 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, to monitor design activity, and for formal property checking. While formal property checking has become an accepted technique to demonstrate desired properties of a design, in practice the remaining challenge is for engineers to know whether they have specified "enough" properties, and, also, how much of the design is actually covered by the existing properties. This is still 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, and 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 chose. This has the added advantage of not having to define a domain-specific semantics for most of the model - this comes "for free". Self-consistency and completeness are inherent to 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 the generation of proof obligations. The third is to demonstrate domain-specific properties which are explicitly introduced during modelling in the form of invariants and theorems. Specific to ISAs are, e.g. deadlock freedom, i.e. the ISA defines transitions for all input conditions for all processor states, and determinism, i.e. the ISA defines 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 demonstrate 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 level, this one is typically referred to as the specification, and one lower level. 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 used include model checking, predominantly in industry, and theorem proving, 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 also complete in that it must cover all the behaviors 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 formalization 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 the next generation of the processor is being developed. Typically, this 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 lends itself naturally to modification and extension based on the principle of stepwise refinement. The need to formally establish model consistency between refinement steps, which is inherent to the Event-B method, guarantees the absence of inconsistencies being introduced during the ISA extension process.

9.12 Conclusion

During 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. Once again Pareto was right: large amounts of resource were expended on covering a small part of the specification's 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 these 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's 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 by the start of a new large project. However, the open-source culture and infrastructure of the formal method used has allowed the model to be preserved and published outside the company, maintaining the possibility of future deployment and opening this application up for future research.

References

1. J. Bendisposto, F. Fritz, M. Jastram, M. Leuschel, and I. Weigelt. Developing Camille: A text editor for Rodin. *Software: Practice and Experience*, 2011.
2. J. Bergeron. *Writing Testbenches: Functional Verification of HDL Models*. Kluwer Academic Publishers, 2nd edition, 2003.
3. H. Chockler, J. Y. Halpern, and O. Kupferman. What causes a system to satisfy a specification? *ACM Trans. Comput. Logic*, 9:20:1–20:26, June 2008.
4. H. D. Foster, A. C. Krolnik, and D. J. Lacey. *Assertion-Based Design*. Springer, 2003.
5. J. Hennessy and D. Patterson. *Computer Architecture: A Quantitive Approach*. Morgan Kaufmann, 2003.
6. D. May. *XMOS XS1 Architecture*. XMOS Ltd., July 2008.
7. J. Roder. Relevance Filters for Event-B. *ETH Zrich*, 2010.
8. D. Watt. *Programming XC on XMOS Devices*. XMOS Ltd., 2009.
9. S. Wright. Automatic Generation of C from Event-B. *Workshop on Integration of Model-based Formal Methods and Tools*, 2009.
10. S. Wright and K. Eder. Using Event-B to Construct Instruction Set Architectures. *Formal Aspects of Computing*, 23(1):73–89, January 2010.

Appendix B. Full Programme of the Federated Event

Deploy Federated Event

Rodin Developer Tutorial Programme

Monday 27th February

Time	
08:30 – 09:00	***Coffee***
	Session 1: *Using the Rodin Theory Plug-in* - Issam Maamria (1h) *Using the Rodin prover API to connect external provers* *SMT Solver Integration* - Systerel (30 min) *Isabelle/HOL integration* - Matthias Schmalz (30 min)
11:00 – 11:15	***Break***
	Session 2: *Creating and using custom/parameterized proof tactics* - Nicolas Beauger, Jean-Raymond Abrial (30 min) *Integrating plug-ins with ProB* - Jens Bendisposto (30 min)
12:15 – 14:00	***Lunch***
	Master Class session 1 (1h30) in 4 Rooms: Room A : *Developping Theories* - Issam Maamria Room B : *Using custom/parameterized proof tactics* - Jean-Raymond Abrial, Nicolas Beauger, Thomas Muller Room C *: Parameterizing Isabelle with theories* - Matthias Schmalz Room D *: Developing for ProB* - Jens Bendisposto
15:30 – 16:00	***Coffee***
	Master Class session 2 (1h30) in 4 Rooms: Room A : *Developping Theories* - Issam Maamria Room B : *Using custom/parameterized proof tactics* - Jean-Raymond Abrial, Nicolas Beauger, Thomas Muller Room C *: Parameterizing Isabelle with theories* - Matthias Schmalz Room D *: Developing for ProB* - Jens Bendisposto

Deploy Federated Event

Rodin User & Developer Workshop Programme

Tuesday 28th February

Day One

Time	
08.30 – 09.00	***Coffee***
09.00 – 10.30	***Session 1:*** Welcome *SafeCap Modelling Environment* - Alexei Iliasov *Verification of a Railway Interlocking UML Model Translation to UML-B* - Gintautas Sulskus & Colin Snook *Component Reification in System Modelling* - Jens Bendisposto & Stefan Hallerstede
10.30 – 11.00	***Break:***
11.00 – 12.30	***Session 2:*** *Code Generation Update* – Andrew Edmunds C.J.Lovell, R. Silva, I.Maamria & M.Butler *Ensuring Extensibility with Code Generation* – Chris Lovell, A. Edmunds, R.Silva, I. Maamria & M. Butler *Towards a Certifying Code Generator for Rodin* – Alexei Iliasov *Generating Executable Simulations from Event-B Specifications* – Faquing Yan, Jean-Pierre Jacquot, and Jeanine Souquières
12.30 – 13.30	***Lunch:***
13.30 – 15.30	***Session 3:*** *Fault Tolerance Views* – Ilya Lopatkin, Alexei Iliasov, Alexander Romanovsky *Use of Rodin in FDIR Architechture for Autonomous Systems* – Jean-Charles Chaudemar *Pattern for Modelling Fault Tolerant in Event-B* – Michael Poppleton & Gintautas Sulskus *Extending Event-B & Rodin with Discrete Timing Properties* – Reza Sarshogh & Michael Butler
15.30 – 16.00	***Coffee:***
16.00 – 17.30	***Session 4:*** *A Framework for Diagrammatic Modelling Extensions in Rodin* – Vitaly Savicks & Colin Snook *Systematic Development for Embedded Systems Design using RRM Diagrams and UML-B* – Manoranjan Satpathy, Colin Snook, Silky Arora *CODA: A formal Event-B based Refinement Framework for High Integrity Embedded System Development* – John

	Colley, Michael Butler, Colin Snook, Neil Evans, Neil Grant & Helen Marshall *ADVANCE: Advanced Design & Verification Environment for Cyber-Physical Systems Engineering – The Multi-Simulation Framework* – John Colley & Michael Butler

Deploy Federated Event

Rodin User & Developer Workshop Programme

Wednesday 29th February
Day Two

Time	
08.30 – 09.00	***Coffee***
09.00 – 10.30	***Session 5:*** *Proving Consensus* - Jeremy Bryans & Alexei Iliasov *Verification and validation of BPEL processes - A proof and animation based approach* - Idir Ait-Sadoune, Yamin Ait-Ameur & Mickael Baron *Formal Specification of a Mobile Diabetes Management Application Using the Rodin Platform and Event-B* - Daniel Brown, Ian Bayley, Rachel Harrison, Clare Martin *Synthesis of Processor Instruction Sets from High-Level ISA Specifications* - Andrey Mokhov, Alexei Iliasov Danil Sokolov, Maxim Rykunov, Alex Yakovlev, Alexander Romanovsky
10.30 – 11.00	***Break:***
11.00 – 12.30	***Session 6:*** *An Event-B Plug-in for Creating Deadlock-Freeness Theorems* - Faquing Yang & Jean-Pierre Jacquot *The Theory plug-in and its Applications* - Issam Maamria & Michael Butler *Can rippling discover the missing lemmas for invariant proofs?* - Gudmund Grov, Yuhui Lin & Alan Bundy *Proof Hints for Event-B Models - Extended Abstract* - Thai Son Hoang
12.30 – 13.30	***Lunch:***
13.30 – 15.30	***Session 7:*** *Requirements Traceability between Textual Requirements and Event-B Using ProR* - Michael Jastram, Lukas Ladenberger & Michael Leuschel *Towards Relating Sub-Problems of a Control System to Sub-Models in Event-B* - Sanaz Yeganefard & Michael Butler *Lessons from Deployment* - Manuel Mazzara, Cliff Jones & Alexei Iliasov *Assessment of the Evolution of the RODIN Open Source platform* - Christophe Ponsard, Jean Christophe Deprez, Jacques Flamand
15.30 – 16.00	***Coffee:***

16.00 – 17.30	***<u>Session 8:</u>*** *A Rodin Plugin for automata learning and test generation for Event-B* – Ionut Dinca, Florentin Iplate, Laurentiu Mierla & Alin Stefanescu *VTG - Vulnerability Test cases Generator, a Plug-in for Rodin* - Aymerick Savary, Jean-Louis Lanet Marc Frappier & Tiana Razafindralambo *Visualisation of LTL Counterexamples with ProB* – Andriy Tolstoy, Daniel Plagge & Michael Leuschel.

Deploy Federated Event

DEPLOY Industry Day Programme

Thursday 1st March

Time	
08:30 – 09:00	***Coffee***
	Introduction – Alexander Romanovski *Experience in Formal Modelling of Mode-Rich Systems in the Space Sector* – Timo Latvala *Formal Modelling in the Railways* – Hung Le Dang
10:30 – 10:45	***Break***
	Formal Methods in the Engineering of Enterprise Applications – Andres Roth *Formal Modelling in the Automotive Sector* – Rainer Gmehlich
12:15 – 14:00	***Lunch***
	Event-B Modelling of a CBTC system and its (3D) Animation – Thomas Muller *Formal Proofs for the NYCT line 7 (Flushing) Modernization Project* – Denis Sabatier *The Xcore Instruction Set Architecture in Event-B* – Stephen Wright
15:30 – 15:45	***Coffee***
	Event-B for Embedded Systems – Jose Reis *Possible Applications of Event-B in Small Industries* –Aryldo Russo *A Collaborative FAQ Approach for Collecting Evidence on Formal Method Industrial Usage* - Jean Christophe Deprez *Formal Mind, ProB, ProR and Data Validation with B* – Michael Leuschel