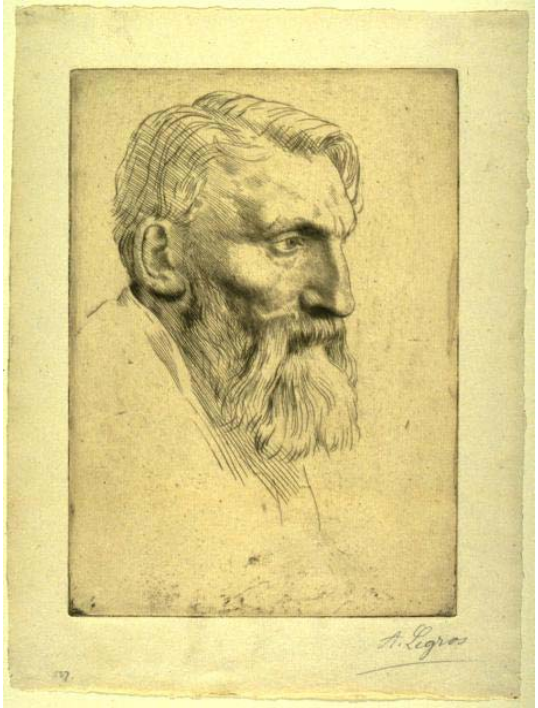
Project IST-511599

RODIN

"Rigorous Open Development Environment for Complex Systems"



RODIN Deliverable D34 (D7.4)

Assessment Report 3

Editor: *Pete White (Praxis High Integrity Systems)*

Public Document

26th October 2007

http://RODIN.cs.ncl.ac.uk/

Contributors:

Jean-Raymond Abrial (Swiss Federal Institute of Technology, Zurich, Switzerland),
Budi Arief (University of Newcastle upon Tyne, UK),
Michael Butler (University of Southampton, UK),
Joey Coleman (University of Newcastle upon Tyne, UK),
Alexei Iliasov (University of Newcastle upon Tyne, UK),
Ian Johnson (ATEC, UK),
Cliff Jones (University of Newcastle upon Tyne, UK),
Victor Khomenko (University of Newcastle upon Tyne, UK),
Maciej Koutny (University of Newcastle upon Tyne, UK),
Linas Laibinis (Åbo Akademi University, Finland).
Sari Leppänen (Nokia, Finland),
Thierry Lecomte (ClearSy, France),
Michael Leuschel (Heinrich-Heine-Universität Düsseldorf),
Ian Oliver (Nokia, Finland),
Rozilawati Razali (University of Southampton, UK),
Abdolbaghi Rezazadeh (University of Southampton, UK),
Alexander Romanovsky (University of Newcastle upon Tyne, UK),
Colin Snook (University of Southampton, UK),
Elena Troubitsyna (Åbo Akademi University, Finland),
Laurent Voisin (Swiss Federal Institute of Technology, Zurich, Switzerland),
Jon Warwick (University of Newcastle upon Tyne, UK).

REVISION HISTORY

Version	Status	Date	Notes
0.1	Draft	02/06/05	First draft for internal comments (year 1)
0.2	Draft	18/07/05	Complete draft ready for external review (year 1)
1.0	Definitive	26/08/05	Definitive issue following external review (year 1)
1.1	Draft	28/09/06	Draft for internal comments (year 2)
1.2	Draft	29/09/06	Draft for external comments (year 2)
2.0	Definitive	02/10/06	Definitive issue following external review (year 2)
2.1	Draft	22/08/07	Draft for internal comments (year 3)
2.2	Draft	5/9/2007	Draft for internal comments (year 3)
2.3	Draft	6/9/2007	Draft for internal comments (year 3)
2.4	Draft	17/9/2007	Draft for internal review (year 3)
2.5	Draft	20/9/2007	Draft for internal review (year 3)
2.6	Draft	2/10/2007	Draft for internal review (year 3)
2.7	Draft	4/10/2007	Draft for internal review (year 3)
2.8	Draft	8/10/2007	First complete draft for internal review (year 3)
2.9	Draft	9/10/2007	Includes review comments from C Jones
3.0	Definitive	26/10/2007	Definitive issue following Rodin exec review

ANTICIPATED CHANGES

None.

TABLE OF CONTENTS

Section 1 Introduction 5
1.1 Background 5
1.2 Scope 5
1.3 Purpose 5
1.4 Structure 6
Section 2 Assessment Approach 7
2.1 RODIN Objectives 7
2.2 Response to Year Two Project Report 7
2.3 Measurement Criteria 9
2.4 Approach 10
Section 3 Overall Assessment Results 12
3.1 Overview 12
3.2 Overall Results Summary 13
3.3 Summary of Quantitative Metrics Assessment 15
3.4 Assessment overview conclusions 18
Section 4 Methodology Assessment 21
4.1 Introduction 21
4.2 The (generic) Event-B methodology 21
4.3 Case Study Feedback 22
Section 5 Case Study Assessments 23
5.1 CS1 – Formal Approaches to Protocol Engineering 23
5.2 CS2 – Engine Failure Management 32
5.3 CS3 – Formal Techniques within an MDA Context 55
5.4 CS4 – CDIS Air Traffic Display Information System 61
5.5 CS5 – Ambient Campus Assessment 71
Section 6 Open Tool Kernel Assessment 82
6.1 Introduction 82
6.2 Current Status 82
6.3 Progress since Year 2 Assessment 82
6.4 Interaction with Plug-in Developers 83
6.5 Kernel Metrics 83
6.6 Conclusion 89
Section 7 Plug-in Assessments 90
7.1 Mobility plug-in (Mobile B Systems) 90
7.2 ProB model checking and animation 99
7.3 Brama 110
7.4 UML–B 117
7.5 B2RODIN 126
Section 8 References 132

SECTION 1 INTRODUCTION

1.1 Background

This document presents the results of the technical review and assessment conducted at the end of the three-year RODIN project. This document is a RODIN project deliverable.

The assessment was carried out according to the Procedure for Technical Review and Assessment [6], which established the metrics for RODIN, based on the contributions of goals and criteria from each RODIN partner. This procedure [6] was revised and improved in the light of experience gained from production of the first version of this report at M12 (Deliverable: D14 [11]).

1.2 Scope

The scope of this deliverable is focussed on the five case studies under work package 1 (WP1), the open source formal methods tool kernel developed for work package 3 (WP3) and the plug-ins developed as part of work package 4 (WP4).

The methodology, which has been developed under work package 2 (WP2), is also briefly summarised in Section 4. However, as noted in the Procedure for Technical Review and Assessment [6] and supported by comments at last year's review [2], an effective assessment of the methodology will need to focus on more qualitative than quantitative measures. This qualitative summary is already covered in the following project reports, and is therefore not repeated in this report.

- D27 [19] describes how "patterns" and "templates" have been applied by the case studies in order to tailor the methodology to each specific case study's needs.
- D28 [20] provides a qualitative assessment of the methodology as a result of the case study feedback.
- D29 [21] provides more in-depth feedback on the effectiveness of the Methodology as a result of the case studies.

We do not consider WP5 (dissemination) or WP6 (project management) since these work packages do not contribute directly to the measurable RODIN objectives. We do not define metrics to apply to WP7 itself.

1.3 Purpose

This document provides a view of the progress of the RODIN project in meeting its objectives and vision. It highlights the accomplishments of each work package during the project's lifetime.

1.4 Structure

Section 2 of this document discusses the assessment approach used for generating the metrics. Section 3 presents the overall assessment results of the RODIN project at month M36. Section 4 briefly discusses the feedback from WP2, the methodology part of the project.

The document then provides the detailed quantitative feedback, with qualitative evidence and supporting information, from:

- Each Case study, WP1 – Section 5,
- The tool kernel, WP3 – Section 6,
- Each tool Plug-Ins, WP4 – Section 7.

Finally, Section 8 contains the references used within this document.

SECTION 2 ASSESSMENT APPROACH

2.1 RODIN Objectives

To conduct the assessment the projects main objective must first be revisited. RODIN's objective is described in Section 2 of the Description of Work (DoW) [1]:

"The overall objective of the RODIN project is the creation of a methodology and supporting open tool platform for the cost effective rigorous development of dependable complex software systems and services."

The Description of Work [1] identifies the following four specific measurable objectives:

O1 A collection of reusable development templates (models, architectures, proofs, components, etc.) produced by the case studies. The goals of the cases studies will be defined in detail by month 6. Initial and intermediate results will be available by months 12 and 24, while a final set of development templates will be available by month 36 of the project.

O2 A set of guidelines on a systems approach to the rigorous development of complex systems, including design abstractions for fault tolerance and guidelines on model mapping, architectural design and model decomposition. Initial and intermediate guidelines will be available by months 12 and 24, with the final versions by month 36.

O3 An open tool kernel supporting extensibility of the underlying formalism and integration of tool plug-ins. Open specification of the kernel will be made publicly available by month 12 of the project. Prototypes of the basic tools will be available by month 18. Full working versions will be available by month 30, with final versions being ready by month 36.

O4 A collection of plug-in tools for model construction, model simulation, model checking, verification, testing and code generation. Open specification of the plug in tools will be made publicly available by month 12 of the project. Prototype versions of the plug-in tools will be available by month 18, while final versions will be available by month 36.

Section 2.3 describes how each of these objectives has been assessed,

2.2 Response to Year Two Project Report

The Year Two Project Review [1] identified the following potential sources of risk, which needed to be addressed during Year Three:

- the integration of platform kernel and plug-ins, in particular with respect to the combined use of several plug-ins;
- the independence of the methodology work from the platform development and case studies, which should in fact be driving the research in the methodology;

- the dissemination to external users.

The above risks have all been addressed by the project and the mitigations are described in the associated Work Package deliverables. The issue of integration is also covered in this report as follows:

- Case studies have specifically assessed the following objectives, which are focussed on the integration of platform kernel and plug-ins:

    a *Checking the scaleability of the system as its functionality is extended.*
    b *Checking the impact of legacy (sub) systems.*
    c *Checking the scaleability of the system with respect to the size and complexity of the models.*
    d *Checking the sensitivity of the methodology to changing requirements with respect to the models.*

- Each plug-in has assessed platform integration.

In addition the following specific points were made in relation to WP 7, evaluation and assessment:

1 There are inconsistencies between the year one [11] and year two [16] reports (in what was evaluated and how it was evaluated) that make it difficult to judge progress. Similarly, there are inconsistencies within the year two report: subjective evaluation with only vague/generic justification, combined with lack of cohesion between the evaluations of the separate case studies, leads one to question the value of the measurement criteria and the grades awarded.

2 In the assessment for the final year, the reviewers need to be able to see answers to the following questions with respect to the case studies and associated tasks and their role in assessing the RODIN system:

    a What was done?
    b Why was it done?
    c How was it done?
    d Was it successful?

3 The assessment in years one and two focussed on assessing each tool against a set of common criteria. This would not be an acceptable way of assessing progress in year three. Furthermore, in years one and two many of the grades were awarded based on qualitative statements without any quantitative evidence. This will not be acceptable in the report in year three where it is clear that the case studies could have easily collected the necessary hard data.

4 To conclude, the criteria for the final evaluation and assessment are inherently different and much more global than those set up at the beginning in D14 [11]. There seems no definition of the final criteria so far. Such criteria will certainly be less of scientific nature than of practical nature, i.e. from the point of view of the software engineers that are external to RODIN and potentially going to adopt it and to include it in their development process. This should be the point of view that drives the evaluation and assessment, and therefore the whole project, in year three.

To address these points, this report:

- Keeps the same sub-sections and topics as in Assessment Report Two, with additional sub-sections to address work package specific issues.

- Presents progress in a tabular form.

- Each work package progress report now provides:

    - A justification for each evaluation grade;
    - A sub-section, which addresses work package specific evidence of quality or effectiveness. This enables evidence of success that may not fit into the generic structure to be presented.
    - A sub-section, which reports on each of the tools used. This indicates how it was used and what feedback was provided to the tool developers. Similarly, each tool developer work package reports on the feedback it received from the case studies, and how it was able to make use of that feedback.

2.3 Measurement Criteria

The Procedures for Technical Review and Assessment [6] identified quantitative measurement criteria to be applied when assessing RODIN work packages 1 (case studies), 3 (Kernel) and 4 (Plug-ins).

Each of the case studies has provided quantitative feedback on these criteria to assess the extent to which Objective O3 (Kernel assessment) and O4 (Plug-Ins assessment) have been addressed (see section 2.1 above).

Objective O1 concerns the collection of a set of re-usable templates and patterns, which tailor the method and tools to the specific needs of each project. Objective O2 concerns the provision of a methodology, which can be supported effectively by the toolset, and lends itself to adaptation to address each project's specific constraint. Case study feedback against these objectives has been extremely positive, however the feedback has inevitably been centred on specific case study issues. As a consequence, the evidence of patterns and templates, which were adopted for each case study, provides the only quantitative measure of these objectives.

The Procedures for Technical Review and Assessment [6] defines the following generally desirable properties, which are applicable to software development methods and tools. For our quantitative assessment, each property has been assessed to identify both the presence and quality achieved during the study.

- **Usability (U)**: methods, tools and templates should be as easy to use as possible, and be as generic as feasible in order to be applicable to other problems and other application domains.

- **Cost-Effectiveness (C)**: compared to existing tools and methods, those developed for RODIN are competitive with respect to the amount of computing and manual analysis needed for a given benefit.
- **Openness (O)**: ability of other researchers and commercial companies to apply the tools and techniques from RODIN in their own domains, without onerous intellectual property constraints.
- **Extensibility (X)**: the methods, tools and templates should admit extension so that other researchers and commercial companies can make their own changes to the items to make them more useful in their developments.
- **Soundness (S)**: the methods and tools used should be as correct as possible, and this correctness should be justifiable. The case study requirements and specifications should be consistent and coherent. The case study implementations should be correct with respect to their specifications and requirements.

2.4 Approach

The assessment was conducted using a set of questions relevant to each of the work packages. These questions were listed in the Procedure for Technical Review and Assessment [6] and updated as a result of year one review and year one experience. The questions vary for each WP1 case study. A single set of questions was used for the assessment of the open source tool kernel (WP3) and the various plug-ins in (WP4).

Each question has codes attached, which link them back to the measurement criteria in Section 2.3. These codes are listed in brackets next to each question, e.g. (C, S) or (S, X).

The following products were assessed:

- Case study 1: Formal Approaches to Protocol Engineering
- Case study 2: Engine Failure Management System Assessment
- Case study 3: Formal Techniques within an MDA Context
- Case study 4: CDIS Air Traffic Display Information System
- Case study 5: Ambient Campus Assessment
- Open source tool kernel
- Plug-in: Mobility model checker
- Plug-in: ProB model checking and animation tool
- Plug-in: Brama
- Plug-in: UML–B
- Plug-In: B2RODIN

For each product, there is an associated producer and a reviewer; the former was responsible for carrying out a self-assessment and the latter for providing an independent view of its validity. The appointment of independent reviewers is a change to the original assessment process based on experience from the first year of RODIN. Reviewers were chosen with appropriate knowledge of the area in which the product was applicable, but with no link with the producers.

The review process conducted was as follows:

1 The producer and the reviewer agreed the criteria, making use of the generic criteria defined in Procedures for Technical Review and Assessment [6]. Where necessary customised criteria were also agreed which were better suited to the current state of the products being assessed.

2 The producer self-assessed his/her work based on the generic/customised criteria. The output from this process was a written assessment report for each item of work.

3 The reviewer then carried out a sufficiently independent assessment to validate the self-assessment and added their findings to the written self-assessment.

4 Praxis collated the validated written assessments into this overall assessment report, D34.

The producers and reviewers were asked to provide answers to their corresponding set of questions. These answers consisted of two parts:

1 Numeric Grade between 0 and 5. The following interpretation was used for each of the grades:

[0] - Failure (Total failure)
[1] - Weak (Unsatisfactory)
[2] - Average (Acceptable but with room for improvement)
[3] - Good (As good as, up to an expected standard)
[4] - Very Good (Better than existing)
[5] - Excellent (Satisfaction in every respect)
[N/A] - Not applicable

N/A is intended for use where it is too early to form a judgement; e.g. progress has been made on a component but it is not yet in a state where the measurement criteria can appropriately be applied.

2 A written justification giving the rationale for the chosen grade together with any guidance on how to interpret the answer.

Section 3 below summarises the responses received and relates them to the overall project objectives.

SECTION 3 OVERALL ASSESSMENT RESULTS

3.1 Overview

To facilitate comparison with previous assessment reports, each of the following sections adopts the structure proposed in the assessment procedure report [6].

Section 4 provides a summary of the qualitative feedback from the Methodology.

Section 5 through to Section 7 provide more detailed responses to the assessment questionnaire for each of the reviewed areas. The progress reports in this document provide only a very brief summary of each RODIN constituent activity. Further detail can be found in the other respective year three deliverables [18,19,20,21].

As a result of the case studies, we have achieved good coverage of the overall project objectives, c.f. §2.1. The preliminary report on methodology [9] provided a sound set of guidelines for the rigorous development of complex system (objective 2) at the end of year one. The case studies have subsequently adapted these guidelines to meet their specific project needs.

Table 1 provides a summary of the scope of assessment covered by each case study report in Section 5.

	CS1 Formal PE	**CS2 FMS**	**CS3 MDA**	**CS4 CDIS**	**CS5 Ambient Campus**
O1: Templates and patterns provided	✓	✓	✓	✓	✓
O3: RODIN Kernel assessed	✓	✓		✓	✓
O4: Plug-Ins Assessed					
Mobility checker					✓
ProB	✓	✓	✓	✓	✓
Brama		✓			
UML–B	✓	✓	✓	✓	
B2RODIN		✓		✓	✓
Integration Assessed	✓	✓		✓	✓

Table 1: Summary of coverage of RODIN objectives by each case study

3.2 Overall Results Summary

A summary of the quantitative results of our assessments is given in Section 3.3 below. Table 2 summarises the qualitative results from each product of the RODIN project.

Area	Achievements
Methodology (Section 4)	The methodology reports [9,15] have delivered a comprehensive set of generic guidelines, which have been applied successfully across all RODIN case studies.
CS 1 (§5.1)	The case study has demonstrated the feasibility of integrating formal methods into an existing industrial software development process. The automatic refinement from Lyra/UML-2 models into the formal framework added significant value to Nokia process.
CS 2 (§5.2)	Two prototype products have been developed, a Failure management system and a production acceptance test system. In both cases UML has been integrated with Event-B. This case study has made the widest use of RODIN plug-ins. ATEC conclude that the major benefit of the method and toolset is through requirement refinement, rather than specification translation into implementation. At times the toolset currently lacks the maturity to support certain application types.
CS 3 (§5.3)	The case study has focussed on the integration of the RODIN methods and tools with the OMG Model Driven Architecture (MDA) framework. Latterly the focus has been on the validation of the RODIN platform, tools and methods. UML–B and ProB have been used extensively. It concludes that the RODIN toolset has made use of Event-B possible, with ProB plug-in adding value in the validation of models. However, certain aspects of Event-B are too strict/abstract for use by a team, which is inexperienced in formal techniques.
CS 4 (§5.4)	The CDIS case study has concentrated on assessing the RODIN platform against alternative tools and methods. The original VDM CDIS models have now been ported to the RODIN platform It concludes that RODIN is more productive and easier to use than other similar tools such as B4free and AtelierB. Furthermore, the methodology's modular approach to system modelling, using fewer constructs than standard B, should simplify deployment.

Area	Achievements
CS 5 (§5.5)	Case study five has developed a novel approach for modelling and verifying the correctness of complex mobile agent systems, which could not be captured by any existing languages. Throughout the RODIN project, three ambient campus scenarios have been developed using this Context-Aware Mobile Agents (CAMA) framework. The major achievement in year three was the development of a single hybrid (Event-B together with a process algebra with mobility characteristics) high-level programming notation that is capable of capturing both the behavioural and functional model of agents.
Kernel (Section 6)	A public version of the Event-B RODIN open tools kernel has been successfully delivered. Feedback from the case studies has been a major influence on the final product. The case studies and plug-in providers report that the tool is easy to use, scaleable, and easily extended.
Mobility Checker (§7.1)	The mobility checker plug-in has been properly integrated into the RODIN platform and supports the automatic verification of mobile agent systems. It has been used extensively by case study five.
ProB (§7.2)	The ProB plug-in has been widely used across the project (see Table 1 for details) with very positive results. A number of case studies continued to use the pre-RODIN version of the tool, as the Eclipse plug-in doesn't currently supports all features.
Brama (§7.3)	The Brama plug-in has been used by case study two, and provides a useful animation capability, which supports model validation. During year three the plug-in was improved and fully integrated into the RODIN platform.
UML–B (§7.4)	During year three the UML–B plug-in was redesigned via an independent meta-model. As a consequence it has now been used by most case studies (see Table 1 for details). The results of the study indicate that integration between the tools is very good, and UML–B is significantly quicker to understand and modify than Event-B.

Area	Achievements
B2RODIN (§7.5)	The B2RODIN has been used by three case studies (see Table 1 for details). During year three the plug-in was improved and fully integrated into the RODIN platform. The plug-in is reported as providing a robust means of transferring AtelierB models, which conform to the Event-B language, onto the RODIN platform.

Table 2: Qualitative summary of results from each RODIN product

3.3 Summary of Quantitative Metrics Assessment

The following charts summarise the quantitative results, which are derived from the more detailed summaries in Section 5 through to Section 7.

Each assessment score, 0 to 5 and not applicable (N/A), was allocated against the case study, kernel, and plug-ins used by the case study as appropriate.

Figure 1 shows the number of times each assessment score was achieved. Figure 2 shows, for each of the assessment criteria: Usability (U), Cost-effectiveness (C), Openness (O), Extensibility (X) and Soundness (S), the percentage of the results that were assessed with a particular score.

Figure 3 and Figure 4 show the distribution of scores allocated to each case study and kernel/ plug-in respectively.

Finally Table 3 summarises the case study responses to the integration criteria (c.f. §2.2), which were established at the end of year two.

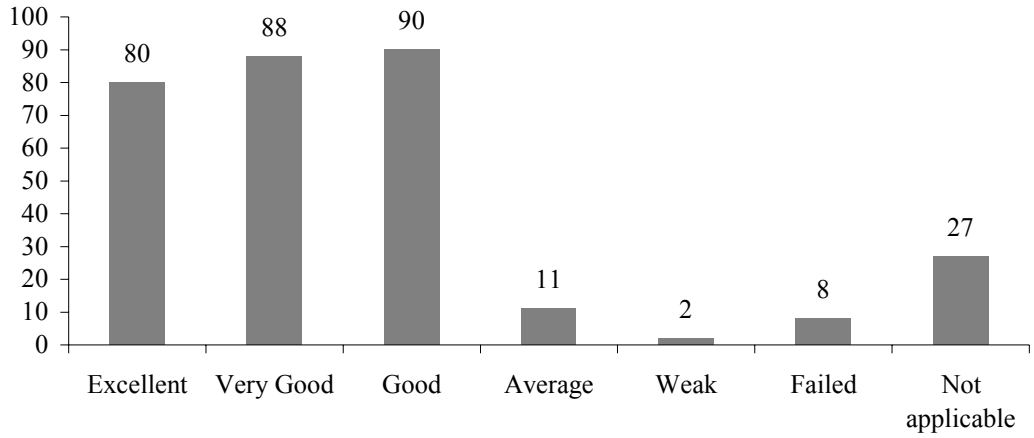


Figure 1: Chart showing distribution of assessment scores

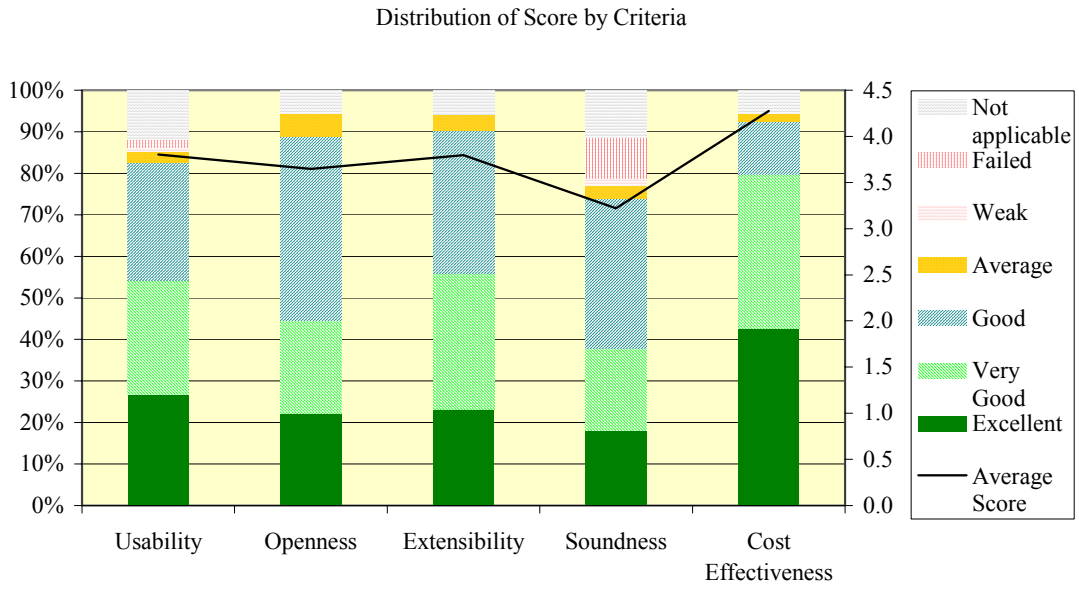


Figure 2: Chart showing distribution of scores across assessment criteria

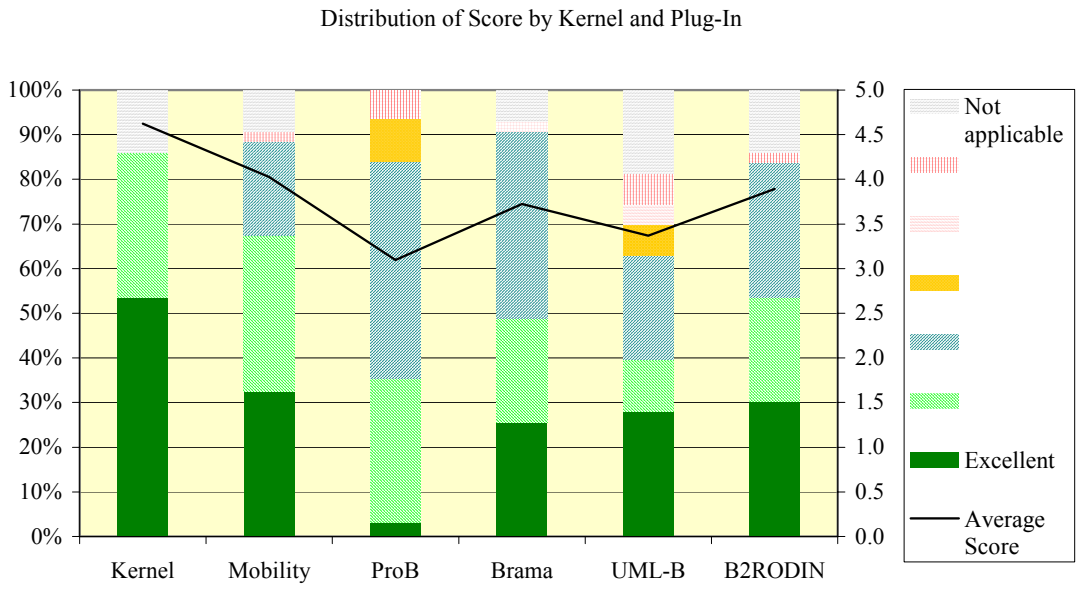


Figure 3: Chart showing distribution of assessment scores across case studies

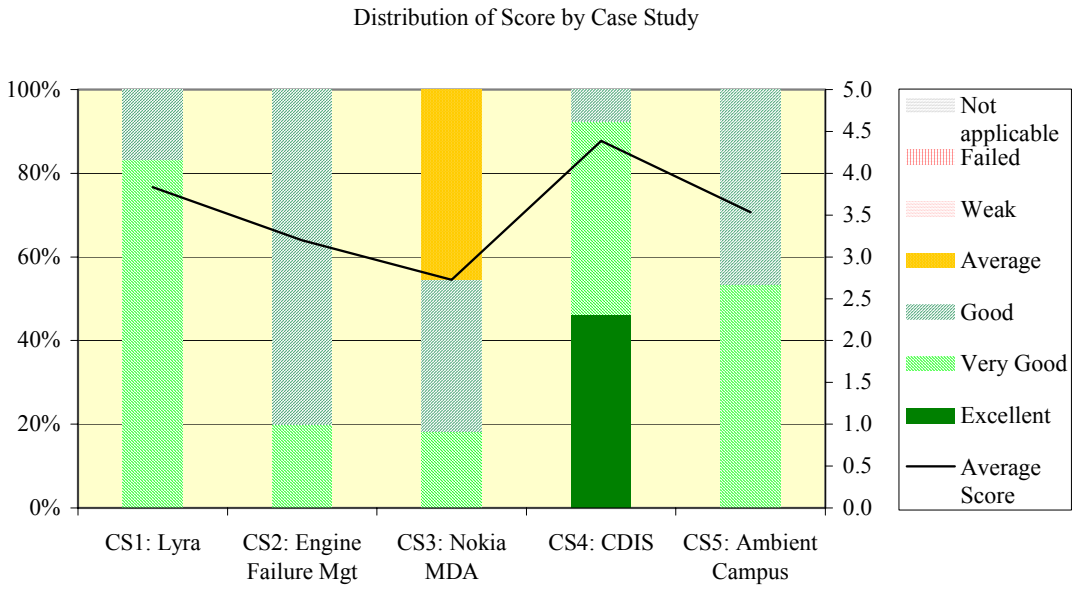


Figure 4: Chart showing distribution of assessment scores across platform

		Score		
Integration criteria	**Case Study**	**3**	**4**	**5**
a) scaleability as functionality extended	CS1	✓		
	CS2		✓	
	CS4	✓		
	CS5		✓	
Count of scores awarded – a)		*2*	*2*	
b) impact of legacy (sub) systems	CS2	✓		
	CS5	✓		
Count of scores awarded – b)		*2*		
c) scaleability w.r.t. model size and complexity	CS1		✓	
	CS4			✓
Count of scores awarded – c)			*1*	*1*
d) sensitivity to changing requirements	CS4		✓	
Count of scores awarded – d)			*1*	
Count of scores awarded		**4**	**4**	**1**

Table 3: Integration assessment scores for each Case Study

3.4 Assessment overview conclusions

In this final year of the project, a summary of the extent to which each objective has been achieved is presented below:

O1 The case studies (WP1) should provide a collection of reusable development templates (models, architectures, proofs, components, etc.).

This objective has been achieved, with case studies 1, 2, 4 and 5 all delivering useful results, and in particular all case studies have provided a set of patterns and templates for future use. See §5.3 below for details regarding case study three.

A brief qualitative assessment of the project's performance against this objective is provided in Section 4. Further detail is available in the Final report on Case study development, D26 [18], and the Case study demonstrators deliverable, D27 [19].

Case Study	Templates and patterns developed
CS1: Formal PE	Lyra UML2 profile. Structural consistency guarantees among Lyra phases. Patterns enabling verification of Lyra decomposition phases. Lyra fault-tolerance templates. Scenario-based Lyra model testing. Automated Lyra system design flow.
CS2: FMS	Verification and Validation methodology, covering animation validation, model validation, model verification, animation verification and interactive proof. Generic problem domains in UML.
CS3: MDA	Formal transformation of platform independent models (PIM) to platform specific models (PSM). Use Case / SDL development. Requirements change addressing fault tolerance.
CS4: CDIS	New Event-B method. Support for structured data. Event splitting and refinement. Productivity improvements.

Case Study	Templates and patterns developed
CS5: Ambient Campus	Context-Aware Mobile Agents (CAMA) framework, comprising: fundamental abstractions and property verification support. Methodologies and frameworks to support mobile agent systems (MAS), which include a number of refinement patterns.

Table 4: Reusable patterns and templates delivered by each case study

O2 The Methodology (WP2) should provide a set of guidelines on a systems approach to the rigorous development of complex systems, including design abstractions for fault tolerance and guidelines on model mapping, architectural design and model decomposition.

The year one methodology report, D9 [9] provided a comprehensive set of generic guidelines, which were applicable to the RODIN case studies. J-R. Abrial's new book, "Modelling in Event-B: System and Software Design" (see §4.2 below for more detail) forms the major input to the generic RODIN methodology. This has been used extensively on all case studies, with the example proofs being checked by the RODIN tools.

This work was further enhanced by the year two methodology report [15], which addressed a number of key outstanding issues.

Feedback from the case studies on the applicability and effectiveness of the methodology has been very positive. As a consequence, we claim that this objective has also been achieved.

Again, a brief qualitative assessment of the project's performance against this objective is provided in Section 4. Further detail is available in the report on assessment of tools and methods, D28 [20], and the final methodology report, D29 [21].

O3 The Tool Kernel (WP3) should deliver an open platform, which supports extensibility of the underlying formalism via integrated tool plug-ins.

The evidence from the case studies clearly demonstrates that this objective has been achieved. Each Case study has made extensive use of the Tool kernel with favourable assessment scores, particularly with regard to its usability, extensibility and cost-effectiveness. The kernel platform supports plug-in development very effectively. As a consequence five plug-ins have been actively used on the RODIN case studies.

The quantitative assessment results presented in Figure 1 indicate that the platform is better than existing platforms to support formal specifications.

O4 WP 4 should deliver a collection of kernel plug-in tools for model construction, model simulation, model checking, verification, testing and code generation.

The five plug-ins, which have been actively used by the case studies, address:

- Model construction: UML–B, B2RODIN
- Model simulation: Brama
- Model checking: B2RODIN, ProB, mobility checker
- Validation and testing: Brama, ProB

Thus all aspects of this objective, with the exception of code generation, have been addressed.

Figure 4 summarises the effectiveness of the RODIN plug-ins, and again demonstrates that the platform and tools provide better support for formal specifications than existing platforms.

Evidence from the case studies gives a clear indication that the approach is capable of delivering real benefit for industrial scale software developments.

Figure 2 clearly illustrates the positive assessment feedback received on the project, showing that for all the assessed criteria, 55% returns indicated that the RODIN provides an improved support environment for formal software development (score 4 or 5). There were no significant points where the assessment was “weak” or “average”. Only eight areas were assessed as “failed”; all were related to the level of integrity and verification applied to plug-ins.

Furthermore, the integration assessment results (c.f. Table 3) indicate that the platform is capable of extension to support other key aspects of a large-scale formal system specification.

SECTION 4 METHODOLOGY ASSESSMENT

4.1 Introduction

Earlier work package reports, D9 [9] and D19 [15], have examined progress in each of the case studies in relation to the RODIN methodology. This has led to questions and/or discussions about the use of aspects of the proposed methods within RODIN.

As a result of this work, we have concluded that any successful formal method will always need to be integrated into the methods of industrial user organisation. As such deployment of the method is critical to the success of a formal methods project.

For the RODIN project, we now have a core RODIN method, which has its manifestation in five different deployments, as described in Sections 5.1 to 5.5.

4.2 The (generic) Event-B methodology

As reported in the Final report on Methodology (D29) [21], Jean-Raymond Abrial is close to finishing "Modelling in Event-B: System and Software Design". This new book summarises the development method for Event-B and contains fully worked examples whose proofs have been constructed and/or checked with the RODIN tools.

The emphasis throughout the book (and all RODIN methods) is on "correctness by construction", (CxC). In summary, the overall plan of CxC is to begin with an abstract model and to introduce design decisions as refinements. At each stage of refinement, one proves that the previous specification will be met if the subsequent ones are fulfilled.

Abrial's book contains eloquent advice on:

- abstraction and refinement and the layering of design decisions;
- the structuring of requirements;
- proving properties of (abstract) models as a way of increasing confidence that the delivered system will meet the expectations of its commissioners and/or users. Here the RODIN tools provide added value by also offering links to simulation.

The major technical innovation in moving from B to Event-B is the introduction of *guarded events*. These can present "deadlocks" and appropriate proof obligations have to be discharged to establish that this is not the case. The RODIN platform and tools provide enormous help to the user of Event-B.

The wide range of examples in Abrial's new book makes it easier to relate the method to new application areas. There are also chapters on formal development of "sequential programs", "concurrent programs" and "electronic circuits".

Overall, the feedback from the case studies on the usefulness and general applicability of this generic method has been extremely positive.

4.3 Case Study Feedback

As noted in §4.1, any successful formal method needs to be integrated into the methods of its industrial users. The RODIN project specifically aimed to face this issue by undertaking five separate case studies, each with different characteristics. For further details see the Final report on Methodology (D29) [21].

Case study one's investigation into the use of RODIN in "protocol engineering" has paid particular attention to the integration of RODIN with the Lyra method.

Case study two's work on engine failure management, addressed use of the method by staff with limited formal methods experience. Part of this work has examined use of the method during the verification and validation project lifecycle stages.

Case study three examined the use of the method by a major industrial partner, Nokia, and how formal techniques fit with Model Driven Architecture.

Case study four, CDIS, was able to compare the results of using the RODIN method with the formal methodology, VDM [89], which was adopted by Praxis at the outset of this implementation in 1992. Although the resultant model presented in Event-B is both much clearer and more tractable, we do not claim that the entire difference is down to the RODIN language and methods. It is always possible to improve on a formal model; in this case the improvement has been dramatic by clever factoring of ideas.

Finally case study five attempts to address issues concerned with ambient systems. Here the main faults, which need to be tolerated, are concerned with transmission errors/failures. This work has led to the use of "patterns" to reduce the level of formal proof necessary for each new application.

We therefore feel that the RODIN project demonstrates that the generic method is both capable of supporting a wide range of industrial strength projects, and can be adapted via re-usable patterns and templates to integrate with existing methods in an industrial context.

SECTION 5 CASE STUDY ASSESSMENTS

5.1 CS1 – Formal Approaches to Protocol Engineering

5.1.1 Introduction

The case study investigates the use of formal methods (in particular, refinement and model checking and model-based testing techniques) for industrial-scale development of telecommunication systems and communication protocols. The work of the case study focuses on formalisation and validation of the *Lyra design method*, an industrial-strength domain specific design method example, developed at the Nokia Research Center.

One of the main objectives of the case study is to integrate formal methods into the existing development process at Nokia through automation of the refinement steps in the design flow. This has been achieved through automatic translation of Lyra/UML-2 models into the formal framework. The process includes:

- Automated refinement with in-built correctness and consistency checking, and
- Enhancement of the models with automatically generated fault-tolerance behaviour.

Nokia perceive the major criteria for evaluating the success of RODIN to be:

- The degree of automation achieved. This indicates the applicability and usability of the enhanced development process in an industrial setting.
- The enhanced development flow. This provides added value to the industrial system and product development process.

The objectives of the case study (from the Description of Work [1]) are to:

a Investigate the benefits of using refinement approach versus algorithmic verification to verify system decomposition and composition.
b Investigate model reduction techniques and proof methods for data abstractions and the use of model checking to verify correctness of system components.
c Investigate applicability of formal reasoning techniques about fault tolerance in this application area.
d Validate top-down and bottom-up formal techniques and supporting tools.

5.1.2 Current Status

Year	Achievement	Objective	Papers
1	Traceable requirements document for the positioning system case study, suggested by NOKIA	c), d)	[4]
1	Specification and refinement patterns reflecting essential Lyra models and transformations	a), d)	[25]
2	Modelling fault tolerance mechanisms in formalised Lyra-B models	a), c), d)	[26]

Year	Achievement	Objective	Papers
2-3	Developed methodology for verification of the consistency of provided Lyra/UML2 models	a), d)	[24]
2-3	Developed methodology for model-based testing of Lyra models and transformations	a), b), c)	[23]
3	Extension of the developed specification and refinement patterns (with incorporated fault tolerance mechanisms) to model parallel execution of services	a), c), d)	[27]
3	A prototype of model-based testing plug-in	a), b), d)	
3	Automatic support for translation of Lyra/UML2 models into Lyra-B models, creating automatic system design flow	b), d)	

Table 5: Case Study 1 - Annual achievements against objectives (c.f. §5.1.1 above)

5.1.3 Progress since Year 2 Assessment

In the final year of the RODIN project our work on the case study has progressed in four dimensions.

1 In the work related to formalisation and the theoretical basis for automated refinement, the Lyra specification and refinement patterns, including the incorporated fault tolerance mechanisms, have been extended to cover modelling of services executing in parallel.

2 To adjust the theoretical approach into an industrial-scale development framework and to allow MDA-like model transformations to implement automated refinement, the existing Lyra/UML2 profile has been enhanced further to include all definitions and constraints related to the developed refinement patterns.

3 For the model-based testing of Lyra models and model transformations, a methodology and a plug-in prototype has been developed.

4 To implement the developed approach with the RODIN tool platform and demonstrate the enhanced development process in practice, and also to address the Year 2 reviewer recommendations [1], the Integration Plan for CS1 has been developed.

This has resulted in the development of an automated tool chain for Lyra-B. Mapping of Lyra/UML2 models into UML/B concepts together with consistency checking of structure and behaviour is now supported by the RODIN tool platform. The generated B models and their implementations will be used as the input for the model-based testing plug-in.

Furthermore, the generated models describing the “basic valid behaviour” are used as a basis for implementing the “correct-by-construction” design paradigm, i.e. enhancing them with fault tolerant and parallel behaviour.

5.1.4 Contribution to the Development of Platform and Plug-ins

RODIN Platform

The RODIN platform has been used for the formal development and verification of essential Lyra models and transformations. The following assessment is based on the few months' experience with platform version 0.7.4.

Event-B view: The interface is nicely structured and easy to use. The different windows are well thought out and provide a good overview of projects. The wizards for adding events etc. can be very useful when making big additions at the same time. Buttons for adding and removing are also very useful when editing the machines. However, it is a bit strange that not all operations have buttons. For instance, adding witnesses or refine events, as well as removing events has to be done via the right-click menu. Overall, most of the things you need are in plain sight and easy to find.

Prover view: Again, this provides a nicely structured, easily understood and navigated view. The automatic prover handles many trivial proof obligations. However, many recurring, simple proof obligations seem to be quite troublesome and often require manual proving. For example, proving that a set is not empty is usually a major problem. On making a small change to an invariant, you also get a huge number of broken proofs that have to be dismissed manually, even though the automatic prover should have been able to check through these on its own much quicker. Fairly often the automatic prover takes a wrong turn at the very first node, resulting in a very hard proven tree, while even the p0 predicate prover would have been able to prove the initial proof obligation.

Features: RODIN is an open platform only having the core functionality; the plug-ins should take care of the rest. Still there is a feeling that some features like animation should be available when you want to test your machine.

General With some experience with B and Eclipse it's easy to get started with RODIN. A few hours of testing features and generally getting to know where to find the things you need is all that's needed to get a decent start. An easily available Event-B language manual would improve usability. This would help awareness of Event-B specifics, e.g., there is no point in trying to make a sequence, or that a carrier set can't be empty.

This case study has reported that a major problem with the version of the RODIN platform used is the lack of redundancy in handling the files where the machines are saved. In particular, on several occasions when RODIN hung or crashed while building a project, the entire project was ruined (with no way to import data). Unless RODIN terminates properly, the workspace cannot be reused until the Eclipse metadata is deleted. (It should be noted that this problem has not been reported on other case studies.)

The error messages from syntactical errors are in most cases very unclear and hard to understand. The guard or action, which caused the error, is identified; no further assistance is provided.

RODIN seems to run a considerable number of provers, some of which may not be required, when building a project; this causes building to take a long time. A facility is required to suspend proof until the user decides the model is ready.

ProB Plug-In

The ProB plug-in is used in combination with the model-based testing plug-in.

The RODIN ProB plug-in has proved to be very useful for animating Event-B specifications. The graphical user interface of ProB plug-in is quite intuitive and user friendly. It is easy to animate and generate execution traces of Event-B specifications. However, there are still a few bugs in the prototype version, which does not fully support the Event-B language.

We have used the ProB plug-in in the context of developing the model based testing (MBT) plug-in. The MBT plug-in uses the ProB engine to generate execution traces. ProB is easy to configure and use as an independent plug-in for the RODIN platform. However, the plug-in to plug-in interaction is not well defined. There was very little support available for such a purpose. The lack of documentation and unavailable application programming interface (API) is also a major concern.

UML–B Plug-In

The UML–B plug-in is used for translating Lyra/UML2 models into the corresponding B specifications.

Once the platform was installed, installing the UML–B plug-in was straightforward. UML–B appears as a separate perspective in the Eclipse environment. Similarly, as in modelling in the RODIN platform, a project in UML–B has the corresponding nature, which makes it easy to distinguish between the other types of projects (e.g. Event-B).

Modelling in UML–B was less intuitive at the beginning, despite the solid knowledge of UML. The main difficulty in modelling in UML–B was caused by a quite rigid design flow supported by the modelling tool. Namely, the design should start with a package diagram, then we should attach appropriate class diagrams to these packages, and then, possibly, we could attach a state chart to classes. When the flow deviates from the required flow, the tool exhibits some unexpected behaviour. However, a manual would help avoid situations like this. In response to our comments, the tool has since been modified to enforce this design flow.

Another difficulty was typing attributes of classes in class diagram. It was less clear immediately how to (and if it is possible) introduce new types, different from those already pre-specified. Creating a state chart was intuitive enough; however, we would expect more support for state chart refinement. There are some features that are more difficult to understand (e.g. different types of treatment of states), especially for novices.

Our work on the case study resulted in few requested features. UML–B support of refinement was the most important in the list of these requests. We needed support for refinement of states on a state chart, but also of refinement of classes in a class diagram.

Overall, the tool became more mature during the case study development, and most importantly, more stable. With a proper guidebook, it can be adopted quite fast and used in developments. However, we believe that it is necessary to have prior knowledge of Event-B in order to undertake successful developments in UML–B, although the Event-B seems almost transparent.

5.1.5 Contribution to the Integration Objectives

a) Checking the scaleability of the system as its functionality is extended

CS1 has used the new RODIN toolset (the platform, and the UML–B and the Pro-B plug-ins) for automated Lyra/UML2-to-B model transformations and for generation of automatically refined models, which are "correct-by-construction". The experiments have not indicated any scaleability concerns related to the RODIN platform. Integration of the model-based plug-in tool, developed in CS1, has shown that it could be quite difficult to implement interactions between separately developed plug-ins. In particular, the model-based testing plug-in depends on outputs (execution traces) produced by the ProB plug-in. However, missing API and appropriate documentation makes it really difficult to establish the connection between the plug-ins.

Grade: [3]

c) Checking the scaleability of the system with respect to the size and complexity of the models

Most of the developed formal B models are in the form of specification and refinement patterns that can be instantiated easily during the "correct-by-construction" development process. We expect this to increase their scaleability. The instantiation and managing large data sets causes some concern.

CS1 uses a set of Lyra/UML2 models for the Position Calculation System and PCAP (Position Calculation Application Protocol) to trial and evaluate the feasibility, applicability and scaleability of the developed approach. These models are representative examples of the size and complexity of the models produced in an industrial development process by a system designer or a small team of designers. In CS1 the RODIN tools have been applied for models of this size and complexity. The experiments show that the RODIN platform performs well at this level.

The Lyra method focuses on the description of system structure and behaviour. Data (included mainly in the message parameters) has been encapsulated into abstract data structures. The underlying idea is to use different tools and methods, specific for data handling, in this area. Data handling has been left as a minor issue in CS1; only the data, which directly affects the system behaviour, has been treated in more detail in the

formalization. Therefore, scaleability of the RODIN platform in instantiation and managing large data sets has been outside the scope of CS1.

CS1 has not addressed the scaleability of the RODIN platform in a distributed development environment (geographical distribution, large collections of separately developed system parts). In supporting a compositional development concept in industry, scaleability of the design tools and approaches at the module/component level is regarded as the primary goal. Module/component integration, both regarding the system composition and the tool environment, are separate concerns that need to be addressed separately.

Grade: [4]

5.1.6 Case Study Specific Metrics

1 How well do the developed concepts, methods, and tools fit with the existing development framework? (U)

The goal of this metric is to assess seamless integration of formal methods and tools to existing industrial development process. To realize this, an invisible link between the existing development framework and the RODIN platform should be provided. The created link should provide an engineering environment easy-to-use for "non-formalists".

The target has been to:

- link the first three (out of four) Lyra phases into the RODIN platform, and
- link the Lyra method with the model-based testing methods and tools in the RODIN platform.
- To develop a small prototype for automated model transformations as a proof-of-concept for the automated "correct-by-construction".

These experiences and trials provide information for estimating the feasibility and applicability (w.r.t. competences, resources, etc.) of the approach in an industrial context.

During the RODIN project, the following methods and tools were developed to integrate RODIN and Lyra/UML2 approaches:

- Lyra/UML2 and B profiles and meta-models,
- tools for transforming UML2 models into B,
- Lyra B specifications and refinement patterns,
- Model-based testing (MBT) plug-in.

Currently, and mainly as a result of work done during year two of the project, those three phases, namely Service Specification, Service Decomposition and Service

Distribution are linked to RODIN platform. This allows the use of RODIN tools and methods in a more rigorous development flow without significant competence renewal. UML2 models from these Lyra phases are translated (using ATL, or ATL - U2B combination) into the corresponding B models. The translation process is based on the use of meta-models, which allows flexible enhancements to the methodological framework later on. During the translation the syntactic consistency of input models is checked. Currently the linking only works in one direction, so that Lyra UML models and other required design information are used as inputs for the RODIN model transformations. Therefore, the design errors discovered with the RODIN platform cannot be traced back to UML2 models. Traceability was not included in RODIN targets, but should be considered as a research item for future actions. The tool chain and conceptual work flow for automated model transformations will be demonstrated with a small prototype implementation.

Model-based testing is not yet fully linked. Currently, the theoretical basis for model-based testing of Lyra B models has been developed. Implementation options for Lyra – MBT linkage to the RODIN platform either by using the ProB plug-in tool or an external model-checker tool have been examined.

Based on the experiences in successfully trialing this fully automated approach, the anticipated need for competence renewal inside the company is minimal. Also, the threshold for adapting the "invisible" formal methods framework into production level system development processes should be low.

The target in trialing and demonstrating the integration of formal methods and tools to existing industrial development processes and prevailing practices has been well achieved and exceeded in certain areas. On the other hand, the progress in the MBT side has not met the target level.

Grade: [4]

2 *How much support does the RODIN approach provide for a more rigorous development process? Specifically, how many new tasks in the development process can be tackled using the methods developed in RODIN? (C)*

The goal of this metric is to evaluate large-scale applicability of the RODIN methods and tools in an industrial development process, in order to provide significant amount of added value to the whole industrial development in terms of improved quality and R&D efficiency.

During the RODIN project, the following methods have been developed to achieve this goal:

- the constraints for checking inter- and intra-consistency of provided Lyra UML models,

- B-Lyra specification patterns, enhanced to include fault tolerance mechanisms,
- tool support to ensure consistency and verify correctness of development steps.

The target is to provide support for more rigorous development in the first three Lyra phases (out of four). The tools and methods in the RODIN platform should be used to assist and automate the more rigorous system development flow. RODIN work, including concept definitions and more detailed specifications for later implementations, should give reference for estimating the anticipated quality improvement and workload reduction.

The behavioural consistency of the refinement steps is validated using the RODIN platform. The consistency checks are based on meta-models and specifications for refinement patterns, including fault-tolerant behaviour. Consistency checks cover both the refinement steps between the first three Lyra phases (inter-consistency rules) and between the modelling concepts inside a certain Lyra phase (intra-consistency). The refinement patterns provide a basis for automating the industrial engineering workflow producing designs "correct-by-construction". Fault-tolerance mechanisms (in combination with service decomposition and distribution) are automatically incorporated into the refinement process. Fault-tolerant behaviour has not been considered much in the original version of the Lyra method nor in the UML models given as input for the RODIN work initially. This part of the RODIN work enhances significantly the original method description.

Based on the interviews with industrial engineers, their estimation is that the productivity could be increased four-fold by launching this kind of an automated and more rigorous workflow. Also, correct-by-construction designs would reduce significantly the need for testing, which currently constitutes approx. 70% of the development time.

The target for this task has been well met. Although the handling of user data was included in the original goals, this is a challenging research item of its own; it has left for future development.

Grade: [4]

3 *How much support does RODIN provide for automation of the development process? Specifically, how many new tasks in the development process can be tackled using the methods developed in RODIN? (C, X)*

The goal of this metric is to assess the automation of a "correct-by-construction" design approach in an industrial development process. How much quality is increased due to ensuring correctness by design? How much R&D productivity is enhanced through:

- reduced overall work effort in an industrial development process (automation),
- reduced need for testing due to design-time validation,
- direct use of design models in testing, e.g. model-based testing and automated test case generation.

Automation of model transformations (Lyra UML models to B or UML–B) with the ATL tool has been automated at the required level – a small prototype implementation will be demonstrated. The refinement process verifying the behavioural consistency between the Lyra phases is fully automatic. However, it is based on using specification and refinement patterns, and indicates the future need for additional support for model instantiation. For MBT some initial simple examples exist for test generation using ProB plug-in.

This task overlaps with the previous tasks. The level of automation achieved in these proof-of-concept trials and planned as near-future research items confirm the estimations for R&D productivity increases that we have presented in the evaluation statements of the previous tasks.

Grade: [4]

5.1.7 Conclusion

Our evaluation has shown that the work on the case study has progressed according to the initial plan and achieved the expected objectives. The achieved results allow integration of formal methods into the existing development process at Nokia through automation of the refinement steps in the design flow and automatic translation of Lyra/UML-2 models into the formal framework. Nokia considers the achievement of such automation as having added significant value to industrial system development.

5.2 CS2 – Engine Failure Management

5.2.1 Introduction

The engine failure management system provides a protective wrapper to the Engine control subsystem, protecting it from failures in its system inputs and so enhancing the dependability of the control system. It detects failures, and then manages these failures in order to provide the control subsystem with an acceptable input or graceful degradation of behaviour. (Deliverable D2 [3] and the initial RODIN presentation provide a more detailed definition.)

Additionally, during RODIN year three, ATEC undertook a further production acceptance test case study (PAT) as part of this work. The PAT case study is described in the deliverable D26 [18] as part of case study 2. PAT provides a configurable system of test specifications for the production hardware of the engine controller.

The objectives of the case study (from the Description of Work [1]) are to:

a Investigate the use of formal methods on engine failure management applications.
b Investigate methods of using formal methods efficiently via reuse.
c Investigate methods for specifying a generic software support package for engine failure management applications.
d Produce prototype products that could be used to support the development of a software support package for engine failure management.
e Investigate benefits of integrating UML and B.

5.2.2 Current Status

Year	Achievement	Objective	Papers	Contributor
1	Initial failure management behavioural model	a)	[28]	Soton, ATEC,
1	Traceable requirement specification of FMS	a)		ATEC
1	Generic model and methodology for developing domain models	a), b), c)	[29,30]	Soton, ATEC
1	ÅA development of behavioural model using classical refinement	a), b)	[31]	ÅA
2	Generic model development Features and develop Context manager to handle instantiations	a), b), c)	[30]	Soton
2	Independent Pilot study investigation assessing methodology and evaluating technology	a), e)	[30]	ATEC
2/3	ÅA further work to convert to event B and UML–B and provide guidelines	a), d), e)	[33]	ÅA

Year	Achievement	Objective	Papers	Contributor
3	Behavioural modelling of generic model	a), b), c), d), e)	[36,79]	Soton
3	PAT case applying RODIN related technology on legacy system for a commercial customer	b), c), d), e)		ATEC

Table 6: Case Study 2 - Annual achievements against objectives (c.f. §5.2.1 above)

5.2.3 Progress since Year 2 Assessment

Failure Management System (FMS) developed by Southampton University (Soton) and Åbo Akademi University (ÅA)

Work has advanced from the static model, which was developed during years one and two, to a dynamic failure management system. The current model is a generic failure management system, which is re-usable and extensible. Re-usability is demonstrated by the integration of more specific contributions by ÅA in section 3.2 of the final delivery report D26 [18].

Further contributions made by Åbo Akademi (ÅA) and Soton can be summarised as:

1. Transfer of pilot study failure management system onto RODIN platform (Soton)
2. Generic failure management system (Soton)
3. Dynamic features of failure management system (Soton, ÅA)
4. Development of failure management system using refinement (Soton, ÅA)
5. Translation of parts of ÅA's classical B models into UML–B (Soton)
6. Integration of ÅA and Soton ideas (Soton, ÅA).

In year three, the focus of the development of FMS was on producing the dynamic part of the existing static model. The model was kept generic and abstract, such that it can be refined later into different more specific applications of FMS – thus enabling re-use. The model is mainly built – using UML–B – out of components with associations and object constraints, using an object-oriented development approach. The generic model can be seen as the composition of a number of functional *features* – detection, confirmation, condition and action – which can be adapted in further refinements.

The FMS case study has seen progress through the academic partners' development of their models using the new RODIN tools.

The formal, classical refinement development of the FMS can be enhanced by the use of UML, in particular, a subset of UML called UML–B [32]. The result of integrating this formal refinement approach into the UML-based development of the FMS is a set of UML–B models distributed through phases of the development process [33]. Each development phase corresponds to a refinement step. It is characterized by a set of UML–B models (class and state chart diagrams) representing the main structural and behavioural aspects of the FMS at the corresponding level of abstraction. To automate

the process of obtaining a formal specification from UML–B models, we use the U2B tool [34], which translates the UML–B models into Event-B. We use the automated Event-B tool support for verifying correctness of our development. The results showed that we were able to prove the correctness of models significantly faster, with higher percentage of automatic proofs than in our previous classical refinement development [35].

The new RODIN tools were used for the validation and verification of the UML–B model. Both the classical ProB model-checker and the RODIN ProB plug-in animator were used to validate the model; the RODIN prover supported model verification.

Production Acceptance Testing (PAT) undertaken by AT Engine Controls (ATEC)

Although ATEC has needed to divert some work from the FMS case study, they have utilised RODIN technology for a new situation, a semi automatic Production Acceptance Test system (PAT). The PAT system tests the hardware platform and in-situ software (which includes an FMS system) for manufactured production units. The requirement and a more detailed description of its development are described in the main report D26 [18].

The challenge was to utilise the RODIN technology to develop a real implementation for a customer operating under commercial timescales. This has been achieved through the development of a static UML–B structural model of the system. This model was used to drive a configurable specification that interfaces with a target system, which uses some existing code. The development environment used the Eclipse and EMF environment shared by RODIN. The new RODIN tools were exercised in some simple modelling of legacy in the domain.

5.2.4 Contribution to the Development of Platform and Plug-ins

Feedback on the Platform and Plug-ins has been given in presentations and directly to developers. Bug tracking and forum websites were also made available.

RODIN Platform

In both cases models have been entered via the UML–B Plug-In, which uses the Event-B platform. Verification of the behavioural models was undertaken on this platform using the platform's static checker and prover.

ATEC perspective (Year 3)

Installation

ATEC initially found the installation of the Platform and Plug-Ins required some knowledge of the Eclipse environment and familiarity with update sites. Initially this was not very intuitive. However, the use of automatic update sites, made later installations much easier. It would be useful to have a more documentation about manual procedures,

which could be followed for installation to the platform, when there are difficulties with automatic updating or further procedures have to be followed. A proposed CD containing a complete installation should resolve any difficulties for the novice.

Long-term support of the installation is a significant issue for the future since ATEC needs to support some products over long life spans, e.g. 10 years or more.

A concern is that future changes in the platform may make models developed on older platform configurations obsolete. Backwards compatibility of the platform for its models and availability of previous platform releases is an issue, which needs to be addressed for any long-term adoption of the technology.

Navigation

The platform is windows intensive and encourages the use of larger displays. The navigation around the platform is menu driven but locating information was not always intuitive. New users may benefit from examples for navigating between different perspectives. Once familiar with the platform, this was not such a problem. However swapping between different perspectives normally required some window adjustment.

PAT Case Study Feedback

The UML–B plug-in supported the entry of the PAT partial specification. The Event-B translation of the partial specification in UML–B allowed the UML–B model to be viewed from the Event-B perspective of the platform, which was significant for viewing the full textual translation (via pretty print) and providing the facility to check and prove the partial specification model. The reactive nature of the prover was useful in identifying problems as they occurred, allowing them to be addressed early. Initially much of the model was entered using the UML–B interface, rather than using an incremental approach. This meant that several line items required proof obligations. However the ability to add and verify functionality gradually is not diminished, as it was useful to identify problems caused by small model changes. The order that the proof obligations were resolved did not appear to impact proof. The model was checked and animated using the ProB plug-in.

The PAT generic editor did not directly involve the RODIN platform but used its underlying technology; this study forms the basis of future research work for UML–B development in the Platform (see comments on UML–B below).

University of Southampton perspective (Year 3)

The RODIN platform is an excellent toolset for specifying critical systems and verifying their correctness; however, due to the complex layout of the platform it is very hard to navigate all the menus and buttons. The platform is based on too many wizards, which eventually confuse and take time.

The following bug report and feature requests are of note:

- Bug Report #1724770 - axioms entered in constant wizard not displayed

  The wizard view does not show straight away, whether a variable or axiom has been added successfully. Thus for some time, when adding a constant and an axiom using the constant wizard, the fact that an axiom was dropped and disappeared from the model was not revealed. Fewer wizards would make the platform more usable and less cluttered.

- Feature Request #1777260 - Propagation of changes

  The RODIN platform does not seem to allow for late model changes. If a change is made to the abstract model, or a low-level refinement, these changes have to be made manually to all subsequent refinements. This is especially time-consuming if new events are added or a type change made. Changes made to more abstract refinements should thus be propagated automatically to subsequent refinements.

- Feature Request #1779420 – Error Messages

  In the event of errors, RODIN generates error messages and warnings. Sometimes these error messages are hard to understand and not very meaningful. However with some experience it will be easier to interpret them and understand their cause.

RODIN Prover

The proving capabilities of the RODIN platform are very useful, and greatly reduce the effort required for at least some POs that are proved automatically. In comparison with B4Free, these would need manual proof, which is tedious.

The interactive prover interface is set up to provide useful information about the PO, which makes it easier to discharge the PO. Hypotheses can simply be added or ignored and automatic rewriting can be applied for some constructs.

Plug-in Integration

Plug-ins are integrated into the RODIN platform using perspectives, which is very useful, because the user can easily switch between them. The plug-ins used in case study two are evaluated below.

Åbo perspective (Year 3)

Launching the application was easy. A very basic tutorial accompanying the tool provides significant help. However, that is the point at which help stopped. There was no available manual on how to access certain tool features or how to proceed with

specification and proving. Recently, a manual has been included in the tool distribution, and this should help its dissemination.

Without the guide, the tool was explored intuitively. This is one of the tool's strengths. Indeed, we found the tool interface reasonably intuitive. Available wizards for creating events, adding variables and invariants give very good guidelines for creating specifications. This, however, is only because we are familiar with the concepts and have experience of other B tools. The main difference is in the organization of a project, i.e., specifications, which adheres to Eclipse standards. Moreover, type checking of specifications is performed each time a project is saved, instead of launching the type checker individually. However, reported error messages are not always clear and understandable. Sometimes they may be very obscure and confusing. Having the list of possible error messages with some guidelines would be extremely helpful.

Each time the project is saved, not only is type checking performed but proof obligations are also generated and discharged automatically if possible. This really saves time compared with AtelierB. The RODIN platform enabled more proof obligations to be discharged automatically, compared with AtelierB, for the same development, i.e., based on the same specifications. However, when it comes to proving the remaining proofs, not everything is as intuitive. Here, there would be real benefit from a detailed proving manual, although the initial guidebook provides a partial explanation.

Having previously used AtelierB Event-Based modelling, translating them into Event-B was fast and intuitive. We noticed few differences and difficulties. For instance, the elements of an enumerated set, i.e., constants, are not distinguished and this has to be done manually. Regarding proof, it is less easy to understand the best approach to case analysis, suggesting a witness for existential proofs, the types of provers to use and when.

Overall, the tool is quite stable, with no unwanted behaviour (e.g. system crash). The acceptance period was rather short. However, initially it would probably be faster if there were a nice, small guide-example for practice.

ProB Plug-In

The pre RODIN ProB plug-in was used in the FMS case study in year two and the RODIN version in the final year. The RODIN version was also used on the PAT case study during year three.

The Plug-in was used on the Dual Case sensor model in the year two pilot study by ATEC [14,16].

The following observations were made.

- Useful tool and intuitive
- Could be improved by:
    a enhanced animation to highlight what has changed from each event

b improved refinement checking
c integrating with same platform as Prover.

In response a RODIN plug-in was integrated onto the same platform as the Prover

ATEC perspective (Year 3)

ATEC applied the ProB Plug-in on PAT later than Southampton (see below). Consequently, they did not experience the same syntax. The points were reported:

- Installation on the RODIN platform is not fully automatic as it required additional installation of some objects.
- The ProB integration onto the same platform as other tools was an improvement, however its selection and use on this platform could be clearer.
- Documentation describing the use of the ProB on the RODIN platform needs to reflect this
- The partial specification model of the PAT was successfully verified using the ProB disprover and animator.
- Not all ProB features have been accommodated on the new platform, notably the model checker. Also the animator did not show a history of its execution. The tool needs to export to classical B to use this feature.
- The disprover gives acceptance of proof if it cannot find any counter examples. This could be misleading where a small state space limit may give acceptance but a larger state space may find a counter example.
- Overall the ProB tool is very useful for the industrial novice. However it needs all the features of the pre RODIN tool to be included on the RODIN platform.

University of Southampton perspective (Year 3)

The ProB plug-in is one of the most used tools of the RODIN platform. It is extremely useful to animate a model to check whether the implementation matches the required functionality. It is very important to animate the model, as this is the only way to make sure that the model performs the way it should. ProB can also be used to aid discharging POs by animating the model with the goal of the PO in mind.

In the early stages, there were slight problems with differences in supported syntax, where some syntax, which was allowed in RODIN, was not yet recognised by the ProB plug-in. This did not hinder model animation, as it was always possible to rewrite the unacceptable statement. Furthermore, these errors were reported to the developers, who addressed them instantly and provided an updated plug-in.

An earlier version of the ProB plug-in used a view to switch to the ProB plug-in. This was improved and a ProB Perspective was made available. In this way, the user can now switch to the ProB animator more easily.

The following feature request is of note:

- Feature Request #1777267 – Model Checking

  A desirable ProB feature is the model-checking capability of classical ProB. Currently, the ProB plug-in can only be used to animate the model – for more advanced model checking functionality, the model has to be exported as a B machine and can then be animated using the standalone ProB tool.

Brama

The University of Southampton assessed this Plug-in.

University of Southampton perspective (Year 3)

The Brama plug-in was not used for the development of FMS due to the lack of detailed documentation and restricted time. The animation part of the tool was, however used on small models, where it was quite successful. A Flash animation was not constructed, as it requires background knowledge of Flash animation creation.

UML–B Plug-In

The pre RODIN UML–B plug-in and its related method have been assessed in the FMS models years one and two. The RODIN plug-in was available in year three and was used in the FMS and PAT case studies.

Feedback from first year

- Use of the uB action and constraint language requires detailed understanding of the translation into B, which depends on details such as multiplicities in associations.
- Graphical feedback to indicate inconsistencies in model.
- The requirement to handle data instantiation of generic models.

Response

- uB has been improved such that it is more intuitive from a UML–B modelling perspective.
- Some graphical visualisation of errors has been added.
- New requirements manager tool was developed (later versions referred to as a Context manager plug-in).

Feedback from second year

- Pilot study identified need for stronger guidelines for novice to refine models.

- UML vs. B investigation/justification by industrial user identified potential improvements in UML–B by adopting some UML constructs, e.g. sequence diagrams, Case diagrams.

Response

- Guidelines for refinement patterns developed by ÅA.
- Investigation and development into UML extension of UML–B

ATEC Perspective (Year 3)

- Easy installation
- Relatively easy to use but insufficiently mature for large development. This is largely due to minor bugs and some significant workaround, e.g. refinement of classes not possible.
- Several bugs found have been reported in SourceForge.
- Documentation would also from having examples that could be imported to assist novice learning.
- The most useful improvements in RODIN UML–B, over pre RODIN, were visibility in error marking and the use of contexts.
- Context views found useful for partitioning (c.f. metric 6, §5.2.6 below)
- Further enhancements have been identified, mainly through the development of the generic editor on the PAT case (c.f. metric 6, §5.2.6 below).

University of Southampton perspective (Year 3)

UML–B was used in the FMS case. However, at some stages, in the development only Event-B was used because of the rapid changes made to the UML–B plug-in. This made the development very hard because some older versions of the model could not be imported into the newer versions. Due to the immaturity of the tool, a lot of bugs were present, of which major ones were reported (see D26 [18]). This all contributed to delay the development of the UML–B FMS. Once a more stable version of UML–B was available, the Event-B model was translated into UML–B; only minor problems then had to be overcome.

The following feature requests and error report are of note:

- Feature Request #1777265 - comment on specific variable (etc) in UML–B

  The ability to comment on specific variables, axioms, invariants etc, especially if hidden from the actual UML–B diagram, would be extremely useful. These comments could be inserted in the properties tab (as in Event-B), and then added to the pretty print.

- Feature Request #1777262 - State machine transition naming

At the moment it is not possible to create a state machine that has two transitions with the same name. This, however, would be very useful so that the developer can represent a disjunction within the guard of an event.

- Feature Request #1777268 - refinement by state machine

  Some events might be specified as events of a class. It may be that at a later refinement stage, these events would be more suitable to be refined using a state machine. Currently this is not possible, and as a solution, the complete event has to be moved into the state machine. This is very time-consuming and error prone.

- Feature Request # 1779366 – UML–B integrated roundtrip engineering

  A highly desirable capability of UML–B for productive working, which should be addressed in the next round of UML–B development, is integrated round-trip engineering. Currently, when saving a UML–B model, the user must switch to the RODIN/Event-B perspective to analyse the model. Once he has identified consequent changes, he must return to UML–B to perform that change. For industrial-strength productive usability, the RODIN platform and Event-B language should act as far as possible as a "black box" analysis engine for the UML–B modelling activity, and its workings should only be visible through the UML–B interface. Specifically:

  - Errors and warnings from RODIN syntax/type analysis should be presented on the appropriate UML–B diagrams.
  - Similarly for proof obligations and animation interactions (state variable values, enabled operations).

- UG Forge Error Report #58

  UML–B currently lacks support for refinement. The model-to-be-refined has to be copied and pasted within the `.UML-B` file and then renamed. The *refines* events and *seen* contexts etc. have to be set manually. These missing features are already addressed and automatically undertaken within RODIN, and would be a very useful feature to include in UML–B. The problem associated with the lack of refinement support by UML–B is that the instance context also gets copied when copying the machine in the `.UML-B` file.

Abo perspective (Year 3)

Åbo have reported that their experience of using UML–B on case study two introduced no additional points to those already reported on case study one, see §5.1.4 above.

B2RODIN

ATEC applied this plug-in on its Dual Case sensor model during year three, as reported in D26 [18].

ATEC Perspective (Year 3)

- Easy installation.
- Good documentation with example.
- Easily converted documentation example.
- Unsuccessful conversion of dual case as some pre-formatting was required, e.g. dual case conversion of some classic B features (e.g. definitions, use of pre conditions) had to be reworked into Event-B format.
- The error detection was adequate and usefully supported further by Event-B static checker.
- Likely use will be to convert large-scale classic B to Event-B system on RODIN platform. Guidelines to support pre-formatting of Classic B file would be useful, e.g. what will not convert (e.g. definitions and recommended workaround in each case)?

5.2.5 Contribution to the Integration Objectives

a) Checking the scaleability of the system as its functionality is extended

Feedback on the new RODIN toolset has been received from Southampton, Åbo (FMS model) and ATEC (PAT development). No partner has indicated scaleability as a significant issue with the integrated toolset.

However it has already been stated that:

- Navigation is more difficult as windows platforms are extended.
- Maintenance of old models could become a problem as platform functionality is added since there will be an increased risk that upgrades may not be backward compatible with older versions of the tool.

Managing scaleability in the models

The UML–B methodology used in all the models addresses scaleability as it encourages O-O design concepts, such as classes, which naturally lend themselves to large-scale instantiations. RODIN event refinement also allows the specification of alternative development paths, e.g. the refinement of machine2 as either machine3a or machine3b. This will facilitate the production of behavioural model variants, thus supporting generic/product-line working.

FMS models

Managing scaleability has been addressed principally in the case study by the development of the generic model (c.f. §5 of D26 [18]), where the configuration of the model was intended to cater for FMS variants.

The development of the generic FMS model in year one identified issues concerned with large data set model instantiations and managing dynamic behavioural changes through features. This resulted in the development of a *requirement manager* tool (later referred to as the *context manager*) in year two and some exploration of feature development. The *context manger* was not applied to the year three generic model development, as it needed further modification to maintain compatibility with the UML–B developments.

The less-generic Åbo model also addresses scaleability by adopting UML–B classes to handle collections of sensors and by refining classes to develop behaviour.

In contrast one aim of the ATEC pilot study in year two was to explore basic development techniques for a fairly concrete dual sensor system with the view either to integrate its functionality in the scaleable generic model or to extend its scaleability through UML–B development. This was not undertaken in the final year due to ATEC's focus on the PAT case study.

PAT case study

The PAT structural model has been derived from domain analysis. Its O-O design lends itself to scaleable data centric configurability.

The model was used to generate automatically the generic editor, which illustrates the extensibility of the approach to requirements scaleability.

The model domain elements were also used to guide the structural development of the interpreter, which was implemented along more traditional methods. (The intention to model all the behaviour of the interpreter and then translate this to an implementation was considered too risky given the project team's experience).

The development of the generic system delayed large-scale data entry until the model was considered stable. This resulted in delays to the verification of the interpreter by traditional methods.

Grade: [4]

b) Checking the impact of legacy (sub) systems

The PAT system has had to consider legacy issues due to:

- The need to re-use existing application functionality, including low-level application code such as communication drivers, menus and error handling.
- The application of a legacy compiler to support legacy code.

The development is described in D26 [18] and is commented on below.

The implementation showed that reuse of legacy code could be integrated with new design requirements. However in practice this involved extensive rewriting of existing code, as the new requirement was better understood and a more generic implementation was needed. The UML model, which was used to generate the generic editor and guide the structural development of the interpreter, was useful in defining the generic design requirement. The interpreter needed to use some legacy code and the intention was to model fully the interpreter functionality, but this was not achieved. Instead some partial specification modelling was provided to examine the impact of legacy functionality on the new requirement. A simple functional model of a test element was specified and its dependency on legacy code was introduced. The behaviour of the legacy code was introduced only where it impacted the new requirement. Further refinements introduced other dependant behaviour. The model, though trivial, illustrates the impact of the legacy behaviour and how it can be incorporated into the requirement. The intent was to use the specification to isolate and replace legacy code as required for future maintenance.

The RODIN platform was used to develop the partial specification. The UML–B plug-in was useful in specifying partial behaviour as it provided a good visual representation of the legacy system. The use of contexts was also used to isolate the legacy structural items, which helps in maintaining legacy code. The animation of the specification using ProB helped to illustrate this behaviour to other users.

The target implementation needed an existing compiler, which it proved impractical to integrate into the Eclipse platform. However the use of an alternative compiler installed on Eclipse assisted development by providing a more effective interactive development environment. The implementation code was imported from the old compiler, modified and compiled in the newer environment, then imported into the older compiler for final translation. Having two platforms for the development only caused a minor hindrance.

In conclusion the impact of legacy systems can be applied in modelling partial behaviour to assist new development requirements analysis. However the formal translation of the full requirement involving the legacy system, from a model into an implementation, requires significant experience. The legacy development environment, i.e. compiler, is old and inappropriate for integration into the Eclipse platform. Thus the design requires porting to a standalone platform, which could create potential translation problems.

Grade: [3]

5.2.6 Case Study Specific Metrics

These metrics principally address ATEC's assessment of the usability of the technology.

1 *Evaluate the reduction of cost of development in future products. (U, C)*

Criteria Rationale: development efficiency makes the technology commercially attractive to the industrial user. This assessment is based on the hypothesis that:

UML–B will support the specification of generic products that can be efficiently configured for different specific applications products.

ATEC is interested in the costs of using the technology throughout a project lifecycle. This evaluation has involved an exploratory investigation into workflow and the associated learning costs.

Justification of grading:

Learning

- The cost of learning and applying the methods requires strong guidance for novice industrialists (c.f. §5.2.4, Assessment Report 2 [16]).
- The RODIN toolset was relatively easy to use and was applied by the industrialist to the PAT model. However, lack of experience in the methods limited a novice's progress. More behavioural examples in event UML–B would have benefited a novice's understanding of the notation.
- The academic experience of learning and using the tools on the FMS are reported below. However, in both cases the academics had previous experience, which was beneficial in adopting the toolset. This has been considered in ATEC's evaluation of the training and investment required.

Åbo perspective

Åbo already had a level of experience using several related tools and methodologies prior to RODIN. They are:

- experienced in the B Method, and the corresponding AtelierB and B toolkit modelling and verification tools;
- familiar with Event-Based modelling;
- proficient in UML and the underlying concepts of O-O design.

This prior knowledge of specific methods and associated tools had a large impact on our understanding and acceptance of the tools developed within RODIN, in particular the RODIN platform and the UML–B plug-in.

University of Southampton perspective

The RODIN platform and plug-ins are easily understandable with previous knowledge of B, UML and the ProB tool. The transfer from B to Event-B is straightforward, as it appears a simplification of B. The disadvantage of Event-B is that it does not contain all data structures that might be needed –should a new data structure be needed, it has to be modelled using functions and other available features. UML models can be constructed in the usual and familiar way and B annotations can be made to the diagrams. It takes some time to remember some of the modelling rules dictated by the UML–B approach, e.g.

the UML–B view must be used to make changes to Event-B, otherwise changes will be overwritten. The ProB animator is straightforward to use and should be self-explanatory even for someone without previous knowledge of ProB.

Potential Workflow improvement

The impact of modelling on workflow was assessed largely in year two; in year three the impact on legacy systems (c.f. §5.2.5 above) was considered. The new toolset's contribution to improvement in the modelling process was undertaken in year three and has been reported in the platform and tool evaluation sections (c.f. §5.2.4 above).

- Modelling assists requirement analysis, which reduces the risk of late changes due to the wrong requirements being met. Although the approach can be adapted, it can prove inflexible at times. However, modelling takes more time at this stage than traditional methods of requirement analysis (c.f. §5.2.4, Assessment Report 2 [16]).
- Modelling enables the intended requirements to be translated into design and implementation accurately with minimum errors. This can result in reduced verification time and provide a reference to assist the efficiency of other workflow processes e.g. testing, documentation and certification (c.f. §5.2.4, Assessment Report 2 [16]).
- Modelling doesn't increase significantly the burden on other workflow processes such as maintenance and testing (c.f. §5.2.4, Assessment Report 2 [16]).
- Limited use of the technology can applied to the development of legacy products (c.f. §5.2.5 above, PAT partial specification).
- The UML–B toolset could provide more efficient modelling once it is robust enough for industrial use.
- The use of UML–B to express modelling behaviour was shown to have benefits over the more textual approach used in the Dual case sensor of year two. Significantly the partial PAT model could be expressed better visually and more efficiently using state machines. Furthermore the use of contexts enabled easier structuring of design so that a mapping between the functionality and its static structure was visible.

Reusability and configurability of models

- The generic model was not developed sufficiently to verify different engine variants configurations. However the work demonstrates this is achievable.
- The generic PAT case study editor successfully provided a configurable package for real system test specification variants.
- Not all FMS models included all the behavioural functionality identified in the traceable requirements deliverable D4 [4]. However we have illustrated how this can be achieved.

- The final FMS models have demonstrated that the domain aims of genericity and reusability have been met.

Grade: [4]

2 *Evaluate the impact on maintaining current quality levels. (U, S)*

Overview

Criteria Rationale: This assessment is based on the hypothesis that:

The use of the UML–B method will result in high quality products.

Thus, UML–B and the RODIN methodology should prevent errors from being introduced and detect errors early to promote their removal.

ATEC has considered how effective the methods are at detecting and avoiding errors from year two. This was reviewed in year three.

Grade Justification:

- ATEC explored the flexibility of formal models when developing prototype requirements in year two. They found the formal approach restricted rapid development but could be adopted.
- The strong process used to translate requirements into design helps avoid errors. However it requires guidance during the refinement process.
- The RODIN methods, tools and processes do not address non-functional requirements.

Year 3

- The UML–B method enabled formality to be expressed at a higher level; consequently the novice could avoid errors. In the PAT model the *dot* notation was used in the diagrams rather than the lower level syntax. The automatic generation of Event-B from the classes also reduced errors. The partial model translated into Event-B and proved by automatic proofs, further illustrated the effectiveness of error avoidance through design translation.
- Error detection in the application model has been assisted by the functional visualisation. In the PAT model it was useful for the legacy read behaviour to be expressed visually as two events (read and abort) on a state diagram as this highlights the source of this behaviour in this case of a legacy read (c.f. §3.2 of D26 [18]).
- The new UML–B plug-in provides error marking in its diagrams for syntax errors. This was useful in identifying incomplete model requirement (currently only errors which violate the UML–B abstract syntax are checked, further work is required to implement other syntax errors).

Grade: [3]

3 *Evaluate whether the improvement of ease of certification has been achieved (U, S)*

Overview

Criteria Rationale: The method may facilitate a means of producing documentary evidence for certification, such as that required in EUROCAE ED-12. This would lead to greater acceptability of the methods by the industrial user.

During year two ATEC considered how the methods contribute to certification standards and how the products are suitable for independent verification (c.f. §5.2.4, Assessment Report 2 [16]).

Table 7 summarises the year two investigations. The ✓/✗ column indicates whether the RODIN method and platform helps (✓) or hinders (✗) certification.

No	Description	Consideration	✓/✗
1	*Traceability between system requirements and software requirements should be provided to enable verification of the complete implementation of the system requirements and give visibility to the derived requirements* Note system requirements allocated to software include functional, performance and safety related requirements. These requirements are considered high level. Some derived requirements are regarded as high level though not directly traceable to the system requirement. (c.f. §5.5 [88])	"Information is traceable if the origin of its components can be determined." this assists verification processes of review and analysis for requirement coverage. B refinement models do not label explicitly the introduction of requirements and their development. This may hinder the analysis and assurance of a model as it would: a compromise tracing to and from the system requirements. b compromise visibility of derived requirements for safety analysis. A solution may be to explore ways to tag model development against requirements in order to generate traceability matrixes.	✗
2	Compliance of safety requirement.	By adopting invariants that reflect system safety constraints, confidence is increased that the model is complete and its implementation will conform. This should assist certification.	✓
3	The software architecture and low level requirements are developed from the high level requirements.	The refinement process and checking mechanism, which controls permissible changes, aids development of compatible low-level	✓

No	Description	Consideration	✓/✗
	The goal is to avoid errors during the development process. (c.f. §4.4, 5.2 [88])	requirements. Identification of data invariants with refinement proof by the tools may also support the verification of this requirement.	
4	*Traceability between the low-level requirements and high level requirements should be provided to give visibility to the derived requirements and the architectural design decisions made during the software design process and allow verification of the complete implementation of the high level requirements.* Note low-level requirement refer here to design requirements from which source code can be produced (c.f. §5.5 [88]).	The guideline distinguishes design requirements from high-level requirements in its analysis to allow for separate review and analysis activities. This distinction is less clear using B model refinement, where it seems that high-level requirements modelling and design are closely bound. The tracing of high-level requirements to the design is compromised, as requirements are not tagged in the model.	✗
5	Test cases are required to demonstrate compatibility of the system with requirements and provide assurance of requirement coverage (c.f. §6.2 [88]).	Conventionally requirements compatibility and test coverage is only intended to be applicable to testing of the target code and is not concerned with modelling. However demonstration of test case completeness will be easier if the B refinement model can demonstrate requirements coverage and the resulting implementation can be shown not to introduce incomplete, or unintended requirements. Mapping of requirements with animation test cases using ProB should help to give this assurance.	✓
6	Dead code should be removed (c.f. §6.4 [88]).	Although the guideline requires structural coverage analysis to be applied to the source code, the identification of dead model code would provide assurance in the source code and assist certification. The tracing of model detail to requirements would assist this.	✓
7	Deactivated code is allowed but needs to be verified and shown not to be executed inadvertently (c.f. §6.4 [88]).	Though not identified in the pilot study model the generic FMS model could generate deactivated code. This would be where a configuration did not require instantiation of all the	✗

No	Description	Consideration	✓/✗
		generic model behaviour.	
8	Qualification for tools is required when the output generated by the tool is not verified. The objective of qualification is to provide confidence that the tool is at least equivalent to the processes it is eliminating or reducing (c.f. §12.2.4 [88]).	UML–B will not need to be verified as its output will be verified during model verification. The *requirement manager* may need verification depending on how the requirements are verified in the model verification. E.g. what ensures that the instantiation is as intended? The verification tools ProB and B4free will need qualification if their verification results are to be relied on. The onus appears to be on the tool user to gain qualification agreement. A formal methods novice may not find this easy.	✗
9	User modifiability- What ensures that a designated non-modifiable component of the software is protected from unintended modification? (c.f. §5.2.3 [88]).	The FMS generic model may need to show how its behaviour remains protected when a user can instantiate the model. This may be achievable by only allowing instantiation through a specific interface.	✗

Table 7: Year Two Assessment Overview

Year three did not investigate certification issues further. The grading is derived from the following summary justification.

Justification:

- Certification can still be applied
- Slightly more effort may be required

Grade: [3]

4 Evaluate the reduction of cost of late requirement changes (U, C, S)

Overview

Criteria Rationale: Typically the cost (impact) of incorporating late requirements change is expensive. Thus, any cost reduction in this area, which results from using the RODIN method and tools, will provide real benefits.

ATEC has considered how the methods contribute to early validation of requirements (reducing the likelihood of late change), understanding design, as well as managing the impact of late changes. The industry often needs to support long maturity cycles of products (over 10 years) so long term maintenance has also been considered.

Justification:

- Early requirement validation is possible (c.f. §5.2.4, Assessment Report 2 [16]).
- Change can be accommodated relatively easily if the model is designed well (c.f. §5.2.4, Assessment Report 2 [16]).
- The three-year FMS models successfully introduced some detailed functionality into each model, i.e. a confirmation mechanism without the need to change the static design.
- The PAT generic model easily configured test variants. However the development of the structural model did impose delays to development and the interpreter code needed rewriting. This implies that large structural changes to the model from new requirements could have a potentially significant impact on maintenance.
- The PAT legacy modelling is expected to assist future development.
- Long-term maintenance is a concern where backward compatibility of the RODIN platform with its plug-ins needs to be assured .

Grade: [3]

5 *Evaluate whether improvement of portability has been achieved (U, X)*

Overview

Criteria Rationale: The evaluation of portability has been assessed in terms of MDA (model driven architecture) concepts. This assessment is based on the hypothesis that:

> *The use of the UML–B method will result in platform independent models and facilitate the transformation to different platform specific models thus improving portability.*

ATEC has considered how the methods contribute to developing a platform independent model and platform specific model.

Grade Justification:

- Pilot study successfully developed a platform specific model for dual sensors (c.f. §5.2.4, Assessment Report 2 [16]).

- The final FMS generic model does not yet contain all the functionality of the case. However the functionality introduced is generic and thus some form of platform independence has been achieved. The transference to a platform specific model, i.e. the generation of a variant, has not yet been achieved.
- The PAT model generic editor provides a platform independent model.
- The Åbo FMS model provides a template pattern that can be reused in the FMS environment, i.e. it is portable to some degree. However when the pattern was applied to the PAT case, it was shown to be less appropriate as the pattern didn't focus on the main functionality of the case study.

Grade: [3]

6 *Evaluate the improvement to the UML–B method (U, C)*

Overview

This evaluation is based on the extent to which the UML–B method and its development have assisted model development through reusability and validation.

FMS models

UML–B influenced the development of the generic FMS models by dictating an object-oriented modelling approach from the outset. Initially the context diagram, gave structure to the model, which enabled its behaviour to be defined.

Object-orientation was natural for FMS; its structure consists of components, relations and instances that have different behaviours.

The UML–B modelling approach encouraged development of re-usable models. The instance context can be replaced easily. The behavioural model, which consists of several refinements, can be reused by adding further refinements, thus extending the functionality of the abstract model, or by feature replacement.

The new RODIN UML–B plug-in extended the FMS modelling capability in year three, as its new features, e.g. contexts and hierarchical state machines, allowed behaviour to be expressed more clearly. Model translation on the RODIN platform has enabled the powerful verification capability to be used for model proving.

In contrast to the generic FMS model, the Åbo model development began as a classic B design and was modified to UML–B, which made the visualisation of the design more understandable. The immaturity of the early RODIN UML–B plug-in hindered its model development in year three.

The integration of specific behaviour from the Åbo model into the generic model took place later in year three using the new UML–B tool (though some workarounds were required to cater for current tool bugs).

Unfortunately ATEC had insufficient time to apply the new UML–B plug-in to FMS.

However in assessing the final FMS development, undertaken by University of Southampton and Åbo, ATEC believes that UML–B is more readily understood than classical B.

E.g.:

- The UML–B package diagram shows graphically the contexts to machines relationships, which is much clearer than textual naming conventions.
- The context diagram visualisation of static structure dependencies is clearer.
- Contexts allow design layering. This eases maintenance. In year three the application of UML–B illustrated how the context data could be layered, for example model primitives were separated from the rest of the static data. Similarly the test environment, such as ProB, can be isolated from the design.
- State machine representations provide a clearer representation of state.

However the FMS models developed did not manage to address all the functionality of the requirement specification, which was disappointing. The logistics of tool feature availability and the general difficulty of domain modelling contributed to this shortfall in functionality rather than the modelling language.

UML–B's model verification and validation capability was improved by its translation into Event-B. This allowed use of the powerful RODIN platform provers. However, the UML–B (diagrams) notation's error marking of incorrect syntax is limited.

PAT Case Study

The new UML–B plug-in was successfully applied to a partial specification of the PAT case study. ATEC viewed this as a success, since it enhanced the understanding of the legacy behaviour, which is beneficial for future maintenance and development of the interpreter. The RODIN platform provers and ProB provided a sound verification and validation of the model.

Future enhancement

Although the PAT generic editor was created with UML and used EMF, its verification would have been strengthened by the adoption of UML–B, which would have allowed formal verification to be investigated. However this was impossible as UML–B lacked a number of key features. This use of UML–B, and its application using EMF technology, forms the basis of future development research of UML–B on the RODIN platform. The case studies also highlighted features, such as multiple inheritance and classes as a collection available in UML, which are currently deficient in UML–B (c.f. UML–B, §5.2.4 above).

Grade Justification:

- Year two concluded that benefits were to be gained from choosing UML–B over UML and B approaches (c.f. §5.2.4, Assessment Report 2 [16]).
- Year three indicated potential benefits from using new UML–B on FM and PAT case studies, but the maturity of the tool hindered progress

Grade: [3]

5.2.7 Conclusion

The FMS case study has created several models and in the year three provided some integration of this work. Whilst all of the requirements specification [19] was not addressed they illustrate that the domain aims can be satisfied using this approach.

Industrial acceptance of the technology is seen in the value of model development in the workflow process. The early expectation of the case study, to develop the FMS model translation into an implemented system (concept to grave), was not realised due to lack of experience and pursuit of other evaluation aims. However ATEC feel, while this experience is being gained, it has found the current practical use of modelling with RODIN is through requirement exploration guiding development rather than its direct translation into implementation.

In conclusion the case study has successfully applied RODIN methods and tools to both cases and generated several domain models (e.g. generic FMS) and guidelines for working (Åbo template).

The new toolset has been relatively easy to use once familiar with the methodology. However, in some cases the tools still lack the maturity in features or development to be used commercially or for some types of application. The bugs and feature requests have been stored in SourceForge.

5.3 CS3 – Formal Techniques within an MDA Context

5.3.1 Introduction

This case study is concerned with the formalisation of various subsets of the MITA platform [42] (developed in Nokia within the NoTA – Network on Terminal Architecture project), including the formalisation of the infrastructure and development of techniques to allow MDA to be used more formally.

The objectives of this case study (from the Description of Work [1]) are to:

a Investigate how formal techniques fit into the Model Driven Architecture (OMG MDA) framework as "MDA Mappings".
b Investigate which techniques are applicable at which stages of platform independence and platform specific models.
c Investigate how to integrate and compare the verification and validation results from the various levels of abstraction.
d Investigate methodological issues relating to formal model development with an emphasis of refinement and retrenchment.

5.3.2 Current Status

The primary driver within Nokia for this case study has been the NoTA project, which ended in December 2006. Consequently, although use has been made of the early versions of the ProB and UML–B tools, this case study has not been able to provide feedback on the versions of these tools, which are now integrated with the RODIN platform.

The main mandate for case study three was to investigate the use of formal methods, in particular B/Event-B/B-Method, in the context of existing Nokia approaches and practises focusing on MDA.

Several results on the use of MDA and MDA-based techniques (a.k.a.: MBE, MDE, CtM etc.) have been promising (see D26 [18] for more detail), including a method developed for introducing formal transformations of platform independent models (PIM) to platform specific models (PSM) [41]. At the same time a number of factors have caused difficulties, e.g. the need to change from an O-O type of modelling to Event-B development methods has not been properly supported by tools, weakness of the UML semantics, and unavailability of suitable and generic transformation mechanisms.

During year three a number of important contributions have been finalised; including: a method for incorporating fault tolerance into NoTA systems, an approach to refinement of the context, and a prototype of a plug-in for circuit development.

During year two the NoTA High Interconnect layer was formally developed using UML

and B (see D18 [14], §4.2.6). To allow this, high level models of the context in which this layer functions, including NoTA Domain Model, the NoTA Architecture and the Interface partitioning, were formally defined.

Nokia consider the three years work on the RODIN case study three to be a partial success and that the objectives set for this case study have been achieved.

The results from year three demonstrate that while the tools and general methods are practical, certain aspects of very strict methods are difficult to apply cost-effectively. The extent to which the methods and tools can be applied very much depends upon the nature of the project. In certain, well defined, controlled cases a strict refinement based approach provides benefits.

Year	Achievement	Objective	Papers
2	NoTA High Interconnect layer modelled	a)	[14]
2-3	Considerable experience in using ProB and UML–B (U2B)	d)	[40,38]
2-3	A method for applying fault tolerance	d)	[39]
2-3	Initial work on model-based testing	c), d)	[38]
2-3	A method for using MDA	a), b)	[41]
2-3	Context-aware refinement	d)	[43,44]
2-3	A prototype of a plug-in for circuit development (B2Bsw)		[37]

Table 8: Case Study 3 - Annual achievements against objectives (c.f. §5.3.1 above)

5.3.3 Progress since Year 2 Assessment

Since year two a number of important contributions, which were initially reported in D18[14], have been finalized – see Table 8 above and D26[18].

Only partial success has been achieved in obtaining insight on various patterns (and thus transformation techniques) for fault-tolerance that is making fault-tolerance a “platform” in MDA parlance. While some patterns do exist these have tended to be tailored specifically for the B/Event-B language, the particular process and problem in question.

Nokia found some success in the hardware domain and work in this area is progressing outside of RODIN due to the lack of support within the project for this work. Hardware development places certain constraints on the use of Event-B and we were unable to investigate this further within the RODIN context.

5.3.4 Contribution to the Development of Platform and Plug-ins

RODIN Platform

The RODIN platform has been used successfully for modelling a subset of the NoTA High Interconnect layer, and for modelling circuits and generating Bluespec programs for Event-B models (in combination with a plug-in for circuit development).

ProB

ProB provides much necessary support for the default theorem proving and thus verification techniques already present in the RODIN tool. Use of ProB was extremely useful in validating verified models – verification removes certain error conditions and ensures that the model we are validating is "correct". Using our development style, Verification became a secondary concern, with the focus more on establishing that the specification met the customer's demands rather than on establishing the adherence to certain properties. This fits in well with the style of development commonly seen in industry where constructing a model to investigate the properties of the system is not always feasible – ProB and the validation style of development in this sense provides a way of first constructing and demonstrating systems then discovering properties later.

In use ProB is stable and reasonably fast. Scaleability is always an issue but in the sizes of models we have presented to the tool we have not seen any problems.

UML–B

Overall UML–B provides a compromise between the mathematical abstractness of B/Event-B to the apparent "concreteness" of UML (at least to the engineer who forgets the underlying concepts of languages such as UML). However this makes the language awkward to use without better methodological support – one has to think more in B/Event-B terms rather than UML. Application of "traditional" (sic.) domain modelling techniques or even E-R techniques produces simple enough static or structural models but actions/operations/invariants are required to be written in a form more applicable to B/Event-B rather than the object-oriented ideals of UML.

5.3.5 Contribution to the Integration Objectives

a) Checking the scaleability of the system as its functionality is extended

During the team's work on case study three no negative effects have been identified whilst extending system functionality (in particular, when the B2Bsw plug-in was added to the platform and when the team experimented with the ProB and UML–B plug-ins).

d) Checking the sensitivity of the methodology to changing requirements with respect to the models

The team performed some initial investigations into how requirements change and the volatility of specifications can be addressed. The B Method itself takes the point of view that these are outside the scope of this particular method, which concentrates on refinement and decomposition of the models. As far as the RODIN toolset is concerned, development of a tool supporting requirement change and traceability was not part of the tool definition. Some investigation was conducted with retrenchment but, again, without tool and method support this has proven difficult. Requirements change and how to handle this formally still remain a challenge and some investigation has been made in collaboration between Nokia and the University of Southampton but the results are still forthcoming.

Some results have been obtained regarding requirements change and the fact that using formal methods reduces the amount of change in a specification due to the level of previous work, which aims to ensure that the system being developed is the system actually wanted by the customer.

5.3.6 Case Study Specific Metrics

The Description of Work [1] defines the following metrics for this case study:

1 *How well does this formal approach fit with existing processes? (U, X)*

Moreover, do existing processes admit these more formal techniques and tools to be integrated with them? Generally this is the case however there are issues regarding the amount of pragmatism that needs to be made in reality.

A fully formal approach is not generally suitable in many cases. However we can show that the additional and sophisticated analyses made during analysis and design time significantly reduce the amount of debugging and testing time later. One of the main problems of testing is that it is not an exhaustive technique and susceptible to not actually testing the relevant features of the system. Model checking, animation and to a lesser degree, proof (though this is mainly due to lack of experience in mapping proof obligations to tests) provide a wealth of information about "weak" or potentially weak points in the system.

Grade: [4]

2 *How well do these techniques integrate with an object-oriented approach? (U)*

B and Event-B are not based on the ideas of object orientation and thus admit different ideas to O-O. However the use of the ubiquitous UML leads to the situations where we need to map O-O ideas to B/Event-B and the problem here is the amount of B/Event-B generated to support the more advanced structures which

has a detrimental effect upon the complexity of the proof and animation of the specification.

U2B in its previous incarnations has been advantageous in the generation of B. We however have not tried the latest versions targeting Event-B due to the necessity of using ProB, which did not support Event-B during the case study three timeframe.

Grade: [2]

3 *How many of these technologies have been transferred to the Nokia Business Units? (U, X)*

NoTA is currently being readied for a transition to a production environment.

Grade: [3]

4 *How well can this approach check components against (very loose) specifications? (C, S)*

Most specifications are inherently loose at the beginning. Formal methods force the developers to concentrate more on the ideas driving the system's development and thus improves the specification overall towards a more well formed and thus less loose specification.

Grade: [3]

5 *What is required to understand over constrained models, their causes and effects?*

This was not explored this in detail, as over-constrainment seems to occur with the composition of features and other factors related to product line development.

Grade: [2]

6 *Can this approach deal with requirements volatility? (C, X)*

Again related to the previous point this was not explored further during year three.

The preliminary results regarding requirements change are that the use of formal methods reduces the amount of change of a specification due to the amount of previous work made on making sure that the system being developed is the system actually wanted by the customer.

Grade: [2]

5.3.7 Conclusion

Nokia consider the three years work on RODIN case study three to be a partial success. They have obtained:

- useful practical results in evaluating feasibility of applying formal methods in the context of MDA;
- considerable experience with the use of B in a number of challenging applications;
- extended skills of using the RODIN platform and the ProB and UML–B plug-ins as the major support for formal modelling;
- good experience in developing RODIN Eclipse plug-ins for the RODIN platform.

However specific methodological support for Event-B and fault-tolerance is still lacking. Other aspects such as mobility, distribution and concurrency still remain to be addressed.

While B/Event-B have their uses and indeed the RODIN toolset itself has made the use of Event-B possible the current incarnations of the toolset and the language is not yet mature enough nor is the support available for the kinds of work the case study three team wishes to undertake at this point in time.

The team's current project is to investigate the use of highly distributed and mobile systems with computationally heterogeneous environments as the basis for a semantic web. The project will require the use of fault-tolerance techniques and potentially the use of languages such as Event-B for specific parts of the system – in this respect parts of RODIN will be used actively within Nokia.

For the coming years the case study three Nokia team will remain the active users of such techniques. Nokia do however have an active education programme specifically for modelling techniques based around the UML language, which specifically addresses the aspects of formality and abstraction of models into which the use of tools such as RODIN can be easily integrated. As preliminary training in B needed to be undertaken before the RODIN platform was available, the AtelierB toolset was used. This was successful, and enabled the engineers to gain early familiarity with formalisation techniques and the RODIN methodology. This highlighted both the amount of preliminary knowledge required by the engineers in some aspects of the method and the misunderstandings about such techniques. No issues were reported in transferring from AtelierB to the RODIN platform.

5.4 CS4 – CDIS Air Traffic Display Information System

5.4.1 Introduction

Originally CDIS was a display information system supporting a new terminal control room at the London Area and Terminal Control Centre in the early 1990s. It was a distributed real-time information system for air traffic controllers. CDIS was developed using advanced software engineering techniques including the extensive use of formal methods for specification and design. Praxis (now Praxis High Integrity Systems) developed CDIS for the Civil Aviation Authority (now National Air Traffic Services).

The CDIS development produced a number of formal specifications, including a 1000+ page main specification written in the VVSL variant of VDM-SL (a model-based specification language). The key concurrent aspects of the system are captured in a separate specification written in CSP (A process algebra).

There are a number of publications on the development of the original CDIS software. The primary reference being “Using Formal Methods to Develop an ATC Information System” [60].

The objectives of this case study (from the Description of Work [1]) are to:

a provide some of the initial drivers for the theoretical work in RODIN, by offering specification, design and verification material from an existing industrial-scale development and identifying the challenging issues that arose during that development;

b exercise the intermediate tools deliverables of the RODIN project by subjecting them to the "stress testing" that only real industrial problems can provide;

c provide a realistic assessment of the final results of RODIN by comparing what was possible in the 1990s with what can be achieved on the same development with the RODIN notations and tools.

This case study was intended to provide RODIN with the opportunity to compare the capabilities of modern formal methods tools against what was commercially feasible ten years ago. The size of the specification was the first major test of the RODIN tool platform, as it had to highlight any scaleability issues, which the developed platform might have. Once the specification was developed on the RODIN tool, the secondary aim was to investigate the degree of analysis that is possible for the specification.

Another key test for the tool platform was the degree to which the tool supports the refinement of the specification to a detailed design. The initial CDIS design is concurrent and heterogeneous, with a number of different classes of workstation and system support devices. The concurrency aspects were not described in the original VVL-based specification of CDIS, and during the original development the concurrency aspects were introduced during a manual refinement step. The main drawback of this approach was that

there were no formal links between the original specification and the subsequent refinements. Another major goal in re-modelling CDIS was to investigate whether the tool can facilitate the introduction of distribution and the resulted concurrency, as this will represent a major advance in the system development process.

5.4.2 Current Status

During the first year of the project the original CDIS specification and design documents were retrieved and we validated our ability to regenerate some part of them from source. In the second stage a subset of the CDIS original specification was selected and extracted by the CDIS Technical Authority, Anthony Hall.

As this case study was intended to use the new Event-B notation and associated methodology and utilize the new tool platform extensively, during year two an Event-B development of this CDIS subset was undertaken. Most of the functionality of the subset was incorporated into the Event-B development. Some elements of the distributed design were incorporated in the development and the rest of it was to be completed in Year three. All Event-B models in the year two CDIS development were checked and verified using the B4free tool.

During Year three the development was ported to the RODIN environment to help with evaluation of the environment. Although some attempts were made during year two to mimic the newly introduced notation of the RODIN Event-B, it was not possible to take the full advantage of the new notation until the new tool platform became available. The reason being that B4free tool only supported the standard B-method. Thus we had to amend the B models, which were developed during the second year to adjust them with the new Event-B constructs.

The process of porting from standard B to the new RODIN platform proved to be very challenging and it has provided the main platform developer with a wide variety of feedback and a wish list to be considered for future extensions.

After releasing a number of sub-versions, the tool has gradually reached a reasonable stage of stability, which can now produce and discharge a much wider set of proof obligations in comparison with the B4free tool. The new facilities resulted in several amendments and enrichments to the produced B models during year three. As another result of tool enhancement we were able to develop our B models further and add two further refinement levels, which now bring the total refinement levels to six in addition to the initial specification model. Most of these refinements are horizontal refinements, where we have inter-cooperated new feature to the previous levels.

Another path taken during the year three was to develop a realistic distributed version for CDIS. As has been noted earlier, the initial VVL-based CDIS models and the corresponding B models which have been developed during year two were all an idealised centralised versions. Some attempts toward producing a distributed model were initiated during year two. During the last few months we have developed a simple witness case study to work out the exact approach to extend the idealised version to a

distributed version. After completing this stage and developing some useful strategies for handling the proof obligation, which have arisen from this extension, we proceeded to the next step. In this stage developed a full-flag distributed version.

The development process of the distributed version involved both horizontal and vertical refinement. As usual the horizontal refinement reflects the process of introducing new features into the system. On the other hand the vertical refinement represents the refinement of data structures and the effects of this refinement on the events to make the whole model more implementable.

A summary of overall activities is presented in the following table:

Year	Achievement	Objective	Papers
1	Retrieval of original CDIS specification and design Extracting the system requirements.	a)	[3]
1	Selecting suitable subset of requirements Strengthening and finalising requirements document.	a)	[4]
2	Producing B specification and refinements for CDIS Porting initial VVL-based Spec to Event-B.	a), b)	[8]
2-3	Producing initial distributed model To produce more realistic model and assess reusability and extendibility.	a), b)	[14]
3	Porting the second year B models to the new RODIN platform and further extend them. To examine the new faculties in the new platform and provide feedback to the tool developer.	b), c)	[18]
3	Producing specification and refinement for a realistic distributed model. To assess reusability and sensitivity to changes in the requirements and model—to assess how the tool is coping with complex models and associated proofs.	b, c)	[18]

Table 9: Case Study 4 - Annual achievements against objectives (c.f. §5.4.1 above)

5.4.3 Progress since Year 2 Assessment

As summarised in the above table, substantial achievements have been made since year two. The major progress has been to:

- Port the B models which were developed during year two to the new RODIN tool
  - This includes replacing standard B constructs using new Event-B constructs and using the new facilities, e.g.: refining one event by more than one event.

- Extend the refinement levels by introducing two new refinement levels.
- Discharge all proof obligations, including new types of proofs, which are specific to the new RODIN tools, e.g.: Well Defined (WD) proofs.
- Apply strict separation of context and model based on the new methodology, which is both recommended by RODIN and enforced by the RODIN tool.
- Produce a realistic distributed specification with four levels of refinement.
    - This stage includes all features, which were described during stage one for non-distributed models.
- Discharge all proof obligations for distributed models.

As is clear form the above list, during year three we have been working very closely with the RODIN main platform team. This has helped us to examine every aspect of the tool and its system modelling facilities. In the next section we discuss how our involvement with the tool has contributed to its improvements.

5.4.4 Contribution to the Development of Platform and Plug-ins

CDIS was an industrial-strength case study with a high degree of complexity. Although as far as RODIN is concerned only a subset of the initial system has been chosen to be redeveloped, but still it intended to utilize the main platform and some plug-ins very heavily. The initial plan was that in addition of the main platform it has to assess ProB and U2B plug-ins. Due to some technical complexity in CDIS models unfortunately up to now we have not been able to use these plug-ins as it was expected. Also we intend to keep this possibility open for near future.

In the next section were review the use of main RODIN platform in CDIS case study and how it has contributed toward the main platform enhancement.

RODIN Platform

As soon as the early pre-release versions of the main RODIN platform reached an acceptable level of stability we started to port the B models produced during year two using the B4free tool.

Initially, we attempted to use the B2RODIN plug-in at this stage to convert and port our models from the B4free based standard B to the RODIN Event-B style. This attempt was unsuccessful and the plug-in did not succeed in producing any Event-B output. Therefore we decided to undertake the above task manually.

As a result a number of issues needed to be tackled; a prime example being the replacement of some standard B constructs, which no longer existed in Event-B, with corresponding constructs and adjusted the modelling style to that recommended by RODIN methodology.

During this stage we provided the developer with a sizable set of feedback and a wish list for future modifications, some of which have yet to be incorporated in the tool. This feedback ranged from interface issues to performance related aspects. Examples are:

- Interface issues:
  - The interface to some resources of the project like "Log files", "Comment view" and "Project resource files" was obscure
  - The "Problem view" on the window platform did not show mathematics style characters correctly.
  - Other interface related issues, where the related view or button/control did not show up as expected
- Some inconsistencies in the early version of the underlying model repositories.
- Performance related issues such as slow speed of workspace rebuilding in the earlier versions.
- Lack of help and documentation when early versions released.

Having completed the CDIS specification and early refinement stages, we started to use the RODIN tool prover. Again during this stage we provided a sizable set of feedback to the tool developer. The issues identified were related either to the prover interface or to weaknesses of the internal prover algorithms.

There have been very noticeable improvements in all aspects of the tool. However we still have a wish list of features to be integrated in future tool versions. Some of the additional features, which we recommend be provided are:

- A higher level of support for model documentation and multiple commentary lines
- An improved editing environment which supports:
  - A free style line format, which support multi-lines constructs, to facilitate the breaking of log lines across multiple lines.
  - Redo and undo facilities
  - Pop-up help in the form of callouts when holding the mouse over predefined variables, constants, etc
  - Auto-complete facilities as provided by other contemporary editors.

Grade [4]

ProB Plug-In

The current version of this plug-in only supports animation of a simplified version of CDIS. This is because we have used some constructs and aspects of Event-B and the new methodology, which are not yet supported by ProB.

Grade [N/A]

UML–B Plug-In

The UML–B plug-in could not be used as planned since it is currently unable to support the CDIS complexity. We expect that this support should be available very shortly.

Grade [N/A]

B2RODIN Plug-In

This plug-in was needed as we intended to port our second year models to the RODIN platform. Our attempts at that time with early versions of this plug-in were unsuccessful.

Grade [N/A]

5.4.5 Contribution to the Integration Objectives

a) Checking the scaleability of the system as its functionality is extended

A full experimentation with plug-ins was not possible in the case of CDIS. Consequently no solid conclusion has yet been reached. Some preliminary experiments in this area suggest that the tool scales well as plug-in functionality is extended.

Grade [3]

c) Checking the scaleability of the system with respect to the size and complexity of the models

As far as the main RODIN platform is concerned, the current version copes very well with sizable complex CDIS models. The automatic proof-discharging rate has increased with the introduction of recent versions. The linking facilities between semi-automatic proofs the source model has been enhanced. Another aspect is the prover interface, which is now much easier to understand. Although the performance of the tool, especially during the workspace re-build process, has improved noticeably, there still appears to be room for further performance improvement. As mentioned in section 5.4.4 above, improvements in the editor interface are highly desirable. The prover itself also needs some optimisation.

Grade [5]

d) Checking the sensitivity of the methodology to changing requirements with respect to the models

This metric depends on the context, in which it is applied. One common aspect when changing requirements is when we develop the system specification and undertake some stepwise refinement. During the refinement process sometimes the developer can identify an inconsistency, ambiguity or incompleteness in the initial requirements. In this situation they have to return to the specification and make some changes. An ideal tool

would propagate the changes, which have been made in higher level, or at least would provide some assistance in this regard. As far as the RODIN tool is concerned, this was not part of the tool definition; it may be considered for future extensions and enhancements.

The second aspect of this metric is the situation similar to that experienced with the CDIS case study where initially we developed a centralised version and subsequently extended it to a distributed version. In this case the strategies, which we adopted, based on the recommended methodology were largely reusable. Additionally, proof obligations of the non-distributed version were very helpful in dealing with the distributed version's proof obligations.

Grade [4]

5.4.6 Case Study Specific Metrics

1. *Comment on the relative ease of comprehension of the Event-B specification as compared with the original VVSL specification? (U)*

   The layered development has considerably improved the comprehensibility of the specification. This is because we were able to capture the essential functionality of the system in the abstract specification. The abstract model is written using slightly less than four pages of Event-B; and we claim that this abstract model allows the reader quickly to grasp the essence of the system. Six subsequent refinements were used to introduce additional features of the system. The main features of these refinements are that they add details to the information structures and introduce further constraints on the events' guard. The layered nature of their introduction means they can be absorbed in a stepwise fashion thus easing comprehensibility.

   Grade: [5]

2. *How does the time taken to construct the Event-B development of the subset compare with the known design time for the original CDIS development? (C)*

   The B development, which was based on the B4free tool in year two, represents about five months of effort. This includes the time to read and understand the original requirements and specification. It also includes the time spent learning to use B4free efficiently, as well as the time spent checking the models and performing the proofs using B4free. Other time included is the time spent clarifying some issues with the system with Anthony Hall (Praxis) halfway through the development and revising the development as a result. It is therefore difficult to compare directly with the time taken for the original VVSL specification. Our subjective assessment is that the time taken is comparable, with the advantage that in the case of the Event-B the development has been machine checked and proved.

   During year three the following activities have been completed:

- Porting the initial B models for the idealized central model, which they have been developed during year two, to the RODIN platform.
- Amending the modelling style, to bring it inline with the RODIN recommended style.
- Adding two further refinement levels, this brings the total levels of refinements to six.
- Discharging all proof obligations. Considering the fact that the new RODIN tools produce more proof obligations and generally behave differently, it took us some time to deal with the new style of proof obligations.
- Developing a distributed version of the CDIS system, including one specification and four levels of refinements.
- Discharging all proof obligation of the distributed system.

The above activities have taken almost six month to complete. Considering that:

- The RODIN tool has evolved during the modelling process and sometimes we had to wait for required facilities or bugs to be fixed.
- The RODIN tool produces more proof obligations and in many situations behaves differently from previous tools such as B4free.
- Until recently there was no adequate help on the different aspects of the RODIN tool and its interactive prover.
- We have developed two more refinement levels in the centralized version.
- We have developed the distributed models including one specification and four refinement levels.

We believe this to be a good achievement. We are confident that similar systems can be modelled much more quickly in future.

Grade: [4]

3. *Comment on the expressive power of Event-B relative to the original specification; how do they improve on the original specification notation (VVSL)? (C, S)*

   The mathematical languages of Event-B and VVSL, in most cases, are equally expressive. The key difference was not the notation; rather it was the style of specification used in the Event-B development. In particular the use of refinement, whereby details of functionality was introduced in layers, led to a more comprehensible specification. The layered approach, along with the contemporary interface of the RODIN tool and its progressive prover, made it possible to mechanically check the models, produce comprehensive proof obligations and discharge all proof obligations either automatically or semi-automatically.

   In a few small cases, such as syntax support for “record types”, VVSL has advantages, which could be added to the Event-B language.

It is worth emphasizing that the CDIS specification is necessarily complicated. Even though the core specification has been criticized for its complexity, it is unrealistic to expect any significant improvements in the size of a specification that captures all aspects of CDIS, regardless of the notation used. However, the bottom-up construction in VVSL forces a level of specification that is too detailed to get an appreciation of the overall system behaviour.

Grade: [5]

4. *As a result of any improved expressiveness, how much additional verification has been made possible by the tool? (X, S)*

   After porting the B models from year two to the RODIN toolset, we have discharged all proofs either automatically or via the interactive prover. The RODIN tool now generates a wider range of proof obligations, compared with B4free. These proof obligations include consistency proof (Feasibility and Invariance Preservation) and Well-Definedness. Separation of context and models, according to the new methodology recommended by RODIN, has made the proof generation and the discharging process very easy. Furthermore the layered development eased the proof of consistency of the specification, since at each step we had a small number of relatively simple proof obligations. In both cases of centralised and distributed models of CDIS most of proofs were discharged automatically. A few numbers of proof obligations, which would not prove automatically, were proved very easily using the interactive prover.

   Grade: [5]

5. *How well has the subset specification complexity and size challenged the space and time limitations of the RODIN tools; can we draw conclusions about whether the tools can cope with the entire original specification?(X)*

   The final refinement level of the CDIS system comprises more than seven pages of Event-B. Also this is a sizable piece of code, which includes complex definitions and different relations, functions and other operators. This didn't challenge the RODIN tool. Despite the above, there some minor cases where performance and efficiency of the tools could be improved. These include workspace reconstruction and proof obligation generation after new changes are introduced into the model and some aspects of the automatic prover.

   In addition to these points, handling and presentation of long and complex guard lines and assignments should be improved to ease formal model management.

   Grade: [4]

6. *How many errors and inconsistencies in (the chosen subset of) the original specification have been identified? (C, S)*

   So far no errors or inconsistencies have been found in the original specification. Some missing features were identified, but these omissions turned out to be a result of the year one sub setting exercise.

   Grade: [4]

5.4.7 Conclusion

In conclusion, it is clear from the above section that the main tool platform has reached a reasonable level of maturity and stability. In comparison to the previous generation of similar tools such as B4free and AtelierB, the tool is much more productive and the interface is much easier to use.

The new adopted methodology provides a much more modular approach to system modelling. Furthermore it uses fewer constructs in comparison to standard B and therefore it should be much easier to deploy.

5.5 CS5 – Ambient Campus Assessment

5.5.1 Introduction

This case study aims to identify the extent to which various parts of the RODIN approach can provide effective support for the most challenging stages of the formal design process of complex fault-tolerant mobile systems. In particular, the wireless communication medium, on which the implementation part of this case study is based, typically causes transmission errors leading to a whole range of critical faults that must be tolerated. Moreover, such mobile applications need to deal with a variety of abnormal and unpredictable events due to system openness, mobility of its participants and their dynamic nature.

The work on the Ambient Campus case study (from the Description of Work [1]) has focussed on:

a elucidation of the specific fault tolerance and modelling techniques appropriate for the application domain,
b validation of the methodology developed in WP2 and the model checking plug-in for verification based on partial-order reductions, and
c documentation of the experience in the forms of guidelines and fault tolerance patterns.

More specifically, in this case study we have been investigating how to use formal methods combined with advanced fault tolerance techniques in developing highly dependable Ambient Intelligence (AmI) applications. In particular, we have been developing modelling and design templates for fault tolerant, adaptable and reconfigurable software. The case study covers the development of several working ambient applications (referred to as scenarios) supporting various educational and research activities (see D26). During year 3, we have worked on the *Student induction* scenario, where we have developed a system providing assistance to new students in the registration process at the beginning of the term and in familiarising themselves with the campus environment.

We drew on the *Context-Aware Mobile Agents* (CAMA) abstraction, framework and middleware (which were developed in Year 1 and Year 2) as the building block of our work in Year 3. The CAMA middleware architecture was formally developed using the B Method. This allowed us to verify the properties of the scoping mechanism and the ability of agents to tolerate disconnections. The result of the modelling activity was used to implement various parts of the middleware, most notably the scope-related operations. Exception handling has proved to be the most general fault tolerance technique; as it allows effective application-specific recovery. If exception handling is to make programmers' work more productive and less error-prone, the programming and execution environments need to provide it with adequate support.

5.5.2 Current Status

Year	Achievement	Objective	Papers
1	Traceable requirements document for the ambient campus case study	a), c)	[4]
1-2	Development of the Context-Aware Mobile Agents (CAMA) framework and middleware	a), b), c)	[8(pp. 63-75),14,61, 62,63,64]
1-2	Development of the first ambient campus scenario (*ambient lecture* scenario)	a), b)	[65,66,67]
1-2	Model checking CAMA systems	a)	[64]
2	Development of the second ambient campus scenario (*presentation assistant* scenario)	a), b)	
2	Finalising the set of CAMA abstractions	a)	
2-3	Refinement and modification of the CAMA framework and middleware	a), b), c)	[69]
2-3	Methodology for formal development of open Multi-Agent Systems	a)	[66]
3	Mobility Checker plug-in for verification of liveness and mobility properties of multi-agent systems modelled using the Event-B formalism	a)	[68]
3	Formal refinement patterns for Event-B method, development of patterns for software fault-tolerance	c)	[70]
3	Plug-in to the RODIN platform for automated application of refinement patterns	b), c)	[71]
3	Involvement in the Advanced MSc student group work at the Newcastle University's School of Computing Science	b), c)	
3	Development of the third ambient campus scenario (*student induction* scenario) using RODIN tools and methods	a), b), c)	[68]

Table 10: Case Study 5 - Annual achievements against objectives (c.f. §5.5.1 above)

5.5.3 Progress since Year 2 Assessment

During year 3 we have completed and finalised the following work: (see D26 [18] and D27 [19] for more details):

- **Methodology:** During years one and two, we developed a framework called CAMA (Context-Aware Mobile Agents), which consists of:
    - a set of fundamental abstractions used in the formal development of ambient systems,
    - verification of properties of their models,
    - a formal design of the CAMA middleware using the B method,

- an implementation of these systems.

In the first two years there was a considerable progress in understanding how agent system development can benefit from formalisation and verification. We consider that formally applied, top-down developments, of agent interaction protocols to be the only complete and rigorous software engineering technique in design of open systems. We recognise that formal methods are not easy to use and the associated costs can be very high. To facilitate adoption of the RODIN formal modelling framework as the mainstream software engineer tool, we have proposed a set of abstract design patterns that provide general guidance during a formal development and also a tool and a set of refinement patterns that considerably reduce development costs. Refinement patterns are formally described reusable model transformation rules. Pattern correctness is proved once and all refinements produced using a pattern are automatically correct, which results in a considerable decrease in number of proof obligations.

To facilitate adopting RODIN's formal modelling framework as a mainstream software engineering tool, during year three we have developed a set of abstract design patterns that provide general guidance during a formal development and also a tool and a set of refinement patterns that considerably reduce development costs. Refinement patterns are formally described reusable model transformation rules. Pattern correctness is proved once, and all refinements produced using a pattern are automatically correct, which results in a considerable decrease in proof obligations.

As part of our work on this case study, we proposed a formal modelling based approach to developing MAS, which should be capable of capturing both the functional model (e.g. what kind of computations an agent is capable of doing) and the behavioural model of an agent (e.g. how an agent moves, how it interacts with other agents, etc.). While it possible to use just the Event-B notation (provided by the RODIN platform) to describe and verify statically the functional model of an agent, it is quite challenging or even impossible to do the same with the behavioural model. In this novel approach, we introduced a hybrid (Event-B combined with constructs inspired by process algebras) high-level programming notation for the specification of mobile applications that can faithfully capture both the behavioural and the functional model of an agent.

Plug-ins: In year three of the project, the development of the mobility plug-in progressed on two different fronts. On the theoretical front, we have developed a hybrid high-level programming notation that is capable of modelling mobile agent systems. Furthermore, in order to utilise our existing efficient model-checking engine, it was necessary to provide a translation of this language into Petri nets. Using the theoretical results developed in year two of the RODIN project for the translation of pi-calculus to Petri nets, it was reasonably straightforward to translate this new language into Petri nets. On the implementation front, we have developed a tool that is properly integrated and uses the RODIN platform functionality. An internal version of the plug-in was released in March 2007 following the plans for the 30M deliverables. The final public version of the plug-in, together with its full

supporting documentation, is scheduled for release according to plan in September 2007. We have completed this work, and we have also carried out the evaluation of the plug-ins relevant to this case study (c.f. §5.5.4 below).

- **Advanced MSc student group work:** as part of the evaluation and assessment tasks, we have conducted an extensive work on teaching two groups of Advanced MSc students at Newcastle University's School of Computing Science about CAMA abstractions, middleware and tool support and on getting feedback from them. Two (separate) systems were designed and developed by the two groups:

    - *eCampus Meeting System:* provides electronic support for meetings, where the meeting's chairperson can set up the meeting, add topics for the meeting and invite others to participate in the meeting. The participants can then post comments, vote on issues discussed in the meeting, and view the minutes of the meeting on the meeting website. Formal modelling through the RODIN platform was used in designing the system, and the system was implemented using the CAMA middleware, as well as database and web servers.

    - *Student and Faculty Interaction System:* allows students and faculty (university staff) to interact as they go about their daily activities on campus, in particular regarding lectures and campus event. A location-based scheme (context-awareness) was simulated, allowing the students and staff to receive services appropriate to their current location. As with the other group, the RODIN platform was used for the formal modelling, and the implementation was based on CAMA.

Student induction scenario: During year three, we have developed the third and final scenario for the ambient campus case study. This scenario demonstrates the ability of agents to migrate logically from one platform to another using the CAMA framework. We used formal specifications followed by step-wise refinement in designing this scenario. We also introduced smartdust devices [72], which provide features for automatic location detection; thus enabling specific services to be delivered in each predefined location. We have also developed a set of demonstrators for this scenario, including models, templates, screenshots and a demo.

5.5.4 Contribution to the Development of Platform and Plug-ins

RODIN Platform

The RODIN platform was used extensively by case study five during year three. The case study five modelling was one of the first applications of the platform in the context of realistic, large-scale specifications. Few problems have been found; mainly with the tool interface and these were promptly addressed by the platform developers. As a result of the study case study five raised four feature requests and six separate bug reports. All bug reports have been closed. Table 11 identifies the feature requests raised by case study five.

ID	Title	Description
#1595966	Sequence types	Sequence types have been requested. Specifically, it would be good to have the classic B concatenate (a ^ b) and explicit specification ([a,b,c,d]) functions.
#1592016	Automatic Completion	When writing Event-B automatic completion (equivalent to Eclipse's ctrl+backspace) would be really helpful.
#1609359	Empty action	When the action is empty, the kernel complains about an invalid assignment. *A warning has now been generated to indicate that an empty maths element has been found whenever this problem occurs with expressions, predicates, or assignments.*
#1793844	Decomposition Support	Support for decomposition (and ideally instantiation as well) is vital if RODIN is to be of practical use.

Table 11: Feature requests raised by case study five

Table 12 highlights the most significant bug reports generated by case study five.

ID	Title	Description
#1609712	Space required after some names	It took me while to figure out that I have to put space after keywords like NAT, INT and etc.
#1665991	Search does not work	Hypothesis search in the prover interface has the following problems: 1 after a search there is no way to select hyp.'s in the bottom pane; 2 search hyp.'s in the proof view do not appear.
#1741022	Syntax error: EOF expected	In version 0.7.4 this error occurs for any expressions containing & and "or".
#1742823	Comments	1 It is hard to open the pop-up window to edit/view a comment (may be Linux-specific). 2 Comments do not need keyboard translations, unless it is expected that users type in formulae instead of text.

Table 12: Significant bug reports generated by case study five

Mobility Checker Plug-In

The ambient campus case study was the main source of requirements used for the building of the mobility plug-in. The feedback received from the case study drove the

development and the addition of features in the tool. Based on the theory and the development approach used in the building of the plug-in, it is possible to model check an Event-B specification (deadlock detection and invariant violations) for free. As a result it will also be possible for case studies that use the ProB model checker to experiment with the mobility plug-in as well.

ProB Plug-In

The ProB plug-in is an essential tool for understanding complex models. Large, involved specifications are hard to read, even more so in Event-B, where specifications tend to have large number of events due to the absence of sequential composition. Model animation is efficient and user friendly for model interpretation.

B2RODIN Plug-In

This plug-in has been developed to transfer AtelierB projects into the new RODIN platform. The plug-in is extremely simple to use and no issues have been found. We have applied the B2RODIN plug-in to transfer previous AtelierB and Click'n'Proof developments into the new RODIN Platform. The plug-in performance was satisfactory and it is very easy to use.

5.5.5 Contribution to the Integration Objectives

Case study five's primary integration focus has been on objectives a) and b).

a) Checking the scaleability of the system as its functionality is extended

This objective is defined below:

> *Checking the scaleability of the system as its functionality is extended* – how does adding a plug-in affect the performance of the system, does it get harder to develop plug-ins (and integrate them into the platform) as the number of plug-ins grows, and how does the usability of the system decline as the number of plug-ins increases?

We have used the full internal versions of the RODIN toolset (including the platform, mobility, B2RODIN, model-based testing and ProB plug-ins) for developing the student induction scenario. The previous versions of the platform and the plug-ins were partially used during years one and two for developing the ambient lecture scenario and the presentation assistant scenario. These experiments have not indicated any scaleability concerns related to integration of new plug-ins into the RODIN platform. The Eclipse framework supports a number of useful ways of virtualising the development environment and avoiding any overheads caused by the use of many plug-ins. Consequently it handles the scaleability issues very well.

Grade: [4]

b) Checking the impact of legacy (sub) systems

This objective is defined below:

> *Checking the impact of legacy (sub)systems* – how does the 'correct through construction' refinement-based methodology integrate with processes where already developed models have to be re-used, is reverse engineering a reasonable approach, and which plug-in(s) will support this, if any?

We are currently at the initial stages of integrating refinement-based methodology with processes, where previously developed models have to be re-used. The main reason for this is that full advantage of the refinement-based methodology complemented by model-checking approach to the verification of mobile systems was not possible until the last phase of the work. The approach, which in our view would be successful, is one where an already developed model is translated into the high-level Petri net formalism, which underpins our model-checking approach. We have already applied such a translation to complex B models developed prior to RODIN. The mobility plug-in would then be very well suited for the verification task involving both pre-existing and newly developed parts of the overall specification.

Grade: [3]

5.5.6 Case Study Specific Metrics

1. *Given the four AmI scenarios developed previously by the ISTAG group, analyse how much has the use of formal techniques improved the original ideas about this application? (C, U, X)*

   Transition from the initial ideas to the formal models was made more explicit and manageable thanks to the application of the requirement development methods proposed by J-R. Abrial (see D8 [8]). Furthermore, we applied the stepwise refinement technique of the B method to develop a concrete model for the case study, which forms the base of the demonstration delivered at the end of Year two. In particular, by introducing error recovery at the early stages of design, this allowed us to address some key aspects of fault tolerance more rigorously.

   In our work we have developed a method, which combines state-based and process-based modelling. This will allow us to support stepwise development using the B method alongside the modelling and automatic verification of complex temporal properties, including those related to mobility.

   With respect to our experience with teaching the advanced MSc students, we described the ISTAG scenarios [74] to introduce them to this area and to help them select the specific scenarios they would implement.

   Grade: [4]

2. *How well do RODIN methods and tools fit to the four ISTAG scenarios, in particular Scenario 4 Annette and Solomon in the Ambient for Social Learning? (C, U)*

   We were able to capture those aspects of ISTAG scenarios [74] that are relevant to coordination, mobility and fault-tolerance using the CAMA framework. The concepts of scopes and agents could be successfully applied in developing Scenario four, although clearly we do not have answers to all the technological challenges raised. Moreover, scoping can be used for isolation of the private conversations between the mentor, the students and the ambients (the issue not mentioned in the original ISTAG scenario) as well as for implementing general meetings taking place in the dedicated room (location in CAMA terms). CAMA concepts of roles as well as coordination model can be applied to ensure consistent access to the shared resources typical for the ISTAG scenarios. The Ambient Social Learning space can be implemented using the CAMA framework; this ensures efficient cooperation whilst preserving de-coupling of actors, as well as system openness. Scenarios one and three, which have physical mobility, can be suitably modelled by CAMA agents migrating between locations.

   As the ISTAG scenarios have neither rigorously defined requirements nor formally defined specifications, we believe that modelling them using the CAMA framework, coupled with the assistance of the RODIN method, is an important contribution to the understanding and refining of ISTAG scenarios.

   Scenario three fits exactly the idea of scenario four (pages 43-48 of [74]). It has been developed to support campus activities, which involve students and teachers moving in the campus and joining various lecture rooms. This is directly related to the idea of accessing various learning spaces and can be implemented in a straightforward way using the logical mobility which CAMA and the mobility plug-in supports.

   The two systems, which Newcastle students developed, in fact represent functionality, which can be viewed as part of scenario four.

   Grade: [4]

3. *How clearly can we show that we have enhanced dependability and fault tolerance? (O, S)*

   There is clear lack of understanding of how to approach the problem of error recovery in multi-agent systems. There are a number of proposed solutions tackling different aspects of the problem; but there is still no systematic approach encompassing all the stages of design and lifetime of multi-agent systems.

   The CAMA framework and formal development allowed us to develop fault tolerant systems with much more effective application level recovery. Formal

reasoning about fault tolerance properties, based on refinement and verification, is a distinct characteristic of our approach; as this is not supported by any of the existing methods. We are working on a formalisation of an exception propagation mechanism that is to be integrated into the formal agent design process.

Students have been able to tackle harder problems quicker and with less mistakes and errors using our software and tools. It would be simply impossible for them to develop the same two systems in the same limited time without their use. They used a number of the specific fault tolerance mechanisms from CAMA – among others, error detection (disconnections), as well as restricting and controlling the number of participants in the scope.

Grade: [3]

4. *How well has the crystallization of ideas via the formal specification improved early discussion of the system requirements, especially whether the requirements are fit for purpose? (C, S)*

   The effort input to the preparation of the requirement document and the formal models has clearly benefited the work on formal specifications for the case study scenario. During this work we discovered a number of omissions and inconsistencies in the requirement document, which resulted in several iterations of work on the requirement document and the subsequent alterations of the formal model. Our experience demonstrated the importance of describing informally relevant decisions about a system design while undertaking formal modelling and development using the refinement method.

   Due to the Newcastle MSc students' limited time available to conduct the group work (8 weeks) – and the fact that these two groups had to learn system design and modelling using the RODIN platform from scratch – the development approach was not strictly speaking a top-down development. Nevertheless, the students learned a lot about the systems they were developing during modelling and feedback from the modelling stage helped them to improve the implementation.

   Grade: [3]

5. *How fully does the implemented case study use the fault tolerance techniques and methodology proposed and how closely they match the case study requirements? (U, X)*

   Currently a preliminary implementation of the proposed error recovery mechanisms has been applied in the case study. More specifically, we employed the exception propagation mechanism to provide cooperative recovery in the situations where there is an inconsistency in the observations of the global state caused by a fault.

Students have clearly enjoyed the working with the CAMA environment; they very quickly learned how to use it, used the full functionality and needed little introduction. It was a very positive experience.

During year three, we found that the techniques provided through CAMA are powerful enough to support the vast majority of the fault tolerance needs, which need to be addressed by the developers of mobile distributed systems. The concepts of scopes and exceptions are crucial in addressing the fault tolerance issues. It is also observed that plug-ins can be used to extend CAMA features. We therefore believe that scaleability is addressed sufficiently in CAMA.

Grade: [3]

6. *How general are the development method and the fault tolerance techniques, judging by their application in several Ambient Campus scenarios? (U, X)*

   The fault tolerance mechanisms provided by the CAMA abstractions, and supported by the CAMA middleware, provide nested scoping for error confinement and flexible exception handling for application specific error recovery. The middleware itself detects disconnections and crashes of the PDAs – the most typical failures for the ambient systems – and raises predefined exceptions to be handled in the agents participating in the corresponding scope. These general mechanisms were successfully applied in the context of the Ambient Lecture scenario and, in particular, in supporting the student group work. The development method, which is based on the B method and process algebra, and used in modelling and designing the scenarios, is general enough, as it is relies on the general concepts of scopes, agents, roles and locations, which are typical for a wide range of ambient applications. The proofs were completed using AtelierB, except for about dozen interactive proofs; the *prover* automatically discharged all proofs.

   Grade: [4]

5.5.7 Conclusion

Case study five has developed and investigated a novel approach for modelling and verifying the correctness of complex mobile agent systems. None of the existing languages were capable of capturing their complete behaviour. Our achievement in year three has been the development of a single hybrid (Event-B together with a process algebra with mobility characteristics) high-level programming notation that is capable of capturing both the behavioural and the functional model of agents. This language has strong theoretical foundations and its structured operational semantics are also presented here. Finally, an efficient model checker has been developed as a plug-in for the RODIN platform. The plan for this tool is to support a significant part of the Event-B notation and also behaviourally rich process algebra expressions. Within case study five, a number of refinement patterns were investigated. We have developed in year three a number of patterns that help to automate design of systems with rich behaviour but shallow functionality. Many mobile agent system protocols and abstractions can be

modelled adequately simply by the composition of these patterns. There are also a number of patterns that help introduce inter-agent communications. The modelling of the case study in year three was one of the first applications of the RODIN platform in the context of realistic, large-scale specifications.

SECTION 6 OPEN TOOL KERNEL ASSESSMENT

6.1 Introduction

The objectives of work package 3 (from the Description of Work [1]) are to develop some *basic kernel tools* implemented on a certain *platform container* that can be extended by the *plug-ins* being developed in work package 4.

To ensure the openness of the platform, we planned to:

a Finalize the Event-B *Language*.
b Provide an adaptation of the *Low Level Basic Tools: Static Analyzers*.
c Provide an adaptation of the *Intermediate Level Basic Tools: Proof Obligation Generators*.
d Provide an adaptation of the *Upper Level Basic Tools: Provers*.
e Design and implement the *Connection Language*.
f Design and implement the *Project Manager*.
g Design and implement the *Open Platform*.

6.2 Current Status

Year	Achievement	Objective	Papers
1	Major technical decisions concerning the platform container and plug-in mechanism taken and recorded.		D3.1[5]
2	Event-B language definition finalised	a	D3.2[6]
2	Overall platform architecture, plug-in mechanism and Event-B kernel tools specified.	b, c, d	D3.3[10]
	Platform prototype and Event-B kernel tools developed and delivered.	b, c, d	D3.4[12]
3	Implementation of the platform and the Event-B kernel tools is on going. An internal version has been delivered in month 30.	e, f, g	D3.5[17]
3	Public version of the Event-B kernel tools delivered.	e, f, g	D3.6[22]

Table 13: RODIN Kernel - Annual achievements against objectives (c.f. §6.1 above)

6.3 Progress since Year 2 Assessment

The progress of this work package is on schedule with respect to the objectives in the Description of Work [1]. Notably, the following milestones have been achieved during Year 3:

- M3.3: Complete platform integrating WP4 plug-ins

Plug-ins are now available via SourceForge for downloading and can alternatively be installed by means of the update and install mechanism built into Eclipse.

6.4 Interaction with Plug-in Developers

Various project partners have contributed to the RODIN platform using the extension mechanisms provided. Examples of the interaction include:

- ClearSy and Southampton have used the extensibility of the RODIN database to add elements that they needed to the Event-B database.
- ClearSy's animator extends the Event-B editor for specifying animation parameters.
- Düsseldorf has (partly) ported ProB to the RODIN platform making use of the Event-B database (this contribution uses mostly the extension mechanisms already present in Eclipse).
- Furthermore Düsseldorf has contributed a plug-in for the prover, with symbolic model checking of goals that can also generate counter examples.

6.5 Kernel Metrics

6.5.1 Requirements and Functionality

1 *How well does the tool perform its stated purpose? (C)*

All essential features of the platform and kernel tools have been implemented. The platform is stable and is used actively by the project members.

Grade: [5]

2 *How rigorously is the required tool behaviour defined? (O, S, U)*

All behaviours of the kernel tool are defined formally. For instance:

- the Static Checker is defined using both ad-hoc formalisms (BNF syntax for the parser, attributed grammar for well-formedness and type-checking) and an Event-B model (for graph checking)
- the Proof Obligation Generator specification is derived from the Event-B language definition, applying some simple mathematical transformations.

Grade: [5]

3 *How much does it contribute to system correctness? (C, S)*

As kernel tools provide means for mathematically proving system correctness, their use provides correctness by construction.

Grade: [5]

4 *How much does it extend/shrink system development and testing phases? (C)*

This criterion is not directly relevant to work package 3. It is assessed in the case-studies work package.

Grade: [N/A]

6.5.2 Tool Usability

1 *How long does it take a developer, who is knowledgeable in the specification language used, to learn how to use the tool effectively? (U)*

This criterion is not directly relevant to work package 3. It is assessed in the case-studies work package.

Grade: [N/A]

2 *How long does it take the tool to run to completion on a specification of representative size? (C)*

The performance of the tools has been improved considerably by implementation of differential static checking, proof obligation generation, and proving. More studies are needed to judge the performance with respect to very large specifications.

Grade: [4]

3 *What is the tool's response time to a change by a user to the specification? (U)*

One of the major requirements when designing the tools was to take into account the need for incremental development of models. Consequently, the tools perform very well when modelling small changes.

Grade: [5]

4 *What is the cost of the hardware and operating system required to run the tool at an acceptable speed? (C)*

The tool runs on a standard PC. There is no requirement to add special hardware or software.

Grade: [4]

5 *What is the cost of the tool licence for one, five or twenty users for a year, including support? (C)*

The tools are freely available on SourceForge. Hence, no licence cost is incurred. With regard to support, no commercial offer is yet available. However, during the course of the RODIN project, free support has been provided through SourceForge.

Grade: [5]

6.5.3 Development and Integrity of the Tool

1 *How quickly can an identified tool bug be fixed in tool's code? (U, X, O)*

Most of the bugs found in the tool during development where fixed in less than a day. A key factor for the ease of bug fixing is due to the fact that kernel tools always provide a trace relating their output to their input. Hence, fault location can be very straightforward.

Grade: [4]

2 *How quickly can an identified tool bug be fixed in the development and testing system? (U, X, O)*

All tests have been developed using the JUnit framework. Hence, it is very easy to add a new test to the test database (just add a new method to an existing test class). It is also very easy to rerun all tests on a tool (just one button click). As concerns the error database, it is located on the SourceForge site and easy to access through any Web browser.

Grade: [5]

3 *How quickly can minor and major new tool features be implemented? (X)*

All tools have been designed and implemented with extensibility in mind. As a consequence, they all provide extension points so that new functionality can be added with minimum effort using the plug-in mechanism of Eclipse.

Grade: [5]

4 *How long is the time required to bring in and educate a typical tool developer? (U, X)*

The tools are developed in Java using the Eclipse Java Development Tools, which are quite widespread. Hence, most developers know the development environment or are familiar with a very similar setting. For instance, ETH Zurich had two undergraduate students working on the tools, and they didn't need any time to learn how to use the tool (although neither Java, nor Eclipse are taught at ETH).

Configuration management is done using CVS on SourceForge, which is also a standard tool.

Grade: [4]

5 *How many operating systems and architectures are supported and how many are possible? (U)*

The RODIN kernel is based on the Eclipse platform and developed in Java. Consequently the tools can be made available on all standard platforms, i.e.: Windows, Mac OS X, Linux, Solaris, AIX and HP-UX. However, due to a lack of test platforms, only a subset has been tested. Currently the tools run on three major computer platforms: PC/Windows, PC/Linux and Mac/MacOS X.

Grade: [4]

6 *How extensive are unit, functional and system testing of the tool? (U, S)*

Some unit tests have been undertaken for sensitive elements of the tools (computation intensive parts). Extensive functional and system testing has been developed for the Event-B tools.

An exploratory development approach has been adopted for the user-interface, which is still maturing. Consequently this element relies on active evaluation by project members with an increasing involvement of functional testing.

Grade: [4]

7 *How easy is it to compare two versions of the tool? (U, S)*

All tests are carried out using the JUnit framework. As a consequence, test results are self-evaluated by the tests themselves, which makes assessing the test results straightforward. Therefore, it is very easy to compare two versions of the tools.

Grade: [5]

8 *How well does the tool admit self-analysis? (U, S)*

The kernel tools do not admit self-analysis. They can prove a model correct by construction but are not relevant for proving a program directly. The use of a code-generation plug-in might alleviate that, but none was available during the kernel tool development and no bootstrapping is planned.

Grade: [N/A]

9 *How reliable are error tracking and regression testing? (U, S)*

All errors discovered in the tools give rise to the development of one or more tests that give evidence that the error has indeed been fixed. All these tests accumulated during the initial development and bug fixing are run in a systematic fashion.

Grade: [5]

10 *What is the self-fault detection history, and in particular how many faults does the trend predict in the current tool? (S)*

It is currently too early to apply an analysis to the fault detection history. The number of incoming fault reports is small but so is the number of users.

Grade: [N/A]

11 *What classes of self-fault are detectable or not detectable in the tool? (S)*

The tools have been developed defensively; they undertake a lot of internal integrity checks (using assertions for instance). All these checks give rise to logging of all cases where an internal inconsistency has been detected.

Grade: [4]

6.5.4 Input (source) Language Issues

1 *What fraction of the possible source language grammar does the tool accept, analyse, and analyse correctly? (U, S, X)*

The support of the Event-B notation of the current version of the tools is almost complete. Only external variables and deadlock-freeness are missing. However, Event-B can already be used efficiently without them.

Grade: [4]

2 *If the source language is based on an external definition e.g. an ISO standard, how much must it change to be acceptable to the tool? (U)*

There is no external definition of Event-B, nor any other tool implementation of the notation. Hence, the RODIN tools are the reference implementation.

Grade: [N/A]

3 *What is the earliest point in system development when a source document may be analysed? (C)*

Source documents can be analysed at any point in time.

Grade: [5]

6.5.5 Output Format

1 *How easy is it to auto-parse the tool output? (U, X)*

Tools output is provided in two forms. Error messages are output as Eclipse markers which can be analysed very easily with a Java plug-in. Proofs are stored in the RODIN platform database which is accessible either by a Java plug-in, or directly as an XML document.

Grade: [5]

2 *What level of control over the output volume and content is allowed? (U, X)*

The RODIN user interface provides various views on the tools output. For instance, for proofs, one can see the proof status of a component, or a list of all proof obligations together with their proof status (discharged or not), or for each proof obligation, a detailed view of its proof tree. Also, the user interface provides various means for filtering the tools output; so that the user can choose precisely which parts he wants to see.

Grade: [5]

3 *How well does the output relate to the input, for instance for error reporting? (C, U)*

All transformations made by the tools (most notably proof obligation generation) are fully traced to their source. As a consequence, when a proof obligation is not provable, it's very easy to trace it back to the source elements that gave rise to it. We expect that this will allow user to find the cause of errors much more easily than with previous formal tools.

Grade: [5]

6.6 Conclusion

The core RODIN platform comprises:

- RODIN database,
- Event-B database,
- Proof obligation manager,
- New predicate prover,
- Proof obligation generator,
- Static checker,
- Event-B editor user interface, and
- Event-B prover user interface.

These components are stable and sufficiently powerful for practical use.

Large-scale trials and/or case studies still have to be carried out to confirm that the tool continues to scale-up. This has certainly been the case for the medium size case studies attempted during the RODIN project.

The design goal of extensibility has been achieved, as demonstrated by the various plug-ins already developed.

The RODIN project demonstrates that the platform is viable and sets new standards for formal methods tools in academia and industry.

SECTION 7 PLUG-IN ASSESSMENTS

7.1 Mobility plug-in (Mobile B Systems)

7.1.1 Introduction

Mobile agent systems (MAS) are complex distributed systems, which are built from asynchronously communicating mobile autonomous components. Such systems have a number of advantages over traditional distributed systems, including: ease of deployment, low maintenance cost, excellent scaleability, autonomous reconfiguration and effective use of infrastructure. MAS are distinct enough to require specialised software engineering techniques. A number of methodologies, frameworks and middleware systems have been proposed to support rapid development of MAS applications. However, currently there is no single widely recognised standard and the problem of building large and dependable MAS remains open.

While it is possible to use just the Event-B notation (provided by the RODIN platform) to describe the functional model of an agent and statically verify it, it is quite challenging or even impossible to do the same with the behavioural model [45,46]. Based on a novel approach presented in [47], we developed a hybrid high level programming notation (Event-B combined with constructs inspired by two process algebras: KLAIM [48,49] and pi-calculus [50,51] which is capable of capturing faithfully both the behavioural and the functional model of an agent. By using a combination of static verification and model checking we are able to offer two different views on a model and carry out complimentary analysis of functional and dynamic properties. Since MAS are highly concurrent the state space explosion problem is present during model checking. One should therefore use an approach that alleviates this problem; in our case, based on partial order semantics of concurrency and the corresponding Petri net unfoldings [52]. A finite and complete unfolding prefix of a Petri net *PN* is a finite acyclic net, which implicitly represents all the reachable states of *PN* together with transitions enabled at those states. Efficient algorithms exist for building such prefixes [53], and complete prefixes are often exponentially smaller than the corresponding state graphs, especially for highly concurrent systems, because they represent concurrency directly rather than by multidimensional "diamonds" as it is done in state graphs. For example, if the original Petri net consists of 100 transitions which can fire once in parallel, the state graph will be a 100-dimensional hypercube with 2^{100} vertices, whereas the complete prefix will be isomorphic to the net itself. Since mobile systems are usually highly concurrent, their unfolding prefixes are often much more compact than the corresponding state graphs. In order to take advantage of the compact representation provided by PN unfoldings, the newly developed hybrid language is translated into Petri nets.

7.1.2 Current Status

During the three years of the project the following goals have been achieved:

- Extension to a full recursive variant of π-calculus [54] for the theoretical and algorithmic foundations of the compositional translation from the π-calculus to Petri nets, which were first developed for its finite fragment [55].
- Tool support (standalone) for the finite fragment of pi-calculus [58] (it should be emphasized that this tool appears to perform well both in speed and scaleability terms; much better than the 'state-of-art' Mobility Workbench).
- Extension of the previous developments to the KLAIM based process algebra [56].
- A new efficient method for computing the shortest violation traces in the Petri net unfolding approach [57].
- Introduction of a high-level programming notation for the specification of mobile agent systems. This new modelling language is a hybrid of Event-B together with a process algebra with mobility characteristics that can faithfully capture both the behavioural and the functional model of an agent.
- Tool support for the automatic verification of mobile agent systems. This tool has been properly integrated with the RODIN platform (plug-in to the platform).
- A proposal to develop algorithms for efficient implementation of the model-checking kernel of the mobility plug-in [59]. The paper introduces a new condensed representation of a Petri net's behaviour which copes well not only with concurrency, but also with other sources of state space explosion, such as sequences of non-deterministic choices.

7.1.3 Progress since Year 2 Assessment

In general, the work progresses according to the milestones listed in the RODIN DoW. In particular, in addition to the progress reported a year ago, in year three of the project we have made definite progress on two different fronts.

On the theoretical front, we have developed a hybrid high-level programming notation that is capable of modelling mobile agent systems. Furthermore, in order to utilise our existing efficient model-checking engine, it was necessary to provide a translation of this language into Petri nets. Using the theoretical results developed in year two of the RODIN project for the translation of pi-calculus to Petri nets, it was reasonably straightforward to translate this new language into Petri nets.

On the implementation front, we have developed a tool that is properly integrated and uses the RODIN platform functionality. The experience obtained in year two of RODIN, from building the standalone tool for the automatic verification of finite pi-calculus, was extremely helpful in achieving this goal. An internal version of the plug-in was released in March 2007 following the plans for the 30M deliverables. At the moment we are working on a public release version of the plug-in together with its full supporting

documentation. Furthermore we are looking for opportunities to test the performance of the automatic verifier for the recently developed models.

7.1.4 Integration with the Platform

The mobility plug-in has been properly integrated with the platform and uses several features provided by the RODIN platform and the Eclipse IDE e.g. access to database, extensions to database, Eclipse user interface, etc.

7.1.5 Use by the Case Studies

The mobility plug-in is used by the Ambient Campus (CS5) case study. The first useable version of this plug-in was made available in March 2007. Actually, this case study was the main source of requirements used for the building of this tool. The feedback received from the case study drove the development and the addition of features in the tool. Based on the development approach used in the building of the plug-in, it is possible to model check an Event-B specification (deadlock detection and invariant violations) for free. As a result it is also possible for every case study that uses the ProB model checker to experiment with the mobility plug-in.

7.1.6 Plug-in Metrics

Requirements and functionality

1 *How well does the tool perform its stated purpose? (C)*

The tool was designed to perform automatic verification of mobile agent systems that we modelled with the newly invented hybrid programming language. Since there is no other tool or language that can faithfully capture the full behaviour (both functional and behavioural) of mobile agent systems we are unable to provide comparative results. On the other hand, the tool will be used for the automatic verification of models developed in the Ambient Campus case study. Since these models are expected to be fairly complex, they will provide a good measure of the performance of the plug-in. Furthermore since we are getting the functionality of model checking Event-B specifications for free, it will possible to have some comparative results with the ProB model checker.

Grade: [4]

2 *How rigorously is the required tool behaviour defined? (O, S, U)*

The tool's behaviour has full theoretical underpinnings in the form of research papers and proofs dealing with the translation from hybrid programming language to Petri nets, as well as with Petri net unfolding theory.

Grade: [5]

3 *How much does it contribute to system correctness? (C, S)*

This is an expected contribution based on the previous advancements made using Petri net-based model checking. Furthermore, it is now possible to contribute to the system correctness of mobile agent systems. The combination of Event-B and process algebra will add to the correctness of the system since we can now combine static verification and model checking, we can offer two different views on a model and carry out complimentary analysis of functional and dynamic properties.

Grade: [5]

4 *How much does it extend/shrink system development and testing phases? (C)*

Model checking is a technique that is particularly effective in detecting concurrency related defects at an early stage of system design. As a result, the use of the tool can shrink system development and testing phases significantly.

However, the main benefit is in verifying correctness criteria rather than relying on testing.

Grade: [4]

Tool usability

1 *How long does it take a developer, who is knowledgeable in the specification language used, to learn how to use the tool effectively? (U)*

Someone, who is familiar with the specification language, should require a small amount of learning time to use the tool. The required steps to operate the tool, after building the specification, are automatic and require minimal user interaction. Furthermore, a large amount of the tool's functionality (e.g. the functional part of a mobile agent system is modelled on the Event-B notation) comes for free from the RODIN platform.

Grade: [5]

2 *How long does it take the tool to run to completion on a specification of representative size? (C)*

The main engine of the tool has a proven performance record in the model checking community. Moreover, scaleable experiments performed in year two, to model check π-calculus specifications [58], show much better performance when compared with other 'state of the art' tools. Finally, the models developed in the Ambient Campus case study are expected to be fairly complex; they will provide a good measure of the performance of the plug-in. We expect that the tool will be able to finish the automatic verification of these models in reasonable time.

Grade: [4]

3 *What is the tool's response time to a change by a user to the specification? (U)*

A change to the specification by the user requires executing the tool from the beginning.

Grade: [4]

4 *What is the cost of the hardware and operating system required to run the tool at an acceptable speed? (C)*

A medium range Windows PC is adequate to run the tool at an acceptable speed. The addition of extra memory allows larger specification to be tackled.

Grade: [5]

5 *What is the cost of the tool licence for one, five or twenty users for a year, including support? (C)*

The released version of the tool will be free.

Grade: [5]

Development of the tool

1 *How quickly can an identified bug be fixed in code? (U, X, O)*

The code of the two translators, which were developed within the project, is relatively small and well documented; consequently, it is quite easy to identify and fix bugs. The main engine has been stable now for some years; changes to it would require more significant effort. On the other hand, the developer of the main engine participates in RODIN and commits some of his time for bug fixes.

Grade: [5]

2 *How quickly can an identified bug be fixed in the development and testing system? (U, X, O)*

The tool uses the standard Eclipse testing facilities, e.g. JUnit testing. Adding new tests is a fairly straightforward process and requires relatively small time and effort.

Grade: [3]

3 *How quickly can minor and major features be implemented? (X)*

Minor features are easy to implement and do not require substantial code rewriting. This is due to the direct access to the combined action and state information, which is easily accessibly in the Petri net representation.

Grade: [3]

4 *How long is the time required to bring in and educate a typical tool developer? (U, X)*

We expect that this will be relatively short for someone familiar with the basic concepts of process algebras and state machines. Furthermore the translators are relatively small and well documented; however, we do not yet have experimental data to support this.

Grade: [N/A]

5 *How many operating systems and architectures are supported and how many are possible? (U)*

The release version will support the Windows and Unix versions of the RODIN platform.

Grade: [4]

Testing and verification

1 *How extensive are unit, functional and system testing of the tool? (U, S)*

Standard JUnit tests will be used for several parts of the tool. These tests will be performed in every release version of the tool and will provide sufficient evidence that the tool is working according to the specification.

Grade: [3]

2 *How easy is it to compare two versions of the tool? (U, S)*

The different release versions of the tool will be distributed with a release history, which documents all changes. Furthermore, if there are no changes in the input language of the tool between different versions, comparing their relative performance is straightforward.

Grade: [3]

3 *How well does the tool admit self-analysis? (U, S)*

The tool has not been designed for this purpose.

Grade: [N/A]

4 *How reliable are error tracking and regression testing? (U, S)*

Error tracking is reliable due to the standard algorithms used for the derivation of counterexamples from the unfolding structures.

Grade: [4]

5 *What is the self-fault detection history, and in particular how many faults does the trend predict in the current tool? (S)*

There are no data available to form a conclusive predictive answer at the moment.

Grade: [0]

6 *What classes of self-fault are detectable or not detectable in the tool? (S)*

It is possible to detect faults in the construction of a Petri net from a given mobile agent systems model, such as isolated places and duplicate arcs. Furthermore, in every release version we will run a standard set of benchmarks to detect differences in performance.

Grade: [3]

Source language

1 *What fraction of the possible source language grammar does the tool accept, analyse, and analyse correctly? (U, S, X)*

The tool is suitable for the task of model checking of mobile agent systems modelled in the newly developed programming notation. This programming notation is a hybrid of Event-B combined with constructs inspired by two process

algebras: KLAIM and pi-calculus. The process algebra constructs are fully supported and we plan to support a substantial part of the Event-B notation by the end of the project.

Grade: [4]

2 *If the source language is based on an external definition e.g. an ISO standard, how much must it change to be acceptable to the tool? (U)*

There is no ISO standard. The tool requires the newly developed programming notation for mobile agent systems as input. No changes to this notation are necessary.

Grade: [4]

3 *What is the earliest point in system development when a source document may be analysed? (C)*

Model checking is a technique that is particularly effective in detecting concurrency related defects and can be applied at an early stage of system design.

Grade: [5]

Output form

1 *How easy is it to auto-parse the tool output? (U, X)*

The output of the tool is in plain text format and it is easy to parse.

Grade: [4]

2 *What level of control over the output volume and content is allowed? (U, X)*

The user has a substantial level of control over the output volume of the tool. The tool can output information about the statistics of the developed models (e.g. size, time to unfold the model, etc.) together with the results of the automatic verification process and a usable trace in case of the discovery of an error in the specification.

Grade: [4]

3 *How well does the output relate to the input, for instance for error reporting? (C, U)*

Part of model checking is the generation of traces leading to error situations, which can then be simulated or visualised with the help of the included animator.

Grade: [5]

7.1.7 Conclusion

During the third year of the project, we managed to continue our work according to the objectives and milestones listed in the RODIN DoW[1]. Based on the experience gained in year two of the project, by building a model checker for finite pi-calculus specifications, we proceeded in two steps. Initially, we developed a hybrid specification language for mobile agent systems together with the theoretical translation of this language into Petri nets. Following this we developed an efficient model checker for the automatic verification of mobile agent systems and we integrated this tool to the RODIN platform. The final public version of the plug-in, together with its full supporting documentation, are scheduled to be released according to plan in September 2007.

7.2 ProB model checking and animation

7.2.1 Introduction

ProB is an animator and model checker for the B-method. This plug-in integrates ProB within RODIN and applies it to Event-B specifications. It can be used for animation purposes, as well as for disproving individual proof obligations.

Writing a formal specification for real-life, industrial problems is difficult and error prone, even for formal methods experts. Whilst specifying a formal model for later refinement and implementation it is crucial to get approval and feedback from domain experts to avoid the costs of changing a specification late in the development. ProB's animation features can help to demonstrate what a specification actually does. Domain experts can explore the B model and check whether a B specification corresponds to their expectations.

The disprover plug-in for RODIN uses the ProB animator and model checker to find automatically counterexamples for a given problematic proof obligation. When the disprover finds a counterexample, the user can directly investigate the source of the problem (as pinpointed by the counterexample) and should not attempt to prove the proof obligation. In some circumstances the plug-in can be used as a prover, i.e., in that case the absence of a counterexample actually is a proof of the proof obligation.

7.2.2 Current Status

Initially the ProB model checker was ported to the Eclipse platform, and extended with domain specific Flash animation features. An extensible editing platform was also developed, and combined with the Animator. This provides an integrated development and testing tool for classical B models. This work was reported at the B2007 conference in Besançon [75, 76].

During a second phase, the ProB model checker has been adapted for Event-B. The animator has been fully integrated into the RODIN platform and can be installed from http://www.stups.uni-duesseldorf.de/ProB/update/prototype/. It supports animation of most Event-B constructs and has been used successfully by various RODIN partners. Moreover, another plug-in has been developed which enables a user to apply ProB during proof development. In that case the tool can be used to find potential counterexamples to the proof obligations. This work was reported at the AFADL 2007 conference in Namur [77].

7.2.3 Progress since Year 2 Assessment

The ProB animation plug-in has been:

- ported from a stand-alone Eclipse version of ProB to an integrated RODIN plug-in;

- enabled to animate Event-B specifications;
- made more robust;
- extended to support further Event-B constructs.

The disprover plug-in has been developed and integrated within the proving interface.

7.2.4 Integration with the Platform

The latest ProB Plug-in uses both the RODIN database and its building mechanism. This improved the robustness of the implementation compared to the previous version of ProB for Eclipse. During the development of the ProB plug-in we also discovered and fixed a performance problem within the RODIN platform.

7.2.5 Use by the Case Studies

ProB has been used by all case studies during the verification and validation lifecycle stage. In all cases feedback has been positive, indicating that ProB adds significant value to the process.

7.2.6 Plug-in Metrics

Requirements and Functionality

1 How well does the tool perform its stated purpose? (C)

If a tool fails to do what it claims, is inconsistent or erratic in its operation or takes too long to do it, it will not be commercially attractive and it will be hard to assure.

The ProB tool can animate large B formal models. It can also be used for automated consistency and refinement checking. The tool has proved useful on various industrial case studies (some of which are within RODIN, such as case study 1, Lyra protocol engineering, and case study 3, Formal Techniques within an MDA Context).

For example, ProB can animate all 13 refinements of Abrial's PRESS B case study. The final model contained "about 20 sensors, 3 actuators, 5 clocks, 7 buttons, 3 operating devices, 5 operating modes, 7 emergency situations, etc" [78]. The 13th refinement has 163 operations.

A number of RODIN partners have successfully applied the Event-B version of ProB to various specifications (e.g. case study 3 used ProB to animate various hardware models, such as a Huffman encoder/decoder).

The disprover plug-in consists of a user interface (UI) that displays the results of a proof and a core component that encapsulates the proof logic. The UI is an extension to the RODIN proving user interface. It allows the user to select a node

in the proof tree, to be checked with ProB. The core plug-in provides a way to apply the ProB disprover. Its role is to:

- Translate the sequence into a B machine.
- Call ProB through the Eclipse ProB core plug-in.
- Return results to the user interface.
- Handle failures, time outs and user cancellation requests.

Grade: [4-5]

2 *How rigorously is the required tool behaviour defined? (O, S, U)*

The use of rigorous mathematical generation enables more rigorous testing, and could support mathematical analysis of the tool's operation.

The tool animates B models according to the standard mathematical B semantics, albeit for given fixed sizes of the basic sets. The relationship to classical B consistency checking and refinement checking is clearly described in various papers about the tool.

Grade: [3-4]

3 *How much does it contribute to system correctness? (C, S)*

Due to the acquisition and user training costs, any tool must provide significant gains in system correctness if it is to be to be adopted commercially.

As the various case studies have shown, ProB has very quickly identified a series of errors. As a result of ProB's automated nature, little additional user training is required.

Grade: [5]

4 *How much does it extend/shrink system development and testing phases? (C)*

If a tool increases system correctness then one would normally expect a decrease in the amount of system re-testing after the tool has been applied.

ProB does contain a very preliminary test case generation component; but currently this cannot be applied to realistic models. Development and testing are diminished only to the extent that the various formal models are more likely to be correct and exhibit the desired functionality. The development phase of the formal models itself should be considerably diminished, as the traditional interactive proofing approach is very labour intensive.

Grade: [3]

Tool Usability

1 *How long does it take a developer, who is knowledgeable in the specification language used, to learn how to use the tool effectively? (U)*

Arguably, this is the prime usability criterion. Typically a development team's trial use of a tool will determine its use. If the tool proves too difficult to learn relatively quickly then it is unlikely to be selected.

ProB can be used straightaway be people versed in B. It has also been used to teach B, highlighting the fact that it can help people to understand and master the B method.

Grade: [5]

2 *How long does it take the tool to run to completion on a specification of representative size? (C)*

The slower the tool, the more the developer will be frustrated.

This question is difficult to answer for ProB, due to the exponential state explosion problem ("how long is a string?"). However, anecdotal evidence indicates that in the early development phases errors are more prevalent and the tool runs much more quickly (as it quickly finds an error). Later in the development, errors become increasingly difficult to locate and the tool takes longer to run. Due to the exponential state explosion problem, an exhaustive check may not be feasible.

Grade: [3]

3 *What is the tool's response time to a change by a user to the specification? (U)*

Typically specifications are created incrementally and usually, in an industrial development, are adjusted throughout the project as a result of requirements change. A usable formal tool must be able to accept such changes without undue overhead in re-analysis and rework.

Using ProB, a new specification can be very quickly reloaded. In its Eclipse version the parser re-runs automatically in the background.

Grade: [4]

4 *What is the cost of the hardware and operating system required to run the tool at an acceptable speed? (C)*

The acceptable tool speed may vary from project to project, but a tool that runs on a standard PC has a clear commercial advantage over one that requires a high-

specification server to run. If a CPU-intensive tool can farm out work over a network of PCs then this may make its use more practical.

A standard PC is adequate for animating all the RODIN case study B models.

Grade: [4]

5 *What is the cost of the tool licence for one, five or twenty users for a year, including support? (C)*

Cost is rarely the primary consideration in tool selection, but it can be significant. If the project can only afford a single licence but has twenty developers then the tool can become a bottleneck and become less usable overall.

Academic licences are free; the tool is free to RODIN project participants. A commercial licensing plan has not yet been investigated.

Grade: [4]

Development and Integrity of the Tool

These results, at the end of year three, measure the usability and extensibility of the tools for external developers.

1 *How quickly can an identified tool bug be fixed in tool's code? (U, X, O)*

This is a combined measure of the precision of the tool's output (defining the bug), accessibility of the tool's source code (finding the bug) the re-analysis (fixing the bug) and re-test (checking the fix) speed. We have also considered whether multiple fixes can be applied concurrently.

Most bugs have been fixed within a few days, usually less.

Grade: [3-4]

2 *How quickly can an identified tool bug be fixed in the development and testing system? (U, X, O)*

This measures the ease of adding tests to the testing system, verifying the test results and maintaining the errors database.

The Prolog source code has a custom framework to support unit and regression testing. It is easy to add new unit tests. New regression tests have to be added to a Tcl script that runs the tool on sample specifications and checks whether stored traces can be reproduced. It is also easy to add a new regression test. The Eclipse version of the tool also has extensive tests.

Grade: [3-4]

3 *How quickly can minor and major new tool features be implemented? (X)*

This metric relates to the tool's overall design; it is hard to define a good design, but this is one significant measure. If even minor features require substantial tool rewriting then the tool is inflexible. If major features can be added in a relatively straightforward manner then this indicates very well designed code.

During year two, several features were added to the tool: integration with CSP, automated refinement checking, Flash-based animation engine, symmetry reduction, support for records and other new B features. This indicates that new features can be implemented relatively quickly.

The new Eclipse version of ProB is designed to be more easily extendable to outside developers: we have provided extension points, which we have used to extend ProB.

Grade: [3]

4 *How long is the time required to bring in and educate a typical tool developer? (U, X)*

This not only measures the accessibility of the tool's design but also the chosen implementation language and the surrounding toolset for configuration management.

Work at the core of ProB requires deep knowledge about Prolog, co-routineing and constraint solving. At present a single person (Michael Leuschel) is primarily developing and maintaining the core, even though a new PhD student was able to add a new data type (free types for Z) after about six months. Work on other components is more accessible, and more than a dozen different people have implemented various extensions. The new Eclipse version makes developing extensions more straightforward (for developers familiar with Java and Eclipse).

Grade: [3]

5 *How many operating systems and architectures are supported and how many are possible? (U)*

It is relatively easy to implement a tool on a single operating system and architecture. Adding a second, very different operating system is typically much more difficult. The classic pair of operating systems to try is Windows vs. UNIX. Getting the tool to run the same way (and demonstrably so) on multiple platforms indicates the platform independence of the code, and indicates good potential longevity of the tool.

We provide precompiled binaries for Linux, Mac OS X, and Windows. Solaris binaries could also be generated (but there has been no demand). A platform-independent version is also available; however this requires a valid SICStus licence.

Grade: [4-5]

6 *How extensive are unit, functional and system testing of the tool? (U, S)*

There remains no substitution for testing. Opinions on the efficacy of unit testing vary, but functional and system tests of the tool are necessary to demonstrate what the tool currently does and does not do.

The Prolog source code has a custom developed unit and regression testing framework. The Prolog tool currently has more than 760 unit tests, which can also be run by the user.

For regression testing there is a Tcl script that runs the tool on sample specifications and checks whether stored traces can be reproduced and whether invariant violations can be found (for example there are various machines which encode hundreds of theorems about set theory, relations, functions, sequences and checks that the tool does not find counter examples to those theorems).

The Eclipse and Java components also have a large number of unit tests associated with them.

Grade: [4]

7 *How easy is it to compare two versions of the tool? (U, S)*

Associated with testing is the ability to compare test results; it is much easier to see what has changed from a known baseline (e.g. the previous release of the tool) and justify the changes rather than justify an entire set of test results. An easy and mostly automated comparison can prove invaluable.

An extensive release history is distributed with the tool, explaining changes between the various versions.

Grade: [3]

8 *How well does the tool admit self-analysis? (U, S)*

Not all tools can do this; the SPARK Examiner, Perfect Developer and ProofPower are examples of tools that can admit self-analysis, whereas FDR cannot – its analysis pertains to parallel systems, not to conventional single-thread programs. Where a tool could reasonably admit self-analysis, it is good to perform it. An alternative view of this issue is to ask the question: "Can the tool be produced using the facilities of the tool itself?"

At present, the tool is not intended for developing code; its purpose is to analyse formal models. Building a formal tool of parts of the tool is not unreasonable; however modelling the core constraint-solving engine of ProB in B probably lies outside the of B's current capabilities.

However, the tool uses its model checking capabilities to check for errors (see regression testing above); several errors were caught (in earlier versions) this way when the tool was able to disprove theorems from set theory.

Grade: [2-3]

9 *How reliable is error tracking and regression testing? (U, S)*

It is not enough to track errors; for a high-assurance tool it must be very difficult for identified errors to slip through the cracks. Applying a hazard analysis to the error tracking system may yield useful information about where such slips may occur and how to prevent them.

No hazard analysis has been performed to date.

Grade: [0]

10 *What is the self-fault detection history, and in particular how many faults does the trend predict in the current tool? (S)*

Tracking the faults identified in the tool will give some idea of whether the faults are limited in number and apparently decreasing (indicating a maturing tool), detected at a near-constant rate (indicating ineffective error fixing) or increasing (indicating a fragile tool design).

No systematic statistics are kept on this matter. However, all emails from users are kept. A log of open issues is maintained.

Grade: [2]

11 What classes of self-fault are detectable or not detectable in the tool? (S)

A combination of the testing and operational environment of the tool will determine what faults in the tool are likely to be detected. For instance, a large regression test suite with automated difference detection is likely to pick up non-deterministic behaviour. The tool developers should identify what self-fault classes cannot be detected, and work to introduce detection or prevention measures.

The tool is run on a standard benchmark suite to detect deterioration (or improvements) in performance.

Grade: [3]

Input (source) Language Issues

1 What fraction of the possible source language grammar does the tool accept, analyse, and analyse correctly? (U, S, X)

Very few tools accept all of a complex source language. However, it is reasonable to expect a very large fraction of the source language to be accepted and the unacceptable constructs to be identified clearly.

ProB accepts 95% of the Event-B language.

Grade: [4]

2 If the source language is based on an external definition e.g. an ISO standard, how much must it change to be acceptable to the tool? (U)

It is plausible that there will be a large set of programs or specifications written in the ISO language. The fewer specification changes necessary to make them acceptable to the tool, the better.

There is no ISO standard. We strive to be compatible with B4Free/AtelierB.

Grade: [3]

3 *What is the earliest point in system development when a source document may be analysed? (C)*

This is an aspect of effectiveness in the system development cycle. It is well recognised that the earlier specification errors can be detected, the cheaper they are to fix.

The new Eclipse version of ProB analyses the specification after 10 seconds of inactivity; As a consequence it is able to analyse source documents as early as possible.

Grade: [5]

Output Format

1 *How easy is it to auto-parse the tool output? (U, X)*

This is often important when the tool is incorporated into an existing program development system, and its output will affect subsequent development operations. This task becomes harder if the tool's output is difficult to parse. The ideal is for the output to be in a suitable standard format e.g. XML, as this allows the use of off-the-shelf parsers and conversion tools.

Where the tool maintains data in a persistent state (such as a database), "auto parsing" should be interpreted as "accessing the data via a standard querying mechanism, such as SQL or ODBC".

Output is in ASCII format (or Prolog format when storing the state space). ProB also provides .dot output as well as Postscript/PDF views of the state space.

Grade: [3-4]

2 *What level of control over the output volume and content is allowed? (U, X)*

Controllable output will broaden the possible users of the tool. Some developers will simply want to know if the analysis is complete and correct; others will want a great deal of data about an identified fault, and it is important that both requirements can be satisfied. The result of a formal proof, for instance, may vary from a binary "proved/not proved" and a full step-by-step proof log detailing each application of a theorem or deduction rule. A caveat is that it should be very difficult to hide actual failures in the output.

A command-line version is available for scripting.

The .dot/PS/PDF output can be influenced by user preferences.

Grade: [4]

3 *How well does the output relate to the input, for instance for error reporting? (C, U)*

This is important for day-to-day operation of the tool. If the tool finds a fault, it should make it easy for the developer to establish the location of the cause of the fault (rather than just the points at which symptoms of the fault are visible).

Syntax errors are highlighted in the code. The new Eclipse version uses the Eclipse mechanism for locating errors in the source code (for syntax errors as well as a few semantic errors). Model and refinement checks provide the user with traces that exhibit the erroneous behaviour. Traces can be saved to an ASCII file.

Grade: [4]

7.2.7 Conclusion

Animation has proven extremely useful when developing B models. Our plug-ins provide support for animating Event-B models, and are fully integrated into the RODIN development environment. The plug-ins have proven popular with other RODIN partners, and have been used to uncover a considerable number of errors in formal models.

7.3 Brama

7.3.1 Introduction

Brama animates Event-B models, with objectives to:

- enable multi-refinement level model to be debugged, thus providing confidence that the model behaves as expected;
- show a B model such that it can be understood by a non B specialist, thus enabling third party validation.

A dedicated website has been created at http://www.brama.fr/index_en.html.

Since D16 [13], Brama has been:

- extended with improvements to:
  - support of Event B expressions and predicates,
  - animation capabilities,
  - man-machine interface improvement
- documented, and
- experimented with in Case Study 2 (c.f. §5.2 above) and other industrial projects.

7.3.2 Current Status

During the three years of the project, the following goals have been achieved:

- Development of an Event-B animation tool.
- Provision of scheduling features, enabling automatic, delayed execution of events.
- Improvement of underlying evaluation algorithms.
- Proper integration with the RODIN platform as a platform plug-in.

7.3.3 Progress since Year 2 Assessment

The Brama plug-in has been:

- Documented.
- Improved, with a better man-machine interface and better error messages.
- Applied to various Event-B scenarios,
- Integrated with the Composys plug-in.

7.3.4 Integration with the Platform

Brama is integrated into the RODIN the platform as a plug-in. It provides several views (events, variables, constants, model structure and execution history). It also contributes the RODIN model editor (constant value editor).

7.3.5 Use by the Case Studies

Case Study 2 – Engine Failure management has used the Brama plug-in (c.f. §5.2 above).

7.3.6 Plug-in Metrics

Requirements and functionality

1 How well does the tool perform its stated purpose? (C)

Brama was designed to assist verification of model correctness and compliance with modelled systems. A model can be examined, by selecting a particular sequence of events or by letting the tool choosing a sequence. Scenarios can be saved, resumed, and replayed. The tool has been used by various students/engineers.

Grade: [5]

2 How rigorously is the required tool behaviour defined? (O, S, U)

The tool behaviour has been specified using natural language.

Grade: [3]

3 How much does it contribute to system correctness? (C, S)

The tool assists:

- model behaviour (scenarios, etc) verification, and
- detection of broken invariants.

Grade: [4]

4 How much does it extend/shrink system development and testing phases? (C)

The tool is applicable at any stage of the development. It has been used at tender writing stage, providing an initial reference for the complete development.

Grade: [3]

Tool usability

1 *How long does it take a developer, who is knowledgeable in the specification language used, to learn how to use the tool effectively? (U)*

Animating a RODIN model takes minutes and is interface-based. Graphical animation requires some Flash knowledge and FlashMX development experience.

Grade: [3]

2 *How long does it take the tool to run to completion on a specification of representative size? (C)*

An animation step executes in under a second. Several animations are available on the web, demonstrating this fact. When the animation engine is unable to find suitable values for variables in a non-deterministic substitution, the animation slows down for seconds.

Grade: [4]

3 *What is the tool's response time to a change by a user to the specification? (U)*

A modification is directly reflected in the animated model. However, the animation has to be restarted after such modifications.

Grade: [4]

4 *What is the cost of the hardware and operating system required to run the tool at an acceptable speed? (C)*

Any PC is able to run the tool.

Grade: [5]

5 *What is the cost of the tool licence for one, five or twenty users for a year, including support? (C)*

The released version of the tool is free.

Grade: [5]

Development of the tool

1 *How quickly can an identified bug be fixed in code? (U, X, O)*

The tool is based on the Java Eclipse framework, heavily using JTesting. Testbenches are executed on Windows, Linux and MacOsX platforms.

Grade: [4]

2 *How quickly can an identified bug be fixed in the development and testing system? (U, X, O)*

Adding new tests is fairly straightforward and requires relatively little time and effort.

Grade: [3]

3 *How quickly can minor and major features be implemented? (X)*

Minor features are easy to implement. Major features are either implemented as plug-in extensions or by modifying object-oriented code.

Grade: [3]

4 *How long is the time required to bring in and educate a typical tool developer? (U, X)*

Educated/non educated tool developers have developed several extensions. The available development documentation has enabled developers to modify efficiently the tool in one week.

Grade: [3]

5 *How many operating systems and architectures are supported and how many are possible? (U)*

The release version supports the Windows, Linux and MacOsX versions of the RODIN platform.

Grade: [4]

Testing and verification

1 *How extensive are unit, functional and system testing of the tool? (U, S)*

The tool comes with an extensive set of test cases. The animation engine has also been used in an industry tool, for verifying railway safety critical data (track topology).

Grade: [3]

2 *How easy is it to compare two versions of the tool? (U, S)*

The different release versions of the tool will be distributed with a release history, which documents all changes.

Grade: [3]

3 *How well does the tool admit self-analysis? (U, S)*

The tool has not been designed for this purpose.

Grade: [N/A]

4 *How reliable are error tracking and regression testing? (U, S)*

Error tracking is reliable due to the Brama data structure.

Grade: [3]

5 *What is the self-fault detection history, and in particular how many faults does the trend predict in the current tool? (S)*

Data is not available to provide a conclusive predictive answer at present.

Grade: [0]

6 *What classes of self-fault are detectable or not detectable in the tool? (S)*

Evaluation errors are detectable if an invariant is broken.

Grade: [3]

Source language

1 *What fraction of the possible source language grammar does the tool accept, analyse, and analyse correctly? (U, S, X)*

The tool supports completely the RODIN platform language.

Grade: [4]

2 *If the source language is based on an external definition e.g. an ISO standard, how much must it change to be acceptable to the tool? (U)*

There is no ISO standard.

Grade: [N/A]

3 *What is the earliest point in system development when a source document may be analysed? (C)*

The tool should be applied at the very beginning of a development.

Grade: [5]

Output form

1 *How easy is it to auto-parse the tool output? (U, X)*

The output of the tool is displayed directly in the RODIN platform interface (animation form).

Grade: [5]

2 *What level of control over the output volume and content is allowed? (U, X)*

The output volume is directly related to the input model.

Grade: [5]

3 *How well does the output relate to the input, for instance for error reporting? (C, U)*

If the model doesn't behave as expected, scenarios can be saved to help determine where the error occurred.

Grade: [5]

7.3.7 Conclusion

As expected, as well as presenting the results of modelling, formal model animation enables non-experts to discover modelling errors. The Brama plug-in has been developed in close collaboration with final users (within and outside the project), and experimented on real, industrial size projects. The regular and accurate feedback collected demonstrates a real involvement and interest from modelling practitioners and animation recipients.

7.4 UML–B

7.4.1 Introduction

UML–B is an extension feature for the RODIN platform. It provides a UML-like graphical front-end for Event-B modelling and adds class-oriented and state machine modelling capabilities. UML–B is automatically translated into Event-B for analysis and verification.

7.4.2 Current Status

UML–B has been used in several of the case studies and extensively on case study 2 (Engine Failure management) (see section 5.2). Several papers [28,29,30,79] have been published concerning its use in case study 2. UML–B has been demonstrated at both RODIN industry days and at the Memot workshop, Oxford, July 2007. Two formal experiments have been performed to compare users' comprehension of UML–B verses B/Event-B models. The first experiment was performed on the pre-RODIN version of UML–B and compared against classic B models. The second, more recent, experiment compares the latest version of UML–B with Event-B.

7.4.3 Progress since Year 2 Assessment

UML–B was not assessed at year two because it was not sufficiently developed to support a robust evaluation. The year one assessment examined the pre-RODIN version of UML–B. The assessment highlighted some of the problems with the old version and these have been taken into account when developing the new version. The new version of UML–B is a UML-like modelling notation rather than a specialisation of UML. This major decision was taken after trying to implement the new version as a 'stronger' specialisation of UML 2.0 via its improved profile concepts. Although this was partially successful, we still felt that increased freedom to design the appropriate modelling concepts was needed, and that these concepts needed to be brought to the fore (rather than attaching them as additional features via stereotyping). Therefore UML–B was redesigned via an independent meta-model.

- The meta-model was used to generate a model repository using the meta-model modelling framework (EMF).
- A graphical editor was generated using the graphical modelling framework (GMF).
- A meta-model 'builder' was implemented to convert UML–B models to Event-B. (Eclipse builders are programs that are informed of changes to resources in the current workspace so that those changes can be processed. For example, the Java compiler runs as a meta-model builder).
- Further meta-model plug-ins were added to provide user interface facilities such as a new project wizard, a project nature, and a perspective layout.

- The UML–B plug-ins were packaged as a meta-model 'feature' and released on the RODIN project file release system so that it can be installed from the RODIN project update site.

7.4.4 Integration with the Platform

The UML–B, Event-B builder uses the platform API in order to create an Event-B project and populate it with the corresponding Event-B models. The API was easy to use. The Event-B keyboard was extended to provide additional key combinations. This was difficult to implement because the lexical analyser is not extensible. Consequently, the lexor was copied and adapted. The 'attributes' extension point of the RODIN database was used to add an attribute to Event-B elements to provide a link to the UML–B element from which it was created. Further integration is currently underway in order to retrieve problem markers from the Event-B elements so that the problems can be reflected onto the UML–B source element.

7.4.5 Use by the Case Studies

UML–B has been used on CS2 by teams working at ATEC, UK and Åbo Akademi, Finland. ATEC provided feedback on the pre-RODIN version of UML–B. The main point raised by CS2 in the year one and year two assessment reports was that use of the action and constraint language, uB, relied upon knowing details of the translation process. This was mainly due to the translation of associations as a function to sets of instances. This has been changed to translate to relations so that the action and constraint language is simpler. In addition, the use of 'self' is now mandatory, whereas before it could be omitted. However, further work is required to complete a self-contained notation. In year two CS2 suggested widening UML–B to utilise other UML diagram notations.

More recently the case study has used the new version of UML–B. This has been successful but has revealed a need for dummy 'refined classes' in order to handle refinement of class features. This feature has now been added to UML–B.

Similarly, CS4 required the use of 'records', which were refined by context extension. Records are analogous to ClassTypes in UML–B (ClassTypes are Classes that have only constant attributes and no variable data or methods). The ability to 'extend' ClassTypes in extended (refined) contexts was added to UML–B as a result of this case study.

CS1 used a bespoke profile of UML called UML-Lyra. As a consequence, Åbo Akademi has investigated a model transformation from UML-Lyra to UML–B.

7.4.6 Plug-in Metrics

Requirements and Functionality

1 How well does the tool perform its stated purpose? (C)

The new version of UML–B works well and is easier to understand than the previous version.

Some housekeeping requires further attention. For example, data can be lost if several diagrams are edited at the same time.

There are several areas that are not yet implemented. For example, better mark-up of errors on the diagrams is required.

Grade: [4]

2 How rigorously is the required tool behaviour defined? (O, S, U)

The abstract syntax for UML–B is defined by a meta-model written in UML. OCL constraints are included to describe well-formedness constraints. The translation into Event-B is defined in natural language and by tables.

Grade: [3]

3 How much does it contribute to system correctness? (C, S)

The tool translates UML models to B where other tools can be used to validate and verify the models formally. The use of a well-known diagrammatic specification notation makes the models accessible to domain experts who can validate them. Thus the contribution is to facilitate the benefits of both UML and B notations.

Grade: [5]

4 How much does it extend/shrink system development and testing phases? (C)

Compared to developing in B, the diagram entities represent significant amounts of B notation making it much quicker to create models and to re-factor them when problems are discovered. Compared to modelling in UML, the usual benefits of formal development (i.e. early detection of errors) are gained.

Grade: [4]

Tool usability

1 *How long does it take a developer, who is knowledgeable in the specification language used, to learn how to use the tool effectively? (U)*

The translation tool runs automatically and requires no user action. Assuming knowledge of the specification language, UML–B, and familiarity with similar UML tools, the drawing tool is easy to use. Students were able to use the tool after a one-hour tutorial.

Grade: [5]

2 *How long does it take the tool to run to completion on a specification of representative size? (C)*

The typical time for tool completion is less than one second. The largest models may take a few seconds.

Grade: [5]

3 *What is the tool's response time to a change by a user to the specification? (U)*

UML–B decreases the rework overhead compared to making changes in B because it is so much quicker to re-work the models in a diagrammatic modelling tool.

Grade: [4]

4 *What is the cost of the hardware and operating system required to run the tool at an acceptable speed? (C)*

Any desktop PC, which is capable of running Eclipse at an acceptable speed, can support the plug-in. A minimum specification of 1Ghz processor with 1GB of RAM is suggested but an older PC with approximately half this spec will run UML–B tolerably.

Grade: [5]

5 *What is the cost of the tool licence for one, five or twenty users for a year, including support? (C)*

UML–B (and all the software it requires) is free of charge. Support is provided, as needed, free of charge.

Grade: [5]

Development of the Tool

These metrics address **tactical** and **technical** views of the **product** and **resource** objects.

1 *How quickly can an identified bug be fixed in code? (U, X, O)*

This depends on the nature of the bug. Simple translation bugs can be fixed very quickly (within a few hours).

However, it is often difficult to determine the cause of the problem when bugs are discovered in the drawing tool. Such bugs may take weeks to fix.

Bugs are usually addressed in batches before a re-release is made onto the SourceForge FRS system.

Grade: [2]

2 *How quickly can an identified bug be fixed in the development and testing system? (U, X, O)*

The bug tracking system of SourceForge is used. There is no testing system.

Grade: [N/A]

3 *How quickly can minor and major features be implemented? (X)*

UML–B is well structured and is modularized into thirteen plug-ins, which have low coupling. The Eclipse extension point mechanisms are used to isolate aspects of the tool with a view to separating concerns. Existing Eclipse tool building projects are used (e.g. EMF, GEF, GMF) to provide powerful programming features. Hence new requirements can usually be added quickly. Some requirements however, may be constrained by these features if they do not align with the vision of the feature builders.

Grade: [3]

4 *How long is the time required to bring in and educate a typical tool developer? (U, X)*

An experienced Eclipse Java developer could easily start maintenance almost immediately. An inexperienced developer (with no meta-model knowledge) has successfully made changes to the translator tool after a day or two. Some parts of the system (e.g. those requiring knowledge of GMF) would take longer to learn.

Grade: [3]

5 *How many operating systems and architectures are supported and how many are possible? (U)*

UML–B runs within Eclipse and therefore runs on any operating system supported by Eclipse. It has been demonstrated to run on the Windows, Linux and MAC-OSX versions of Eclipse.

Grade: [5]

Testing and Verification

The following metrics address **tactical** and **technical** views of **product** objects.

1 *How extensive are unit, functional and system testing of the tool? (U, S)*

Currently, testing is purely on an ad-hoc basis when features are added or corrections are made. Little regression testing is performed.

Grade: [0]

2 *How easy is it to compare two versions of the tool? (U, S)*

The SourceForge CVS facilities are used to develop the code. Eclipse acts as a client to the CVS repository and provides excellent version comparison facilities.

Grade: [5]

3 *How well does the tool admit self-analysis? (U, S)*

Self-analysis is not applicable since the tool is a translator not an analyser.

Grade: [N/A]

4 *How reliable is error tracking and regression testing? (U, S)*

Errors are being tracked using the tracking system on SourceForge (initially a system at Southampton was used, so many bugs are on UGForge). The error tracking system is very reliable. Little regression testing is performed.

Grade: [1]

5 *What is the self-fault detection history, and in particular how many faults does the trend predict in the current tool? (S)*

Bugs are being raised regularly due to the immaturity of the tools.

Grade: [0]

6 *What classes of self-fault are detectable or not detectable in the tool? (S)*

Translation faults resulting in inconsistent B are detected when the output is verified using B tools. A translation fault that resulted in consistent but incorrect B might be detected if animation is performed but could go undetected.

Grade: [3]

Source language

These metrics address **tactical** and **technical** views of **process** objects.

1 *What fraction of the possible source language grammar does the tool accept, analyse, and analyse correctly? (U, S, X)*

All of the source language (UML–B) is accepted.

Grade: [5]

2 *If the source language is based on an external definition e.g. an ISO standard, how much must it change to be acceptable to the tool? (U)*

The source language has been designed as part of the tool and is not based on a standard.

Grade: [N/A]

3 *What is the earliest point in system development when a source document may be analysed? (C)*

UML–B may be used on an arbitrarily abstract package. In this respect, it draws on the underlying Event-B methods, which are designed to identify specification errors as soon as they are introduced.

Grade: [5]

Output form

These metrics address a **technical** view of the **product** object.

1 *How easy is it to auto-parse the tool output? (U, X)*

The output is in Event-B, which is automatically parsed upon creation/update.

Grade: [5]

2 *What level of control over the output volume and content is allowed? (U, X)*

UML–B is a translation tool, not an analysis tool.

Grade: [N/A]

3 *How well does the output relate to the input, for instance for error reporting? (C, U)*

We can re-interpret this question to consider how well errors reported by the Event-B analysis tools can be related back to the UML–B models. Currently there is no automatic feedback mechanism. The translation inserts comments into the Event-B model to assist the user in tracing the UML–B source of Event-B elements. This has been sufficient for the current case study work. We plan to add automatic error mark-up of the UML–B diagrams as a future development.

Grade: [3]

7.4.7 Plug-in Additional Metrics

A series of empirical evaluations, including two formal experiments and two surveys, were performed to assess the usability of UML–B. Usability in this context means the understandability/comprehensibility, learnability, operability and attractiveness of the method [80]. The experiments evaluated the comprehensibility of the UML–B model, while the surveys assessed the usability of UML–B as a whole.

The first experiment compared the comprehensibility of pre-RODIN UML–B with that of classical B. This was used as a baseline and to assess the basic concepts of UML–B. The subjects' comprehension was measured based on the interpretation of the symbols used, the tracing of input and output, the mapping between models and problem domains, and the model modification. The experiment confirmed that subjects found it easier and quicker to understand models in UML–B than in classical B. In particular, the results suggest, with 95% confidence, that a UML–B model could be up to 16% (*overall comprehension*) and 50% (*comprehension for modification task*) easier to understand than the corresponding B model [81].

Following the first experiment, a theoretical underpinning was developed to explain why graphical representation together with textual representation should assist model comprehension. Stronger mechanisms for measuring the degree of comprehension based on the Cognitive Theory of Multimedia Learning [82] were invented. The second experiment [87] utilised these mechanisms to assess subjects' abilities to construct problem domain knowledge. The idea was that a UML–B model is comprehensible if it allows subjects to not only recognise the presented information but also to extend the understanding of the presented information in novel situations such as problem solving. The construction of knowledge structures was measured by subjects' ability to explain cause-and-effect, compare and contrast two elements, describe main ideas and supporting details, list a set of items and analyse a domain into sets and subsets [83]. These criteria were used together with Bloom's Taxonomy [84] as the measurement instrument. The second experiment compared a (RODIN) UML–B model versus an Event-B model. The results of the experiment suggest, with 95% confidence, that a UML–B model could be up to 32% (*overall understanding*) and 76% (*understanding for modification task*) better and quicker than an Event-B model in promoting problem domain understanding. A UML–B model allows the subjects to analyse, generalise and criticise the presented problem domain quickly and use the understanding to provide solutions.

The surveys based on the grounded theory [85] were performed to understand the nature of experience of using UML–B [86]. The first survey investigated the pre-RODIN UML–B and the second the RODIN UML–B. The surveys found that the method appeals to users who opt into B modelling but prefer working with UML's development style. UML–B's graphical representation alleviates the difficulty of developing a formal model from scratch by stimulating the formulation of ideas through the use of visual objects at the abstraction level. However, the method requires users to understand the principles and roles of both UML and B notations as well as the integration rules. Users also require strong support from the environment. Supporting tools and comprehensive documentation need to be available, useful, easy-to-learn and easy-to-use.

7.4.8 Conclusion

UML–B is a usable notation and supporting toolset. Improvements to the notation, based upon a bespoke meta-model rather than a UML extension, have provided greater flexibility and specialisation. This has made the notation easier and more natural to use compared to the previous U2B3 version.

The new tools are based upon the Eclipse plug-in framework and utilise Eclipse feature projects. This has improved the structure and maintainability of the toolset.

Since the U2B translator tools (and RODIN Event-B tools) run automatically as Eclipse builders, integration between the tools is very good. The Eclipse problem view reports errors, which are easily traced back to the UML–B source elements. Further improvements are planned to mark up the errors on the UML–B diagram elements.

A series of empirical investigations has revealed that UML–B is significantly quicker to understand and modify than Event-B for novice users.

7.5 B2RODIN

7.5.1 Introduction

The B2RODIN tool enables existing B models to be reused with the RODIN platform. Such models should comply with the Event-B language definition [7]. B2RODIN tool is reachable at the update site http://www.bmethod.com/html/outils_en.html.

Since D16 [13], the B2RODIN plug-in has been improved, documented and closely integrated with Brama and CompoSys (a RODIN plug-in that has been developed alongside the project and experimented with on other industrial projects). B2RODIN has been experimented with during case studies two, four and five (c.f. § 5.2, 5.4 and 5.5).

7.5.2 Current Status

During the three years of the project, the following goals have been achieved:

- Development of a tool able to transform AtelierB models into RODIN models. AtelierB models must be Event-B compliant, i.e. only a restricted class of models is accepted as input (no nested substitutions, condition/action form, etc.).
- Support for Brama. The Brama animator requires valued constants. This valuation can be added to original B models (as comments) and processed by B2RODIN, to have valid RODIN constants.
- Proper integration with the RODIN platform as a platform plug-in.

7.5.3 Progress since Year 2 Assessment

The B2RODIN plug-in has been:

- Integrated with other RODIN plug-ins (Brama and Composys).
- Documented.
- Improved, with a better man-machine interface and better error messages.
- Applied to various B models (case-studies, industrial projects).

7.5.4 Integration with the Platform

B2RODIN is integrated with the platform as a plug-in and provides a valuable wizard.

7.5.5 Use by the Case Studies

B2RODIN has been used by case studies two, four and five (c.f. § 5.2, 5.4 and 5.5).

7.5.6 Plug-in Metrics

Requirements and functionality

1 *How well does the tool perform its stated purpose? (C)*

The tool was designed to help people migrate AtelierB models to the RODIN platform, without having to retype the entire model. Only Event-B compliant models are accepted (not nested substitutions, SELECT/ANY/BEGIN are the only supported substitutions). There is little filtering of the input model, which sometimes leads to failures, but for adequate models, the transformation is performed as expected.

Grade: [4]

2 *How rigorously is the required tool behaviour defined? (O, S, U)*

The tool is reusing the AtelierB B-compiler, which has been heavily tested. B2RODIN has been checked against a large set of models.

Grade: [4]

3 *How much does it contribute to system correctness? (C, S)*

On its own, the tool doesn't contribute to system correctness. The RODIN static checker, and then the RODIN prover, verifies the validity of the transformation.

Grade: [N/A]

4 *How much does it extend/shrink system development and testing phases? (C)*

The tool doesn't contribute, as it only permits to reuse existing models.

Grade: [N/A]

Tool usability

1 *How long does it take a developer, who is knowledgeable in the specification language used, to learn how to use the tool effectively? (U)*

The tool is provided with documentation explaining what sub-language is supported, and with a tutorial. Handling the tool takes minutes.

Grade: [5]

2 *How long does it take the tool to run to completion on a specification of representative size? (C)*

The majority of transformations are completed in less than 10 seconds.

Grade: [5]

3 *What is the tool's response time to a change by a user to the specification? (U)*

The transformation of the initial AtelierB specification is directly taken into account by the tool, provided that the model complies with the Event-B form.

Grade: [5]

4 *What is the cost of the hardware and operating system required to run the tool at an acceptable speed? (C)*

Any PC is able to run the tool.

Grade: [5]

5 *What is the cost of the tool licence for one, five or twenty users for a year, including support? (C)*

The released version of the tool is free.

Grade: [5]

Development of the tool

1 *How quickly can an identified bug be fixed in code? (U, X, O)*

The tool is based on the B compiler library. The code related to the transformation is quite small and well identified. To date no bug has been reported. Problems detected are more related to acceptance and transformations of non-compliant B models.

Grade: [4]

2 *How quickly can an identified bug be fixed in the development and testing system? (U, X, O)*

Adding new tests is fairly straightforward and requires relatively little time and effort.

Grade: [3]

3 *How quickly can minor and major features be implemented? (X)*

Minor features are easy to implement and do not require substantial code rewriting. This is due to the reuse of the existing B compiler library.

Grade: [3]

4 *How long is the time required to bring in and educate a typical tool developer? (U, X)*

The key issue is learning the B compiler structure. We have in-house documentation describing data structures and examples demonstrating how to extend the library.

Grade: [3]

5 *How many operating systems and architectures are supported and how many are possible? (U)*

The release version supports the Windows, Linux and MacOsX versions of the RODIN platform.

Grade: [4]

Testing and verification

1 *How extensive are unit, functional and system testing of the tool? (U, S)*

A model database has been set up. These models have been tested with the tool. They provide sufficient evidence that the tool is working according to the specification.

Grade: [3]

2 *How easy is it to compare two versions of the tool? (U, S)*

The different release versions of the tool will be distributed with a release history, which documents all changes.

Grade: [3]

3 *How well does the tool admit self-analysis? (U, S)*

The tool has not been designed for this purpose.

Grade: [N/A]

4 *How reliable are error tracking and regression testing? (U, S)*

Error tracking is reliable due to B tree data structure (B compiler).

Grade: [3]

5 *What is the self-fault detection history, and in particular how many faults does the trend predict in the current tool? (S)*

Data is not available to provide a conclusive predictive answer at present.

Grade: [0]

6 *What classes of self-fault are detectable or not detectable in the tool? (S)*

Faults are only related to non-compliant B models. In these cases, transformations are not adequate and resulting RODIN models can't be type checked.

Grade: [3]

Source language

1 *What fraction of the possible source language grammar does the tool accept, analyse, and analyse correctly? (U, S, X)*

The tool supports the Event-B structure and a subset of the AtelierB language. The output language is fully compliant with the RODIN platform.

Grade: [4]

2 *If the source language is based on an external definition e.g. an ISO standard, how much must it change to be acceptable to the tool? (U)*

There is no ISO standard.

Grade: [N/A]

3 *What is the earliest point in system development when a source document may be analysed? (C)*

The tool should be applied at the very beginning of a development.

Grade: [5]

Output form

1 *How easy is it to auto-parse the tool output? (U, X)*

The output of the tool is directly provided to the platform.

Grade: [5]

2 *What level of control over the output volume and content is allowed? (U, X)*

The output model is directly related to the input model. They are both comparable in size and complexity.

Grade: [5]

3 *How well does the output relate to the input, for instance for error reporting? (C, U)*

For compliant models, all model elements are in both input and output. Some modifications exist, mainly in the expression language and may be checked manually. The RODIN type checker helps verification by ensuring that the output model complies with the RODIN modelling language.

Grade: [5]

7.5.7 Conclusion

B2Rodin has proved a useful, time saving tool, helping to migrate Event-B modelling from AtelierB to RODIN. The tool is able to translate correctly the B language subset that closely matches the language supported by the RODIN platform, and provides a means to add semantic information to the original model in order to ease its translation.

SECTION 8 REFERENCES

1. Contract for specific targeted research project, Rigorous Open Development Environment for Complex Systems (RODIN). Proposal no. 511599, Annex I *Description of Work*, 27th April 2004.
2. FP6 Project Review, IST 511599 RODIN, P. Gibson, T. Margaria. 20 Oct 2006.
3. Definitions of case studies and evaluation criteria. RODIN deliverable D2/D1.1, Project IST-511599, School of Computing Science, Newcastle University.
4. Traceable Requirements Document for Case Studies, B. Arief, J. Coleman, A. Hall, A. Hilton, A. Iliasov, I. Johnson, C. Jones, L. Laibinis, S. Leppanen, I. Oliver, A. Romanovsky, C. Snook, E. Troubitsyna, J. Ziegler. RODIN deliverable D4/D1.2, Project IST-511599, School of Computing Science, Newcastle University, 2005.
5. Final Decisions. RODIN deliverable D5/D3.1. Project IST-511599, School of Computing Science, Newcastle University.
6. Procedures for Technical Review and Assessment. RODIN deliverable D6/D7.1, Project IST-511599, School of Computing Science, Newcastle University, 25th May 2006.
7. Event-B language, C. Metayer, J-R. Abrial, and L. Voisin. RODIN deliverable D7/D3.2, Project IST-511599, School of Computing Science, Newcastle University.
8. Initial Report on Case Study Development, E. Troubitsyna, Ed. RODIN deliverable D8/D1.3, Project IST-511599, School of Computing Science, Newcastle University, 2005.
9. Preliminary report on methodology. RODIN deliverable D9/D2.1, Project IST-511599, School of Computing Science, Newcastle University, August 2005.
10. Specification of basic tools and platform. RODIN deliverable D10/D3.3, Project IST-511599, School of Computing Science, Newcastle University.
11. RODIN Assessment Report for Year 1. RODIN deliverable D14/D7.2, Project IST-511599, School of Computing Science, Newcastle University, 26th August 2005.
12. Prototypes of basic tools and platform. RODIN deliverable D15/D3.4, Project IST-511599, School of Computing Science, Newcastle University.
13. Prototype plug-in tools. RODIN deliverable D16/D4.2, Project IST-511599, School of Computing Science, Newcastle University.
14. Intermediate Report on Case Study Development, E. Troubitsyna, Ed. RODIN deliverable D18/D1.4, Project IST-511599, School of Computing Science, Newcastle University, 2006.
15. Intermediate Report on Methodology, C.B Jones, Ed. RODIN deliverable D19/D2.2, Project IST-511599, School of Computing Science, Newcastle University, 2006.
16. RODIN Assessment Report for Year 2. RODIN deliverable D22/D7.3, Project IST-511599, School of Computing Science, Newcastle University, 2nd October 2006.
17. Internal versions of basic tools and platform. RODIN deliverable D23/D3.5, Project IST-511599, School of Computing Science, Newcastle University.

18. Final report on Case study development. RODIN deliverable D26/D1.5, Project IST-511599, School of Computing Science, Newcastle University.
19. Case study demonstrators. RODIN deliverable D27/D1.6, Project IST-511599, School of Computing Science, Newcastle University.
20. Report on assessment of tools and methods. RODIN deliverable D28/D1.7, Project IST-511599, School of Computing Science, Newcastle University.
21. Final report on methodology. RODIN deliverable D29/D2.3, Project IST-511599, School of Computing Science, Newcastle University.
22. Public versions basic tools and platform. RODIN deliverable D30/D3.6, Project IST-511599, School of Computing Science, Newcastle University.
23. Synthesis of Scenario Based Test Cases from B Models, M. Satpathy, Q.A. Malik, and J. Lilius. Proceedings of Formal Approaches to Software Testing and Runtime Verification, First Combined International Workshops FATES 2006 and RV 2006, Seattle, USA, August 2006, Lecture Notes in Computer Science, Vol. 4262, pp. 133-147, Springer.
24. Formal Verification of Consistency in Model-Driven Development of Distributed Communicating Systems and Communication Protocols, D. Iliac, E. Troubitsyna, L. Laibinis, and S. Leppänen. Proceedings of IEEE 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2006), pp. 436-455.
25. Formal Model-Driven Development of Communicating Systems, L. Laibinis, E. Troubitsyna, S. Leppänen, J. Lilius, and Q.A. Malik. Proceedings of ICFEM 2005 – 7th International Conference on Formal Engineering Methods, Manchester, November 2005, Lecture Notes in Computer Science, Vol. 3785, Springer.
26. Formal Service-Oriented Development of Fault Tolerant Communicating Systems, L. Laibinis, E. Troubitsyna, S. Leppänen, J. Lilius, and Q.A. Malik. Rigorous Development of Complex Fault-Tolerant Systems, chapter 14, pp. 261-287, Lecture Notes in Computer Science, Springer.
27. Formal Reasoning About Fault Tolerance and Parallelism in Communicating Systems, L. Laibinis, E. Troubitsyna, and S. Leppänen. To appear in proceedings of the Workshop on Methods, Models and Tools for Fault-Tolerance (MeMoT 2007) at the International Conference on Integrated Formal Methods 2007 (IFM 2007).
28. Rigorous development of reusable, domain-specific components, for complex applications, C. Snook, M. Butler, A. Edmunds, and I. Johnson. Proc. 3rd Intl. Workshop on Critical Systems Development with UML, J. Jurgens and R. France, Ed., pages 115–129, Lisbon, 2004.
29. The engineering of generic requirements for failure management, C. Snook, M. Poppleton, and I. Johnson.
    Accepted for Eleventh International Workshop on Requirements Engineering: Foundation for Software Quality, REFSQ'05, Oporto, 2005.
30. Towards a methodology for rigorous development of generic requirements, C. Snook, M. Poppleton, and I. Johnson. REFT, Newcastle, 2005; LNCS 4157, Springer-Verlag, November 2006, pp. 326-342.

31. Formal Development of Mechanisms for Tolerating Transient Faults, D. Ilic, E. Troubitsyna, L. Laibinis and C. Snook. TUCS Technical Report, No.763, April 2006.
32. UML–B: Formal modelling and design aided by UML, C Snook and M Butler. ACM Transactions on Software Engineering and Methodology, 15(1), 2006, pp. 92-122.
33. Formalizing UML-based Development of Fault Tolerant Control Systems, D. Ilic, E. Troubitsyna, L. Laibinis, and C Snook. Proceedings of the Workshop on Methods, Models and Tools for Fault Tolerance, Jul 2007 (to appear).
34. U2B - A tool for translating UML–B models into B, C. Snook, and M. Butler. UML–B Specification for Proven Embedded Systems Design, J. Mermet, Ed., Chapter 6, Springer, 2004.
35. Formal Development of Mechanisms for Tolerating Transient Faults, D. Ilic, E. Troubitsyna, L. Laibinis, and C. Snook. REFT 2005, LNCS 4157, Springer-Verlag, November 2006, pp. 189–209.
36. Towards Feature-Oriented Specification and Development with Event-B, M. Poppleton. In Proceedings of REFSQ 2007: Requirements Engineering: Foundation for Software Quality 4542, pp. 367-381, Trondheim, Norway. P. Sawyer, B. Paech, P. Heymans, Eds.
37. Circuit Development with Event-B and Bluespec - RODIN Plugin Overview, I. Oliver. Presented FDL'06 (Forum for specification and design languages). September 19-22, 2006, Darmstadt, Germany.
38. Model Based Testing of an embedded session and transport protocols, V. Luukkala, I. Oliver. Presented at the 9th IFIP Int. Conference on Testing of Communication Systems (TESTCOM) and 7th Int. Workshop on Formal Approaches to Testing of Software (FATES), June 26-29 2007, Tallinn, Estonia.
39. An Investigation into the Introduction of Fault Tolerance Concepts with Event-B, I. Oliver, J. Colley, M. Butler. NRC-TR-2007-Awaiting Number, Nokia Research, Finland, 2007
40. Experiments and Experiences with UML and B, I. Oliver. NRC-TR-2007-006, May 2007, Nokia Research, Finland.
41. Formal Transformation of Platform Independent Models into Platform Specific Models, P. Boström, M. Neovius, I. Oliver, M. Waldén. Proceedings of the 7th International B Conference (B2007), Besançon, France, LNCS. 4355, pp. 186-200, January 2007, Springer-Verlag.
42. Mobile Internet Technical Architecture. IT Press, 2002.
43. A Formal Model of Context-Awareness and Context-Dependency, M. Neovius, K. Sere, L. Yan, M. Satpathy. In Proceedings of 4th IEEE International Conference on Software Engineering and Formal Methods - SEFM 2006, Pune, India September 11-15, 2006.
44. A Design Framework for Wireless Sensor Networks M. Neovius, L. Yan. In Proceedings of World Computer Congress - WCC 2006, Ad Hoc networking track, Santiago de Chile, Chile, August 20-25 2006.
45. Combining CSP and B for Specification and Property Verification, M. Butler and M. Leuschel. Proc. Of Formal Methods 2005, J. Fitzgerald et al., Springer, LNCS (3582) 2005, pp. 221-236.

46. How to Drive a B Machine, H. Treharne and S. Schneider. Proc. of ZB2000: Formal Specification and Development in Z and B, Springer, LNCS 1878 (2000) 188-208.
47. Mobile B Systems, A. Iliasov, V. Khomenko, M. Koutny, A. Niaouris and A. Romanovsky. Workshop on Methods, Models and Tools for Fault Tolerance, Oxford 2007.
48. The KLAIM Project: Theory and Practice, L. Bettini et al.. Proc. of Global Computing: Programming Environments, Languages, Security and Analysis of Systems, Springer, LNCS 2874 (2003) pp. 88-150.
49. Mobile Distributed Programming in X-Klaim, L. Bettini, R. De Nicola. Proc. of Formal Methods for Mobile Computing, M. Bernardo and A. Bogliolo, Springer, LNCS 3465 (2005) pp. 29-68.
50. A Petri net Semantics for pi-calculus, N. Busi, R. Gorrieri. Proc. of CONCUR'95, LNCS 962 (1995) pp. 145-159.
51. A Calculus of Mobile Processes, R. Milner, J. Parrow and D. Walker. Information and Computation 100 (1992) pp. 1-77.
52. Symbolic Model Checking: an Approach to the State Explosion Problem, K. L. McMillan. PhD Thesis, Carnegie Mellon University, 1992.
53. Model Checking Based on Prefixes of Petri Net Unfoldings, V. Khomenko. PhD Thesis, University of Newcastle upon Tyne, 2003.
54. Petri Net Semantics of the Finite pi-calculus Terms, R. Devillers, H. Klaudel and M. Koutny. Fundamenta Informaticae, 2006.
55. A Petri Translation of π-Calculus Terms, R. Devillers, H. Klaudel and M. Koutny. Proc. ICTAC, 2006.
56. A Petri Net Semantics of a Simple Process Algebra for Mobility, R. Devillers, H. Klaudel and M. Koutny. Electronic Notes in Theoretical Computer Science, 2006.
57. Computing Shortest Violation Traces in Model Checking Based on Petri Net Unfoldings and SAT, V. Khomenko. Proc. REFT Workshop at FME05, 2005.
58. Applying Petri Net Unfoldings for Verification of Mobile Systems, V. Khomenko, M. Koutny and A. Niaouris. Proc. MOCA, 2006.
59. Merged Processes - a New Condensed Representation of Petri Net Behaviour, V. Khomenko, A. Kondratyev, M. Koutny and W. Vogler. Proceedings of CONCUR 2005 (August 2005), to appear in Lecture Notes in Computer Science.
60. Using Formal Methods to Develop an ATC Information System, A. Hall. IEEE Software, March 1996.
61. Exception Handling in Coordination-based Mobile Environments, A. Iliasov and A. Romanovsky. Proceedings of 29th Annual International Computer Software and Applications Conference (COMPSAC 2005), IEEE Computer Society Press, 2005, pp. 341-350.
62. Towards Formal Development of Mobile Location-based Systems, A. Iliasov, L. Laibinis, A. Romanovsky, and E. Troubitsyna. Proceedings of Workshop on Rigorous Engineering of Fault-Tolerant Systems (REFT 2005), Newcastle Upon Tyne, UK 2005, pp. 53-64.

63. Structured Coordination Spaces for Fault Tolerant Mobile Agents, A. Iliasov and A. Romanovsky. LNCS 4119, C. Dony, J. L. Knudsen, A. Romanovsky, and A. Tripathi, Eds., 2006, pp. 181-199.
64. On Specification and Verification of Location-based Fault Tolerant Mobile Systems, A. Iliasov, V. Khomenko, M. Koutny, and A. Romanovsky. Proceedings of Workshop on Rigorous Engineering of Fault-Tolerant Systems (REFT 2005), Newcastle Upon Tyne, UK 2005, pp. 129-140.
65. On Using the CAMA Framework for Developing Open Mobile Fault Tolerant Agent Systems, B. Arief, A. Iliasov, and A. Romanovsky. Proceedings of SELMAS 2006 workshop at ICSE 2006, Shanghai, China 2006, pp. 29-36.
66. A Framework for Open Distributed System Design, A. Iliasov, A. Romanovsky, B. Arief, L. Laibinis, and E. Troubitsyna. Proceedings of Computer Software and Applications Conference (COMPSAC 07), Volume II - Workshop Papers, 1st IEEE International Workshop on Software Patterns (SPAC 2007), Beijing, China, IEEE Computer Society, Conference Publishing Services, 27 July 2007, pp. 658-668.
67. On Developing Open Mobile Fault Tolerant Agent Systems, B. Arief, A. Iliasov and A. Romanovsky. Software Engineering for Multi-Agent Systems V, LNCS 4408, R. Choren, A. Garcia, H. Giese, H.-f. Leung, C. Lucena and A. Romanovsky, Eds., Springer, 2007, pp. 21-40.
68. Rigorous Development of Ambient Campus Applications that can recover from Errors, B. Arief, A. Iliasov, and A. Romanovsky. Proceedings of Workshop on Methods, Models and Tools for Fault-Tolerance (MeMoT 2007), at the International Conference on Integrated Formal Methods 2007 (IFM 2007), Oxford, UK, 3 July 2007, pp. 103-110.
69. On Rigorous Design and Implementation of Fault Tolerant Ambient Systems, A. Iliasov, A. Romanovsky, B. Arief, L. Laibinis, and E. Troubitsyna. Proceedings of 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC07), Santorini Island, Greece, 7-9 May 2007, pp. 141-145.
70. Refinement patterns for rapid development of dependable systems, A. Iliasov. Proceedings of Engineering Fault Tolerant Systems Workshop (at ESEC/FSE), Croatia, ACM, 4 September 2007.
71. Finer Plugin Introduction.
[Online: http://www.iliasov.org/FinerPlugin.html (last accessed 15 Aug 2007).]
72. Smartdust. Online at http://en.wikipedia.org/wiki/Smartdust (last accessed 14 August 2007).
73. Model-Based Testing using Scenarios and Event-B Refinements, Q. A. Malik, J. Lilius, and L. Laibinis. Proceedings of Workshop on Methods, Models and Tools for Fault-Tolerance (MeMoT 2007), at the International Conference on Integrated Formal Methods 2007 (IFM 2007), Oxford, UK, 3 July 2007, pp. 59-69.
74. ISTAG Scenarios for Ambient Intelligence in 2010. [Online: ftp://ftp.cordis.europa.eu/pub/ist/docs/istagscenarios2010.pdf (last accessed 15 August 2007)].

75. BE4: The B Extensible Eclipse Editing Environment, J. Bendisposto, M. Leuschel, Published B2007: Formal Specification and Development in B, 7th International Conference of B Users, Besançon, France, January 17-19, 2007. Proceedings, 2007, Nr. 4355, pp. 270-273, LNCS, Springer Berlin / Heidelberg, ISSN:0302-9743, ISBN: 978-3-540-687.
76. A Generic Flash-based Animation Engine for ProB, J. Bendisposto, M. Leuschel, Published B2007: Formal Specification and Development in B, 7th International Conference of B Users, Besançon, France, January 17-19, 2007. Proceedings, 2007, Nr. 4355, pp. 266-269, LNCS, Springer Berlin / Heidelberg, ISSN:0302-9743, ISBN: 978-3-540-687.
77. Debugging Event-B Models using the ProB Disprover Plug-in, O. Ligot, J. Bendisposto, and M. Leuschel. Proceedings AFADL'07, June 2007.
78. Case study of a complete reactive system in {Event-B}: A Mechanical Press Controller, J-R. Abrial. Proceedings ZB 2005. [Online: http://www.zb2005.org/]
79. Rigorous engineering of product-line requirements: a case study in failure management, C. Snook, M. Poppleton, I. Johnson. To appear in Information and Software Technology, Elsevier, 2007
80. ISO 9126-1, Software Engineering, Product Quality - Part I: Quality Model. International Organisation for Standardisation, 2001.
81. Experimental Comparison of the Comprehensibility of a UML-based Formal Specification versus a Textual One, R. Razali, C.F. Snook, M.R. Poppleton, P.W. Garratt, R.J Walters. 11th International Conference on Evaluation and Assessment in Software Engineering (EASE), 2007, pp. 1-11.
82. Multimedia Learning, R.E. Mayer. Cambridge University Press, 2001.
83. Teaching readers about the structure of scientific text, L.K. Cook, R.E. Mayer. Journal of Educational Psychology, Vol.80, 1988, pp. 448-456.
84. Taxonomy of Educational Objectives: The Classification of Educational Goals: Handbook I: Cognitive Domain. B.S. Bloom (Ed.), Longmans, New York, 1956.
85. Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory, A.L. Strauss, J. Corbin. 2nd Edition, Thousand Oaks, California, 1998.
86. Usability Assessment of a UML-based Formal Modelling Method, R. Razali, C.F. Snook, M.R. Poppleton, P.W. Garratt. 19th Annual Psychology of Programming Workshop (PPIG), 2007, pp. 56-71.
87. Comprehensibility of UML–B: A series of controlled experiments, R. Razali, C.F. Snook, M.R. Poppleton, P.W. Garratt. Technical Report, DSSE, University of Southampton, [Online: http://eprints.ecs.soto.ac.uk/14426] (2007).
88. Software Considerations in Airborne systems and equipment Certification. DO–178B/ED–12B, RTCA/EUROCAE.
89. Systematic Software Development using VDM, C.B. Jones. Prentice Hall International, second edition, 1990.