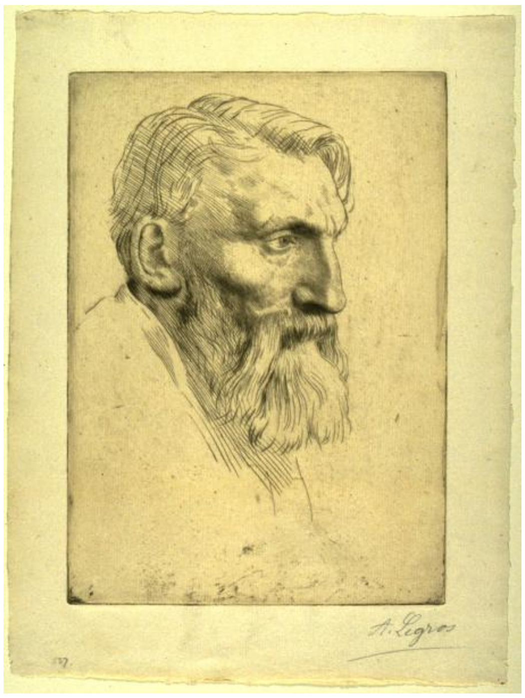
Project IST-511599

RODIN

“Rigorous Open Development Environment for Complex Systems”



RODIN Deliverable D22 (D7.3)

Assessment Report 2

Peter Amey (Praxis High Integrity Systems)

Public Document

2nd October 2006

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

Contributors:

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

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)

Anticipated Changes

Update with progress from year 3.

Table of Contents

1 Introduction

1.1 Background

This document presents the results of the technical review and assessment conducted at the end of the second year of the three-year RODIN project. A final release of this document will follow in month M36. This document is a RODIN project deliverable.

The assessment was carried out according to the Procedure for Technical Review and Assessment [1], which established the metrics for RODIN, based on the contributions of goals and criteria from each RODIN partner. Note that [1] was revised and improved in the light of experience gained from production of the first version of this report in M12. (Deliverable D14).

1.2 Scope

The scope of this deliverable includes the five case studies under work package 1 (WP1), the methodology developed under work package 2 (WP2), the open source formal methods tool kernel developed for work package 3 (WP3) and the plugins developed as part of work package 4 (WP4).

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 results that have been accomplished within the first two years by of each of the work packages.

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 M24. Section 4 discusses the metrics applicable to the WP2 methodology part of the project. Section 5 looks at the metrics from each of the 5 case studies from WP1. Section 6 looks at the metrics for the WP3 tool kernel. Section 7 looks at each of the plug-ins developed as parts of WP4. Section 8 contains the references used in this document.

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) [2]:

"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."

Four specific measurable objectives are also given in the Description of Work:

1. 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.
2. 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.
3. 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.
4. 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.

The Procedure for Technical Review and Assessment [1] identified measurement criteria for each of the work packages pertaining to the objectives listed above. The measurement criteria are listed in section 2.2.

2.2 Measurement Criteria

The following measurement criteria are desirable general properties of the various RODIN products, and aim to identify the presence and quality of the following properties:

- **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.

2.3 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 [1] and updated as a result of year 1 review and year 1 experience. The set of questions varies for each of the case studies in WP1 and for the method (WP2). 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 of the questions has codes attached which link them back to the measurement criteria in section 2.2. These codes are listed in brackets next to each question. E.g. (C, S) or (S, X).

The following deliverables were assessed:

- Methodology
- 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: Code Generation
- Plug-in: Brama

For each deliverable, 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 deliverable was applicable, but with no link with the producers.

The review process conducted was as follows:

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

2) The producer self-assessed his 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 HIS collated the validated written assessments into this overall assessment report D22.

The producers and reviewers, were asked to provide answers to their corresponding set of questions. These answers consisted of 2 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.

3 Overall Assessment Results

3.1 Overview

The structure of the following sections follows that proposed in the assessment procedure report [1]. This has been done deliberately in order to facilitate comparison with previous as well as future assessment reports.

At this stage it is important to show that:

- progress has been made in each area
- that progress is consistent with the overall RODIN objectives (as re-stated in Section 2.1).

Sections 4 onwards detail the responses to the assessment questionnaire for each of the areas under review. It should be noted that the progress reports in this document provide only a very brief summary of each RODIN constituent activity. Fuller details can be found in other scheduled deliverables [7,8,9,10,11,12,13,14].

3.2 Result summary

The qualitative results are summarised in the following table.

Area	Progress
Methodology (see 4)	This area is progressing well, the “Event B” notation with minor extensions has started to be applied to the CDIS case study and the results indicate a dramatic reduction in the length of the specification. A fuller report is in deliverable D19 including the invaluable input from the other case studies.
CS1 (see 5.1)	The Lyra model established in the first year has been extended to include reasoning on fault tolerance, the representation of possible errors and error recovery. In addition this work has created a basis for formally verifying consistency as well as performing model based testing of the Lyra UML models.
CS2 (see 5.2)	Solid progress has been made in terms of model development, methodology and tools on the case study. • ATEC have undertaken an independent evaluation using a pilot study

	• Aabo Academy has developed a Failure Management System (FMS) specific pattern • Southampton University has developed a framework to extend the generic model • And Produced a tool (Requirements manger) to support a generic model
CS3 (see 5.3)	The case study is progressing as expected with the formalisation of various subsets of the MITA platform and the formalisation of the infrastructure and techniques to allow MDA to be used in a more formal manner. The primary driver so far for this case study has been the NoTA project within Nokia, which will be ending in December 2006. Further work within Rodin will take place in a continuation project focussing on the nature and composition of services within a NoTA and extended framework including product line development in this environment.
CS4 (see 5.4)	Event-B development of this CDIS specification subset was undertaken. Most of the functionality of the subset has now been incorporated into the Event-B development. The Event-B models were checked and verified using the B4free tool. The specification is now more comprehensible and easier to reason with.
CS5 (see 5.5)	Work has focussed on the formal stepwise development of the infrastructure (middleware) for supporting the case study. The emphasis has been on the design of "context-aware mobile agents" as well as the rigorous development of fault tolerant mobile agents to run on the specified infrastructure. Close integration of this case study with the main RODIN developments have been achieved.
Open Tool Kernel (see 6)	Good progress has been made in that: • Major technical decisions concerning the platform container and plug-in mechanism have been made • The Event-B language definition has been finalised • The overall architecture of the platform, the plug-in mechanism and the Event-B kernel tools have been specified • A prototype of the platform and Event-B kernel tools have been developed and delivered.
Mobility Checker (see 7.1)	Significant progress against the RODIN DoW has been made, in particular, functionalities in the mobility plug-in have already proved much more efficient than similar tools provided by the standard π-calculus tool implementation (Mobility Workbench from Uppsala U.).

ProB (see 7.2)	The tool has been reworked as a plug-in and can now animate B formal models. Functionality include automated consistency and refinement checking. The tool has proven to be useful on various industrial case studies and successfully identified errors.
Code Gen. (see 7.3)	The B2J translator has been set up and experimented on small size case studies. This translator combines an event recomposer and a Java code generator.
Brama (See 7.4)	This is a new plug-in for the RODIN toolset. The tool animates B models according to the standard mathematical B semantics, with valuated sets and constants. The predicate evaluator, heart of the tool, has been extensively validated as it has been embedded in an application for verifying trackside safety critical invariant data. The tool behaves satisfactorily on simulating small and medium models, with very good response time. Currently tool optimisations are being investigated.

3.3 Metric summary

The numeric metric results are summarised in the following tables.

N.B: No metrics have been derived for the (WP2) methodology as a qualitative assessment was undertaken.

Grade	CS1	CS2	CS3	CS4	CS5
0 (failure)	0	0	0	0	0
1 (weak)	0	0	0	0	0
2 (average, as good as ...)	0	0	3	1	0
3 (good)	0	3	2	3	3
4 (very good)	2	3	1	2	3
5 (excellent)	1	0	0	0	0
N/A	0	0	0	1	0

Interim grade assessment summary for case studies.

Grade	Open Tool Kernel	Petri Net	Constraint Based Checking	Code Gen.	Brama
0	0	0	1	0	2
1	0	0	0	0	2
2	0	0	1	0	0
3	3	3	12	12	9
4	6	11	9	8	9
5	12	8	3	2	4
N/A	5	4	0	4	0

Interim grade assessment summary for open tool kernel and plugins

The tables show that, where progress has been made, the progress is largely satisfactory and that results are up to the expected standard if not better than expected.

3.4 Assessment overview conclusions

The assessment shows good progress towards achieving the overall objective of RODIN. It gives convincing evidence that specific technical objectives set for year 2 in WP1-WP4 (see section 2.1) have been achieved. *As an example, you can refer to prototypes of the basic platforms and plugins produced, experimented with and successfully assessed in D22*

In conclusion the assessment shows that the progress has been made in all technical areas of WP1-WP4.

In summary the assessment has been successful as:

(i) the majority of grades given are very good and excellent;
(ii) the number of criteria which have not been assessed has considerable reduced comparing with year 1 assessment and;
(iii) the justifications of the grades given are sound.

- You can say that the improved assessment procedure proven to be working

4 Methodology Assessment

This section addresses the difficult problem of assessing the development methods that RODIN will propose for the creation of fault-tolerant systems. Work Package 2 has produced deliverable D19 which sets out the current status of the work on methodology. In particular, D19 reports on progress on the "issues" identified during year 1 and reported in D9 and, furthermore, begins the task of reporting how the case studies are informing our views on methodology. As has been made clear in all assessment documents, the main evaluation of methodology will have to come in the final year of the project. (See also the comments on difficulties with comparison in Section 5.2.2.)

4.1 Introduction

As we stated last year in D9, we originally envisaged (and continue with this opinion) that the RODIN project will make its major instantiation around a development of Jean-Raymond Abrial's Event "B" notation.

We feel the resolution of issues listed in D19 is significantly reducing the risk to our project when compared with the situation as at the end of year 1 and reported in D9.

Experimental use of Event B and other notations has been undertaken in the case studies and is reported in D19. We repeat the observation in D9 that it is one of the key advantages of the consortium is that there is a diversity of views about these issues.

We divide our more detailed comments depending on the level of tools involvement.

4.2 Assessment in conjunction with tools

The RODIN tools platform is only now being deployed on the case studies. Clearly no evaluation is yet possible although there has been significant experimental work with earlier B tools on the CDIS study and the very good performance results with the Newcastle model checking plug in are important positive pointers.

4.3 Assessment as Stand-alone

Considering the developed methodology outside the context of the supported tools we can report progress which is described already in D19.

The results from CDIS indicate that with minor extensions the Event-B notation will offer a dramatic reduction in the length of the CDIS specification.

We are also encouraged by the relative ease of adoption of the new methods by Atec who are specifically representing the less experienced users in the RODIN project. They have certainly identified (see D19) areas where extra education and documentation would help but we anticipate that Abrial's planned book will cover many of these aspects.

Of course, all of these comments will have to be reviewed once the tools are available.

We note in particular:

- Section 5.1 of this report describes the Lyra work on fomal reasoning about fault tolerance and its connection with the B methods
- Section 5.2 of this report discusses the exposure of the new users to the RODIN methods
- Section 5.4 indicates dramatic progress on reducing the length of the earlier (VVSL) CDIS specification
- Section 5.5 reports on extensions to cover ambience (the CAMA approach)

More details on all of these topics can be found in Section 2 of D19 (the final text of Section 5.3 of this report was not available when this text was drafted but again D19 has details (see Section 2.3 of D10)).

4.4 Metrics

We are fortunate to have access to an experimental psychologist who is employed on the DIRC project and we are discussing possible interview techniques to be deployed in year 3 of the project for a more quantitative evaluation of RODIN methods (reflecting the different backgrounds of the users).

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 / model-based testing techniques) for development of telecommunication systems and communication protocols. The work of the case study focuses on formalisation and validation of the design method Lyra developed in 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 by providing automatic translation of UML-2 based Lyra design flow into the formal framework. A degree of automation in translating UML2 models and model transformations is perceived by Nokia as a major criterion for evaluating success of RODIN.

5.1.2 Current Status

During the first year of the RODIN project our work on the case study mainly focused on creating a theoretical basis for automation of the Lyra design process. In particular, we have developed formal specification and refinement patterns reflecting essential Lyra models and transformations. This allowed us to conduct verification of the Lyra development using stepwise refinement in the B Method.

In the second year of the RODIN project our work on the case study has progressed in three directions. First, to incorporate formal reasoning about fault tolerance into the formalized Lyra development flow, the specification and refinement patterns for Lyra models have been extended with explicit representation of possible errors and error recovery. Second, to automate translation and verification of Lyra UML models in the B Method, we have developed an approach to verifying the consistency of the provided UML models. The achieved results create a basis for developing a formally verified UML profile for Lyra. Third, to investigate the use of model checking and model-based testing techniques in the context of UML and B, a preliminary methodology for model-based testing of Lyra models has been developed.

5.1.3 Progress

Below we give the evaluation of progress against the objectives of the case study stated in the RODIN Description of Work. The objectives are given in italics

Investigate the benefits of using refinement approach versus algorithmic verification to verify system decomposition and composition

During the second year of RODIN, essential work has been done in formalizing the Lyra design methods and patterns to develop the (formally verified) Lyra UML profile with constraints. Part of these constraints is derived from the Lyra specification and refinement patterns in B developed during the first year. The work done in UML and B allows us to create a bridge between these approaches in order to verify the Lyra design flow. In addition, the basis for automatic generation of Lyra test cases has been developed and published.

Progress evaluation: very good

Investigate model reduction techniques and proof methods for data abstractions and the use of model checking to verify correctness of system components

Nokia has started investigation on developing the appropriate abstract data types for the domain of telecommunicating systems. In the third year of RODIN we are going to use these results to incorporate the developed data types into our UML and B models.

Progress evaluation: satisfactory

Investigate applicability of formal reasoning techniques about fault tolerance in this application area

To incorporate formal reasoning about fault tolerance into the formalized Lyra development flow, the specification and refinement patterns for Lyra models have been extended with explicit representation of possible errors and error recovery. The extension has affected the specifications of service components directly responsible for controlling the service execution flow (called service directors). The recovery mechanisms allowing a service director to retry the failed service execution as well as to "roll back" in the service execution flow have been incorporated in the specification of a service director. Moreover, in the refinement steps modelling service decomposition and distribution over a given network, the fault tolerance mechanisms have been distributed over the involved service components.

Moreover, we are going to extend the Lyra UML profile with the fault tolerance concepts. This would allow the patterns defined in B models to be checked and/or generated in the corresponding UML models. In addition, our work on automatic test generation for Lyra models is also going to address the fault tolerance issues.

Progress evaluation: good

Validate top-down and bottom-up formal techniques and supporting tools

At the moment we are testing U2B, Software Architect and some other commercial Eclipse-compatible tools to automate translation and validation of Lyra models.

Progress evaluation: satisfactory

5.1.4 Metrics

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

The developed concepts, methods, and tools support the design method Lyra, which is already well integrated into the existing development process in Nokia. Moreover, the input language (genuine UML) is widespread and well accepted in the industry. All this makes the technology transfer much easier.

Grade: [5]

2. How much support they provide for a more rigorous development process?

a. Specifically, how many new tasks in the development process can be tackled using the methods developed in RODIN? (C)

The first three (out of four) Lyra development phases are (to a large extent) formalized and verified by the developed methods for both UML and B. Automatic test generation (to be implemented as a RODIN plug-in for model-based testing) and full support for modelling the fault tolerance mechanisms are still under development.

Grade: [4]

3. How much support they provide for automation of the development process?

a. Specifically, how many new tasks in the development process can be tackled using the methods developed in RODIN? (C, X)

We are working to provide significant automation for the first three Lyra development phases. The created B patterns and the work on the Lyra UML profile allow us to expect automatic translation of Lyra UML models into B, at which point the automated refinement and model-based testing techniques can be applied to verify these models.

Grade: [4]

5.1.5 Future Evaluation

The second year of the RODIN project has confirmed adequacy of the chosen metrics. The emphasis of the third year will be on providing full tool support for the developed methods. This would allow us to make complete evaluation of the achieved results at the end of the third year.

5.1.6 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 provide a solid foundation for automated support and verification of the Lyra design method. During the third year of the RODIN project we are going to bridge together three major directions of the case study development (formal verification using the B Method, creation of the formally verified Lyra UML profile, and model-based test case generation) as well as work on reusing tool support (developed within RODIN) for automating the Lyra development process.

5.2 CS2 – Engine Failure Management System Assessment

5.2.1 Introduction

The following summarises the evaluation of case study 2 for year two of the RODIN Project. The case study development has been described in the case study deliverable D18.

5.2.2 Current Status and Progress

Year 2 has seen some model development and progress on methodology and tools on the case study.

- ATEC undertaken an independent evaluation using a pilot study
- Aabo Academy has developed a Failure Management System (FMS) specific pattern
- Southampton University has developed a framework to extend the generic model
- And Produced a tool (Requirements manger) to support a generic model
- And developed its UML-B plug-in tool in line with the new RODIN Platform

The case study metrics used to assess the usefulness of the technology from ATEC's view is given in the next section.

After the first year evaluation ATEC now feel that to quantify objective comparisons with existing methods is no longer viable due to several factors.

1. Difficulty in obtaining comparable metrics which are meaningful.
2. Quantifying modelling processes objectively during learning and research has proven impractical e.g. where methods are being explored and experiences developed it was not feasible to quantify measurement during learning.

These and other difficulties have led ATEC to conclude that the most effective way to provide measurement is through a subjective viewpoint of the technology supported by argument .

5.2.3 Metrics

The evaluation criteria are not orthogonal, that is to say contribution to one criteria may also affect another e.g. quality can affect cost, certification, maintainability.

The first four criteria concern the usefulness and usability of the technology for ATEC future use. The fifth criteria concern evaluating the finished model in terms of its contribution to genericity. The last criteria addresses the justification of the UML-B technology, identifying areas that could be improved, and identifying benefits that may have been gained from adopting the new UMLB version on the RODIN platform.

The criteria and their discussion are given below. An ordinal scale has been adopted to summarise the value of the technology in meeting the goal.

Poor – Significantly worse than existing methods.

Diminished – Only slightly worse than existing methods.

Standard - The same as existing methods .

Good – Better than existing methods.

Excellent - Significantly better than existing methods.

1. Evaluate the reduction of cost of development in future products.

    a. What is the cost of learning the methods? (C, O, U)

Learning the methods in the context of a developing research environment has been slower than I expected as a novice. I originally intended to develop the behaviour of the generic model in year 2 but felt the need to learn more about "B" and UML before I could realistically tackle this. This was a contributory reason for adopting a pilot study. A brief summary view is given in the following table.

Effort required learning/applying B	**Effort required learning/applying UML**
Notation difficult	Intuitive
Lacks process –what and how to refine	Difficult to identify objects
ProB tool easy to use, animator helps provide assurance to learning	Easy to change
Proofs difficult	Difficult to verify

It is difficult to qualify how much the problem of learning is down to lack of experience. It would appear that some students in an academic environment learn the technology relatively quickly. So in theory the cost of learning the methods may not necessarily be any more time consuming than training in any technology.

However a distinction should be made between learning the methods and applying them to a particular problem domain. I hope to supplement my own experiences with learning and applying the technology with the more experienced academic partners in the case study in the final evaluation report. (The impact of this may be to identify a stronger set of guidelines in promoting RODIN methods to the novice industrial user.).

b. How do costs of using methods compare with the existing method (at each stage of the lifecycle)? (C)

The cost benefits of the technology that are to be gained are seen in the context of how modelling can reduce the workflow of the software lifecycle. ATEC has previously not performed modelling in the FMS domain. It views the role of modelling as an extension of the existing workflow processes but expects savings to be made in efficiencies.

The scope of modelling to reduce costs (i.e. effort) in ATEC's software lifecycle is expected to be achieved by;-

a) Modelling provides a means to help with requirement analysis which reduces the risk of late changes due to the wrong requirements being met.

b) Modelling provides a means by which the intended requirements can be translated into design and implementation accurately with minimum errors.

c) Provide a reference to assist the efficiency of other workflow processes e.g. testing, documentation and certification.

d) Not significantly increase the burden on other workflow processes such as maintenance and testing.

The effectiveness in these areas are also addressed in the metric goals that follow but a summary is outlined here.

Assisting requirement analysis

The case study has identified that some states in the application requirement can be held longer than originally intended. Traditionally reviews of requirements at different stages in the lifecycle are used to identify such anomalies, but modelling has provided an effective aid to raising this concern early.

However the modelling process is not independent of review and the choosing of what to model and how to reason about the model is still open to scrutiny and guidance. In the pilot study, the event approach encouraged the modeller to identify events but required knowledge of all the domain functionality. The identification of use cases and system roles (used in UML methodology) may help to ascertain the focus of the events.

The use of safety invariants and refinement can also provide feedback to the reviewer to test assumptions about the requirement relationships being modelled. For example the use of a safety invariant which checks that the system freeze flag is latched (i.e. does not reset) could have been adopted.

Not all requirements need to be modelled if the model is used just for analysis and not as a complete specification for design. To some extent this analysis approach was adopted in the pilot study where only some requirements were explored.

During the study, it was felt that although the B approach was precise and rigorous it lacked flexibility for quick requirement exploration by a novice. The rigour of the approach was felt to require a lot of effort to maintain consistency in the requirement i.e. by the satisfaction of proof obligations. The pilot study experimented with a development approach to establish intended functionality early in the model development to overcome this perceived inflexibility. An early stage of the pilot study tried to focus on establishing what the basic functionality of the model was rather than produce a precise consistent incorrect one. The remaining stages then refocused on precision and consistency through use of formal refinement. The approach was relatively successful.

Contributing towards error free design

The traditional approach to design by ATEC is to map requirements to a design architecture without recourse to requirement modelling. Reviewing and testing are used to validate and verify the design.

In the B modelling approach the requirement model evolves into design through refinement of abstract requirements into deterministic ones. It was easy to see that the final deterministic refinement in the pilot study was not too far removed from implementation code (ref D18 Fig 3-9).

The process of translation into more concrete design was controlled by formal refinement which included the use of gluing invariants and the use of the model checker and prover tools, (ProB, Click' n' Prove) to assist the process.

The refinement of the B model to a level suitable for coding took longer than I would reasonably expect for such a small example. The need to revisit abstractions in order to incorporate a new state change felt particularly cumbersome. I envisage that in a larger scale of model development this reworking of a model may be a significant burden. In theory at least may be desirable to have some form of retrenchment where the impact of a change on a model could be minimised.

However lessons were learned from the pilot study. Significantly that when attempting to satisfy a proof, the abstract model was altered which lead to incorrect behaviour occurring in the level of the abstract which was not detected. The refined model was validated as correct, but if the abstractment (i.e. the abstract machine from which the refinement was generated) was to be reused, then its altered behaviour would now be different and could lead to incorrect behaviour being developed from that abstractment. The lesson learned was how to alter a model to cater for new events.

The inclusion of a stronger invariant in the model would have prevented the alteration in the abstractment to be allowed, but demonstrates how a novice could develop weak invariants resulting in a less robust model. It could also be argued that the need to validate all behaviour on any alteration of an abstractment would have trapped this error, but this could be time consuming and subject to identifying relevant test cases. The creation of test scripts in the animator which could be repeated against each refinement would assist the efficiency of validation.

The modelling in the pilot study has been concerned with the translation of functional requirements into design, however non-functional requirements which include performance issues, compliance to certification etc have to be addressed in design. It is not clear how the modelling process affects these concerns. The ease of certification is examined in goal 3.

Contribution to other work flow processes

The modelling process is expected to reduce testing as designs are expected to contain less errors and so reduce the need to re-test. Testing of source code is not going to be undertaken in the case study. The maintenance of the models will be examined in the final year.

RODIN Grade – GOOD

Grade: [4]

The methodology effectiveness is scoped by modelling s' contribution to the software lifecycle. Modelling in general contributes to analysis of requirements but the formal process is particularly suited to providing an effective way to provide a complete and correct specification for design. The process was time consuming as time was spent on reworking the model, however this can be attributed to inexperience with the methods.

2. Evaluate the impact on maintaining current quality levels.

The main focus here is not to increase the levels of errors in the product and software processes. RODIN methods can contribute to quality by reducing errors, this is achieved in modelling by.

1. Providing an effective means to elicit and analyze system requirements for completeness and correctness i.e. ensuring the requirements are what are intended (error detection).
2. Provide an effective method to accurately translate the requirements to a development implementation (error avoidance).

The approach used to develop the pilot model is described in the D18 report for case study 2.

How effective are methods at detecting errors? (C, S)

The pilot model did not identify any additional concerns over the system requirements. (However in year 1 a concern over the confirmation mechanism had been identified).

This was not too surprising as the example was simple, and the requirement well understood. However the model was deliberately underspecified in its requirements, as the intent was to limit the scope in order to develop the model to a fairly deterministic level.

The event style of decomposition encouraged me to decompose the model in terms of events but I also found it lacked focus on how to decide what the events are and when to develop them. For example in the initial abstraction in the pilot model I tried to consider all the events associated with the dual sensor system in a holistic way, but in a more complicated example all events of the system may not be so easily identified. The consideration of use cases used in UML may be helpful to identify the functionality here.

Another observation is that the decomposition of the events into smaller events are in effect an arbitrary design decision, blurring the distinction between modelling system requirement and design requirement. For example, I chose to represent the validation of the dual signal sensor as several internal events but could have chosen just one. One significance of this, is that the reasoning about system requirements may not be isolated from the model design with the consequence that a different model design could lead to different analysis of requirements.

This is not necessarily a restriction to analysis, as the developing architecture of the model, I think, will raise derived requirements in order for the right system to be developed. For example a consideration of different internal events will require some form of sequencing which may be logically derived from the system requirement but could also take into consideration efficiencies in processing.

(In the Aabo model of FMS, sequencing of stages were done in various cycles, where several inputs were processed before the next stage was undertaken.)

Another consideration was that the model could contain false restrictions. For example the pilot study implied the difference test occurring after the magnitude tests however it could be reasoned that the scheduling of the difference test may also depend on the most frequent magnitude test.

The use of the animation facility of the ProB tool was particular useful in demonstrating the requirements to non B users to elicit further requirement analysis.

How effective are methods at avoiding errors ? (C, S)

In addition to specifying requirements for analysis, the model refinement process in B notation provides an accurate method to incorporate requirements into design and so avoid unintended behaviour i.e. errors occurring due to incorrect translation. It achieves this accuracy using RODIN methods in several ways

a) By the gradual decomposition of the requirement from an abstract description to a deterministic one.

In the pilot model, the non deterministic decomposition of the requirement leading to a deterministic refinement was achieved in several stages (outlined in D18). My main observation of the process is that identifying what to consider for abstraction is arbitrary and knowing how much determinism to introduce at what stage of development requires modelling judgement and experience. This could lead to poorly constructed models, which may be difficult to maintain resulting in further errors. Even the pilot study illustrated that a model could be developed with the confirmation mechanism not decoupled from the validation operation, which if left unrefined could be inefficient to maintain. However modelling does not encourage poorly constructed design models so in this sense it is no worse than traditional methods.

Another observation is, there appears to be little assurance to show when all requirements have been completely modelled, as requirement coverage of a model is not easily traceable in the B specification. This could result in designs omitting requirements rather than designs translating requirements incorrectly.

b) The use of formal notation to express the functionality more accurately

As a novice I found it initially difficult to express functionality using the B notation. A wider concern would be that designers who lack knowledge of B may wish the requirement model to be documented in natural language and lose the benefit of the notation. Consequently their lack of knowledge could prevent them utilising the model in a formal approach to design.

c) Control of model development by the use of invariants and formal refinements supported by the model checker and animator tool (ProB) and the prover tool (Click' n' Prove B4free).

Each machine was initially verified using the ProB model checker and animator. The animation was particularly useful in ensuring that the model was what was intended as it demonstrated the functionality of each stage very easily otherwise this assurance could only be achieved through analysis, which is less illustrative.

A particular benefit being that the intended functionality could be demonstrated to different types of domain user. E.g. an ATEC systems engineer was able to comment on the behaviour of the model without any knowledge of "B".

This approach ensured that the right system was being modelled albeit a consistent different one and provided a validation of concept.

The model checker successfully detected type errors and ensured invariants were not violated (for the given set coverage).

The machines were then formed into a series of refinements. This allowed me to revisit the development and control its consistency through the strengthening of invariants and using the refinement checking facilities of ProB, and then later, the B4free prover.

The final refinement chain was fully proven and animated.

The method though rigorous in its approach to developing consistency between an abstractment and refinement, did have some weaknesses.

- It was easy for a novice to generate invariants that were incorrect which could lead to incorrect specifications. This of course is less likely to occur with experience. However I felt the need to check that the invariants were correctly specifying the intention to provide some assurance. This assurance was achieved by testing the invariant i.e. by generating states that would be expected to violate the invariant and confirming the violation would be detected using the model checker and animation facility of ProB.
- Formal refinements and invariants could only constrain the model, they cannot confirm the functionality that is derived. Functionality could only be validated through the animation. Furthermore formal refinement maintains consistency within the model development but cannot ensure functional consistency. e.g. in the pilot study an early model introduced a magnitude test on only one input instead of two, this remained undetected until validation.
- Using the event approach for data intensive systems may be more problematic. For example, if we considered events associated with inputs additional to the pilot study's dual sensor, then the number of possible output event combinations could be large. Identifying all permutations of output states for all events may be impractical and lead to difficulties in maintaining guards in the output events.

The existing tools have their limitations which can affect their effectiveness in detecting process errors.

1. The model checker can take several hours to check the state space for even small examples, which can encourage the tool to be abandoned.
2. The prover tool does not detect deadlocks? This caught me out where the model was proven, but found to have unwanted deadlocks when run in the model checker. The drop off (a missing guard) was easily corrected and the model rechecked and re-proven.
3. The restriction of data sets to an unrealistic small number of instances in order to prevent state space explosion occurring is a concern for large scale use of the model checker.

Requirement manager tool

The need to develop a requirement manager tool to assist with instantiating the generic model with compatible data was identified from year one work. The manual process of instantiating the generic model showed how easy it was to enter inconsistent data. However the entering of valid data (i.e. data that is intended to configure the requirement), even with the tool, still relies on its validation by the instantiated model. This may still be subject to error. This could be improved in the future by introducing invariants to constrain the data of a given configuration.

Errors in compliance to non functional requirements

Modelling in the case study has been concerned with modelling functional requirements. However how an implementation from the model can be constrained by non functional requirements is also a concern to quality. Non functional requirements include performance, capacity, compliance to standards. Non-conformance to these requirements are regarded as errors in the process. Some ways the methods may affect these requirements are;-

1. Certification – Can RODIN methods meet certification requirements, this is covered in Goal 3 below.
2. Capacity – The direct implementation of the design from the requirements model may not be able to achieve the capacity and performance required of the implementation without modification to the design architecture. Traditionally re usable architectures with known performance characteristics are commonly used in industry to minimise this type of risk. It may be possible to identify different design patterns in B that can be applied to the higher requirement abstractions to achieve this.

Error conclusion/ summary

The B model was useful for specifying system requirement detail for a design but has some limitations to general requirement analysis, i.e. lacks a strong methodology to identify what requirements it needs to explore and when in a model. The rigour of the modelling process also means it is less flexible to accommodating changes quickly in order to explore requirements. The strength of the formal modelling process is that it provides a strong process to translate the requirement into design and avoid errors. However the process does not specifically address non functional requirements which can also affect the design.

RODIN Grade – GOOD

Grade: [4]

The rigorous process was shown to be successful in maintaining quality in the pilot study and should contribute towards improved quality in future products. However it is difficult to judge the efficiency of the process as time was spent on reworking the model, due to inexperience in the method.

3. Evaluate whether the improvement of ease of certification has been achieved.

a. How do methods contribute to certification standards? (C)

b. Are the products suitable for independent verification? (C, S)

The aviation industry uses the RTCA guideline Do-178 (EUROCAE ED-12) to help achieve airworthiness certification of its software. The guideline is concerned with how the system requirements allocated to software in particular safety, are satisfied. A safety classification is used which is then related to a 'level of rigour' to be applied to achieve the certification.

The FMS system has the potential to affect safety and could be classified as potentially catastrophic (Failure conditions which would prevent continued safe flight and landing) and would have the corresponding DO-178B rigour Level A applied to its software lifecycle processes.

The current standard does not address aspects such a Modelling, OO, and only briefly refers to formal methods. This maybe a concern for certification of systems in UML-B which embraces all these aspects. However the guideline is not mandatory and is intended not to preclude these technologies.

Furthermore there is support in the guideline, for formal methods, as an acceptable method for obtaining certification [Do178B 12.3.1].

In my brief investigation of the guideline I have identified the following areas which may affect or contribute to the ease of certification.

Table 1

E – easier to certify D- more difficult to certify

No	Description	Consideration	E/D
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. Ref (RTCA DO-178/b) Sect 5.1.2 ,5.5,6.1,6.2,11	"Information is traceable if the origin of its components can be determined." this assists verification processes of review and analysis for requirement coverage. But in B refinement models the introduction of requirements and their development are not explicitly labelled which may hinder the analysis and assurance of a model as ; A) It would compromise the ability to show traceability from and to the system requirements B) Compromise visibility of derived requirements for safety analysis A solution may be to explore ways to tag model development detail against requirements in order to generate traceability matrixes.	D
2	Compliance of safety requirement	By adopting invariants that reflect system safety constraints then confidence that the model and its implementation will conform is increased which should assist certification.	E
3	The software architecture and low level requirements are developed from the high level requirements.	The refinement process assists in development of compatible low level requirements through refinement checking mechanism	E

	The goal is to avoid errors during the development process 4.4, 5.2	controlling what changes are permissible. Identifying the data invariants in the development with refinement proof produced by the tools may also support the verification of this requirement in the model.	
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 from. Ref (RTCA DO-178/b) Sect 5.5	1. The guideline distinguishes design requirements from high level requirements in its analysis to allow for separate review and analysis activities. But can this distinction be made so clearly in B model refinement development where it would seem that modelling of high level requirements and design are closely bound. The tracing of high level requirements to the design is compromised as requirement is not tagged in the model.	D
5	Test cases are required to demonstrate compatibility of the system with requirements and provide assurance of requirement coverage. Ref (RTCA DO-178/b) Sect 6.2	Conventionally it may be argued that requirement compatibility and test coverage is intended to be only applicable to testing of the target code and is not concerned with modelling. However it would be desirable to demonstrate the requirement coverage in the B refinement model so that the resulting implementation code would not introduce incomplete, or unintended requirements. Mapping of requirements with animation test cases using ProB should help to give this assurance.	E
6	Dead code should be removed Ref (RTCA DO-178/b) Sect 6.4	Although the guideline requires structural coverage analysis to be applied to the source code the	E

		identification of any dead model code would provide assurance in the source code and assist certification. Analysis of the model may achieve this but it would be assisted by the tracing of model detail to requirements.	
7	Deactivated code is allowed but needs to be verified and shown not to be inadvertently executed. Ref (RTCA DO-178/b) Sect 6.4	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 generic model behaviour.	D
8	Qualification for tools are 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 eliminated or reduced. Ref (RTCA DO-178/b) Sect 12.2	The development tool U2b would not need to be verified as its output will be verified during a model verification. The requirement manager may need to be verified depending on how the requirements are verified in the model verification. E.g. what ensures that the instantiation is what is intended? The verification tools ProB and B4free would be required to be qualified if their verification result is to be relied on. The onus would appear to be on the user of the tool to gain qualification agreement This may not be easy to achieve by a novice to formal methods.	D
9	User modifiability- What ensures that a designated non modifiable component of the software, is protected from unintended modification . Ref (RTCA DO-178/b) Sect 5.2.3.	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.	D

RODIN Grade - Standard

Grade: [3]

The certification guideline can be still be applied. However the level of effort that may be required to conform to the existing guideline would be expected to be slightly higher than the traditional approaches [e.g. ref items 4 and 8].

4. Evaluate the reduction of cost of late requirement changes.

This goal is referring to the robustness and maintainability of systems developed from a model approach.

a. Do methods enable early validation of requirements? (C, S)

Early requirement detection has been covered in goal 2 above.

b. Is the system design easy to understand ? (X, U)

A main aim of the case study is to reduce the semantic gap between the application domain with the design in order to assist maintenance. The third year will assess how well mapping to the system requirements can be achieved.

c. What is the impact of a typical change? (U)

The Pilot study considered change in model development by the introduction of a new “confirming state” to the model. This was accommodated by updating all previous abstractments to recognize the new state. The process was improved by the consideration of a gluing data invariant which reduced the amount of change required in the abstract models, however even this required some rework. Although the iterative process was time consuming, it could be improved on through experience. The pilot study architecture is seen as fairly easy to accommodate a new requirement i.e. simply by adding a new event. However it is not clear how the dual sensor architecture could be scaled up when considering different tests for different types of dual input sensors.

The impact of changes in the model on the design code has not been investigated. In general, in ATEC s’ experience of traditional development, the largest impact on maintenance has been the need to perform extensive module retest. Whilst it may be argued that using formal methods can replace the need for module testing, if module testing was still required, then the impact of model architecture changes on design would need to be considered. (Module testing may be required for example to provide assurance that all the paths through the target source code execute as intended with no side effects).

The refinement approach does not necessarily consider architectural design over its functional decomposition. If not addressed this may create a maintenance issue in the final code where a small functional change could have an unnecessary amount of impact on the design.

The impact of several changes on the Aabo and generic FMS models will be considered in the final year.

RODIN Grade - Standard

Grade: [3]

Since the case study models have not been fully developed then comment will be reserved until the final year. However modelling should be no worse than existing methods to support maintainability.

5. Evaluate whether improvement of portability has been achieved.

  a) Do we have a fully functional PIM (platform independent model)? (U,X)
  b) What is the effort to transfer to a PSM (platform specific model)? (C,U, X)

The behavioural aspects for a reduced Platform Specific Model has been explored in the pilot study by development of the dual sensor model. However the scaling up of this behaviour for a larger case has not been considered.

Outside of the pilot study the feature work on the Generic model has been contributing to the Platform independent model. Two out of four features have been developed i.e. the fault detection feature and perturbation tolerance feature. This can be said to be contributing to 50% of the abstract PIM at an abstract level.

The requirements manager tool which has been developed in response to year one work is contributing towards generating a Platform Specific Model by its instantiation of the model to a specific (if static) system. It is about 50% complete.

The work on the FMS template by Aabo can also be viewed as contributing to behaviour for a Platform Specific Model and is being considered for collaboration work to contribute to the generic model.

RODIN Grade - Standard

Since the case study models have not yet been fully developed then comment will be reserved until the final year. However any modelling should be no worse than existing methods to support portability.

Goal 6: Evaluate the improvement to the UML-B method.

Since the new UML_B method has not been available, assessment has been to justify the UML_B approach.

The pilot study did not specifically address modelling the sensor in UML-B. Some pre-pilot study work established that the UML-B action language was not yet mature to be used by a novice. Principally the action language required further development because its use often relied on an understanding of it translation to B.

The pilot study evaluated the rigorous verification aspect of the UML-B method using the formal tool set and a B modelling approach. Consideration of the generic UML-B model and investigation into UML methods has produced observations about the benefits of UML-B over B.

Justification for use of UML-B in case

UML-B benefits over B

1. Graphic visualisation rather than text.

The structure of the generic model can be seen clearly from a graphic representation of the model depicted by the class diagram. This helps the novice to view the architecture of the model more clearly. Some functionality can also be implied by the structure though the detail and sequencing of events would require the modeller to read the detail of the methods. The mapping of functional requirements to a graphic architecture should help to reduce the semantic gap.

2. The use of a higher domain language Micro B without the need to convert to B is of benefit to the non B specialist who is familiar with a dot notation for manipulating objects. The Micro B language will be evaluated in the final year when it has been updated for the RODIN platform.

3. Potential for reuse through its focus on instantiating OO architecture.

This has not been evaluated, issues concerning re-use of components are to be considered in the final year.

4. Greater Industrial acceptance because of familiarity with UML.

It was easier to communicate the structure of the generic model to other industrial users than the B model. However there is a danger that the similarities of UML can lead to ambiguity where similar terms have different meanings e.g. refinement in UML means extending functionality unlike in UML-B where formal refinement refers to constraining behaviour.

UML-B benefits over UML

Of most significance is the adherence to rigour by its application of the B method. The benefits gained from this rigour has been discussed in goals 1 and 2. Model development in UML does not provide the same degree of rigour as changes to the model are not controlled through formal refinements.

UML-B benefits over traditional approaches

The benefits of modelling over traditional approaches have been outlined in the response to goal 1.The benefits of UML-B modelling over B and UML have been addressed above.

Areas of modelling with UML-B that could be strengthened

From the Pilot study and investigation into UML some improvements to the UMLB method and tools could be made for the industrial user are given below;-

Modelling Method

c) A Novice would benefit from guidance on refinement and modelling techniques.

- Using other UML views might improve the visualisation of behaviour in the UML-B model, in particular the use of sequence diagrams could give some structure to events. The identification of use cases may also help analysis. Of course some UML views could be used independently of UML-B to document the UML-B model rather than be part of it. However, if the UML-B model is to be used to provide a complete specification, then integrating some views could reduce the semantic gap between the requirements and the model.

d) Traceability of system requirements though the model to design need to be clearly identifiable to assess coverage.

e) The pilot study identified the need for good version control of the model as it was easy to develop refinements but difficult to track changes especially where a model was reworked.

Modelling Tools

- ProB

The animation facility is very useful to the novice but considerations to extend its functionality should be made. One idea identified from the pilot was to automate test scripts so that test repeatability could be made in order to provide assurance that an altered abstractment/refinement behaviour was still valid. This could even include an automated pass/fail result summary. Another observation was that after executing an event in the animator it was not always easy to spot what variable states had changed. If the tool highlighted changed variables then this would be helpful.

f) Click'n'Prove

I found the interface of the prover was not very intuitive to use and felt it would benefit from some help information on the acronyms and illustration how each command should be applied. Interactive proofs were particularly difficult as a novice. Perhaps a training issue but a clearer understanding of how to select items to build a proof and how to interpret the output would be of great benefit.

g) UML_B platform

The notification of errors detected by the B tools should be identifiable in the UML-B notation on the new platform so that the developer does not need to be so familiar with B to change the UML-B model.

The final year will evaluate UML-B on the new RODIN platform which will include respect to the sub goals.

1. **Are reusable elements easy to create? (O)**
2. **Are the reusable elements easy to reuse? (O, X)**
3. **How is validation affected? (O)**

RODIN Grade – Good

Grade: [4]

The grading reflects the justification for UML-B in future products over UML and B. The improvements provided by the new UML-B on the new RODIN platform will be assessed when it is available in the final year.

5.2.4 Future Evaluation

The final year evaluation is intended to focus on the case study models developed in the second year, particular their modifiability and portability. The new UML-B plug-in and context manager tool will be assessed.

5.2.5 Conclusion

The evaluation of the technology by ATEC in year two has consisted mainly of its experience with the methods and tools used during the pilot study. The late availability of the new prover and the availability of the new UML_B tool has meant they have not been included in this assessment. The technology was assessed by concentrating on how useful and useable the current technology has been to the novice in meeting its evaluation goals. The technology has shown a perceived improvement over existing methods where it has been applied in the software lifecycle. However the novice would benefit from training and stronger guidelines to use the technology effectively in future products.

5.3 CS3 – Formal Techniques within an MDA Context

5.3.1 Introduction

The case study is progressing with the formalisation of various subsets of the MITA platform and the formalisation of the infrastructure and techniques to allow MDA to be used in a more formal manner.

5.3.2 Current Status

The primary driver so far for this case study has been the NoTA project within Nokia. This project comes to an end in December 2006 to be followed by a series of concurrent follow-up projects focussing on various aspects including the productisation of the technologies developed within NoTA.

Further work within Rodin will take place in a continuation project focussing on the nature and composition of services within a NoTA and extended framework including product line development in this environment.

5.3.3 Progress

Work in Case Study 3 has progressed as expected during the previous year. More description is made in the section on the evaluation of this project.

5.3.4 Metrics

The six metrics are described in document

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 EventB are not based on the ideas of object orientation and thus admit different ideas to OO. However the use of the ubiquitous UML leads to the situations where we need to map OO ideas to B/EventB and the problem here is the amount of B/EventB 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 EventB due to the necessity of using ProB which as yet does not support EventB.

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?

We have not explored this in detail at this moment, as overconstrainment seems to occur with the composition of features and other factors related to product line development. This will be the focus of next year's work.

Grade: [2]

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

Again related to the previous point much of this will be explored during the next year.

We have preliminary results regarding requirements change and 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]

5.3.5 Future Evaluation

Further evaluation will concentrate on the use of formal methods in product line development. We will focus primarily on the latter two points of the aforementioned metrics.

5.3.6 Conclusion

The development of the case study is on track with meeting its objectives and expected results.

Publication of these results has been made as part of the Dagstuhl Workshop on Formal Methods, Nordic Workshop on UML (keynote talk) and a short paper at B2007's industrial track.

5.4 CS4 – CDIS Air Traffic Display Information System

This case study will 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 will be the first test of the RODIN tool platform, as it will highlight any scalability issues that the platform might have. Once the specification can be accepted by the RODIN tool, the secondary test will be the degree of analysis that is possible for the specification.

The case study is using Event-B extensively, with part of the original specification being redeveloped in Event-B. In year 1 a substantial subset of the original CDIS requirements were identified. During Year 2 an Event-B development of this CDIS subset was undertaken. Most of the functionality of the subset has now been incorporated into the Event-B development. Some elements of the distributed design have been incorporated in the development and this will be completed in Year 3. All Event-B models in the Year 2 CDIS development were checked and verified using the B4free tool. During Year 3 the development will be ported to the RODIN environment to help with evaluation of the environment.

Assessment of Year 2 Development

A very different methodology and modelling style was adopted in the Event-B development than in the original VVSL development (VVSL is a variant of VDM). The original VVSL development produced a single large specification that was difficult to comprehend and impossible to reason about using the technology available at the time. In the Event-B development of CDIS in Year 2 we have focused our effort on tackling the comprehensibility issue and the issue of mechanical proof. We quickly found that both these issues could be tackled by using refinement to layer in the functionality of the system in series of steps rather than trying to model all the functionality in one large specification.

We proceed by addressing some more specific questions that were posed at the start of the Event-B development.

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

The layered development helped the comprehensibility considerably because we were able to capture the essential functionality of the system in the abstract specification. The abstract model is just under 3 pages of Event-B and we claim that this abstract model allows the reader to quickly grasp the essence of the system. Four subsequent refinements were used to introduce additional features of the system. The nature of these refinements was that they added additional details to the information structures and placed further constraints on when various actions could happen. The layered nature of their introduction means they can be absorbed in a stepwise fashion thus easing comprehensibility.

Grade: [4]

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 main Event-B development in Year 2 represents about 5 months of effort. This includes time to read and understand the original requirements and the original specification. It includes time spent learning to use B4free efficiently as well as time spent checking the models and performing the proofs using B4free. It also includes time spent clarifying some issues with the system with Anthony Hall (Praxis) half way 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.

Grade: [4]

3. *To what extent has the concurrent design of the system emerged from the specification-to-design refinement, and how much was this supported by the tool? (C, U)*

The main development in Year 2 was of an idealized system where information updates happen atomically. The concurrent design was not addressed for this idealised development. We also developed a functionally simplified version which introduced the concurrent architecture as a refinement. This was a pilot to experiment with refining to a distributed architecture. By modifying the specification to make explicit the asynchronous nature of the updates we were able to perform an Event-B refinement to a distributed architecture. Furthermore, we were able to check and prove the pilot using B4free. In Year 3, the approach taken in the pilot to dealing with distribution will be applied to the main Event-B development so we expect to have a more detailed assessment of the concurrency issue next year.

Grade: [2]

4. *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 language of Event-B and VVSL 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 to layer in details of the functionality, that led to a more comprehensible specification. The layered approach, along with the powerful B4free tool, made it possible to mechanically check and prove the models.

It is worth emphasising that the CDIS specification is necessarily complicated. Even though the core specification has been criticised 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.

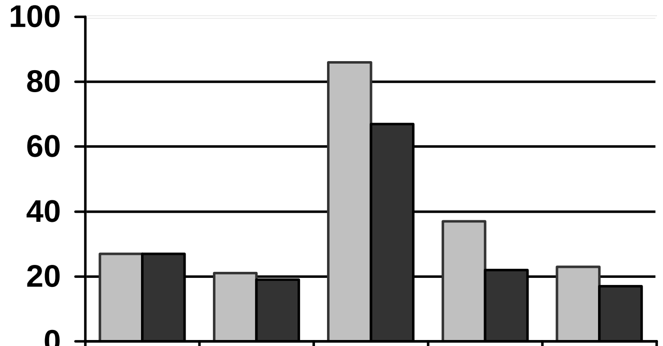
Too much complexity also precludes formal analysis. In order to reason about a specification formally, it is necessary to keep the level of detail as simple as possible. Otherwise mathematical proof becomes infeasible. Analysing monolithic specifications such as the CDIS core specification would be beyond the capabilities of contemporary formal methods tools without intense human intervention. This was not an issue during the original CDIS development because tool support was largely unavailable, and large-scale formal analysis was out of the question.

Grade: [4]

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

All proofs were carried out using the B4free tool and we found B4free to be a powerful prover. Although B4free doesn't support Event-B fully it is possible to mimic many of the features of Event-B through some hand crafting. 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. Figure 1 shows the proof statistics for the development in B4free. In this figure, L0 is the most abstract specification and L4 is the 4th refinement. The table shows the total number of proof obligations (POs) at each level and the number of those POs that were proved completely automatically. All those POs not proved automatically were proved using the interactive prover. As can be seen, a very high degree of automation was achieved in the proof.

Figure 1 Proof statistics for CDIS in B4free



	L0	L1	L2	L3	L4
▨ Number of POs	27	21	86	37	23
■ Auto proven	27	19	67	22	17

Grade: [4]

6. *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?*

The RODIN tools will be assessed in Year 3.

Grade: [N/A]

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

So far no errors or inconsistencies were found in the original specification. Some missing features were identified but these omissions turned out to be a result of the sub setting exercise in Year 1.

Grade: [2]

5.5 CS5 – Ambient Campus Assessment

5.5.1 Introduction

This case study aims at identifying 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, will necessarily generate a variety of transmission errors leading to a whole range of critical faults that must be tolerated. Moreover, mobile applications will inevitably require dealing with a variety of abnormal and unpredictable events due to system openness, mobility of its participants and their dynamic nature.
The overall project work on the Ambient Campus case study is focusing on:

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

More specifically, in this case study we are investigating how to use formal methods combined with advanced fault tolerance techniques in developing highly dependable *Ambient Intelligence* (AmI) applications. In particular, we are 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.

5.5.2 Current Status

At month 24, the status of this workpackage is as follows (see D18 for more details):

- **Methodology:** We developed a framework called CAMA (Context-Aware Mobile Agents, see below), which consists of a set of fundamental abstractions being used in formal development of ambient systems, in verification of properties of their models and in implementation of these systems. We formally developed a distributed CAMA middleware supporting these abstractions. We used Event B method supported by AtelierB. This was an important development as we firstly demonstrated the applicability of the RODIN methods to developing such types of distributed environments, secondly showed how such systems can be developed starting with a well defined set of abstractions and thirdly ensured in some degree the correctness of our middleware.

CAMA supports execution of large-scale open agent systems. To facilitate the construction of such systems, we have developed a methodology based on the application of formal refinement methods. The proposed approach addresses a number of challenges, such as interoperability, decentralised development and code reusability. We are applying these patterns now in developing the first scenario - the Ambient Lecture.

- **CAMA abstractions:** 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 programmer's work more productive and less error-prone, the programming and execution environments need to provide adequate support to it. To support exception handling, CAMA introduces inter-agent exception propagation.

- **Plugins:** We investigated two process algebras for mobility, viz. pi-calculus and Klaim. They represent both synchronous and asynchronous models of distributed systems computation. We developed semantics-preserving translations of both process algebras into suitable classes of high-level Petri nets. The work on translating pi-calculus resulted in two translations: one for finite pi-calculus terms, and recently for general recursive expressions. The translation of Klaim was based on some key ideas of our work on the pi-calculus. The developed tool is suitable for the task of model-checking of finite pi-calculus terms. The experiments conducted so far are highly encouraging and seem to confirm our initial hypothesis that unfolding based model checking is a promising verification technique for the verification mobile computing systems.

5.5.3 Progress

During the second year, our work has been focused on the following major subtask (see Project Description of Work):

- **T1.5.4.** Investigate the use of a refinement-based approach to develop a chosen part of the system. Investigate problems specific to model checking based verification of ambient applications.

We have been mainly working on the first scenario - the Ambient Lecture scenario. The following strands of work have been carried out:

- development of the decomposition and refinement patterns to be used for stepwise rigorous design of complex fault tolerant AmI systems using the B method and the RODIN platform (T2.1, T2.3, T2.4)
- stepwise rigorous development of the fault tolerance distributed AmI middleware using the B method (T2.1, T2.3)

- development of a method for modelling and model-checking AmI systems and application using a prototype of the mobility plugin (the standalone tools) for model checking fault tolerance properties (T2.4, T4.2)
- design and implementation of the lecture scenario demonstrator including the CAMA middleware and a lecture scenario application.

5.5.4 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)

Grade: [4]

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). Furthermore, we applied 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 2. In particular, this allowed us to address in more rigorous way some key aspects of fault tolerance by introducing error recovery at the early stages of design.

In our work we are developing 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.

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)

Grade: [4]

We were able to capture those aspects of ISTAG scenarios which 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 4, although we clearly do not have answers to all technological challenges raised there. 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 CAMA framework to ensure efficient cooperation while preserving de-coupling of actors, as well as system openness. Scenarios 1 and 3 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 and with the assistance of the RODIN method is an important contribution to the understanding and refining ISTAG scenarios.

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

   Grade: [3]

There is clearly a 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 which have much more effective recovery conducted at the application level. Formal reasoning about fault tolerance properties, based on refinement and verification, is a distinct characteristic of our approach as none of the existing methods supports this. We are working on a formalisation of an exception propagation mechanism that is to be integrated into the formal agent design process.

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)

   Grade: [3]

The effort put into 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 an ability to informally describe relevant decisions about a system design while doing formal modelling and development using the refinement method.

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)

   Grade: [3]

At the present moment we have provided a preliminary implementation of the proposed error recovery mechanisms and applied it 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.

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

Grade: [4]

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 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 done using AtelierB, except for about dozen of interactive proofs, all were discharged by the prover automatically.

5.5.5 Future Evaluation

Our main work in the third year will focus on finalising the CAMA modelling notations, conducting Ambient Campus Group Project experiments, developing the second Ambient Campus scenario (most likely, supporting distributed student group work), gaining extensive experience in using the RODIN platform and the mobility plug-in, capturing this experience in developing Ambient Campus scenarios and extracting it in the forms of reusable development patterns and on developing the final RODIN demonstration. In addition, we will then be in a position to fully evaluate the four ISTAG scenarios in the context of the methods and tools developed within the Ambient Campus case study.

5.5.6 Conclusion

During the second year we have been mainly working on the first Ambient Campus scenario, namely the *Ambient Lecture* (AmL) scenario. This has provided us with a valuable experience, In particular, in the areas of requirements analysis and fault tolerance for complex mobile applications. We have also made crucial decisions concerning the hardware and software platforms on which the various Ambient Campus scenarios will be implemented. We feel that the work has so far progressed according to our earlier plans and so we expect a successful outcome of the Ambient Campus case study.

6 Open Tool Kernel Assessment

6.1 Introduction

This section assesses Work Package 3: Open Tool Kernel.

6.2 Current Status

At month 24, the status of this Work Package is as follows:

- Major technical decisions concerning the platform container and plug-in mechanism have been taken and recorded in Deliverable D3.1: Final Decisions.
- The Event-B language definition has been finalised in Deliverable D3.2: Event-B Language.
- The overall architecture of the platform, the plug-in mechanism and the Event-B kernel tools have been specified in Deliverable D3.3: Specification of basic tools and platform.
- A prototype of the platform and Event-B kernel tools have been developed and delivered in Deliverable D3.4: Prototype of basic tools and platform.
- Implementation of the platform and the Event-B kernel tools is on-going. An internal version is expected for month 30.

6.3 Progress

The progress of this work package is on schedule with respect to the objectives in the Description of Work. Notably, the following milestones have been met on schedule:

- M3.1: Complete Definition of the Event-B Language,
- M3.2: Prototype platform.

The only discrepancy with what was expected from the DoW concerns the static checker (task 3.2). Initially, a mere adaptation of Atelier B tools was considered. However, changes in the definition of the mathematical language (in order to improve ease of use) and the need to have these tools implemented in the Java language (instead of C++ and theory language) for easier integration in the Eclipse Platform, led to a full reimplementation of these tools. This change did not have any impact on the RODIN schedule.

6.4 Metrics

6.4.1 Requirements and Functionality

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

Grade: [3]

As the platform and kernel tools are not fully implemented, they do not yet satisfy all of their requirements. However, the part which has been implemented works as expected. We plan to reach a grade of 5 at the end of the project.

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

Grade: [5]

Every kernel tool behaviour is formally defined. 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.

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

Grade: [5]

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

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

Grade: [N/A]

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

6.4.2 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)

Grade: [N/A]

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

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

Grade: [3]

The current version of the tools is reasonably efficient and small to medium size specifications. We're reasonably confident that this will still be the case for large specifications and full-fledged tools.

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

Grade: [5]

One of the major requirement when designing the tools was to take into account the need for incremental development of models. As a consequence, the tools perform very well on small modelling changes.

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

Grade: [4]

The tool runs on any standard PC, without any need for some fancy hardware or software.

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

Grade: [5]

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

6.4.3 Development and Integrity of the Tool

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

Grade: [4]

Most of the bugs found in the tool during development where fixed in less than a day. One of the reason for the easiness of fixing bugs is due to the fact that kernel tools always provide a trace relating their output to their input. Hence, locating the bug is made much more easy.

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

Grade: [5]

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.

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

Grade: [5]

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.

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

Grade: [4]

The tools are developed in Java with 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 know 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 very standard tool.

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

Grade: [4]

Being based on the Eclipse platform and developed in Java, the tools will be available on all standard platforms, namely Windows, Mac OS X, Linux, Solaris, AIX, HP-UX. However, due to the lack of testing platforms, it is tested only on a subset of those. Currently the tools run on three major computers: PC/Windows, PC/Linux and Mac/MacOS X.

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

Grade: [4]

Some unit tests are done for delicate parts of the tools (computation intensive parts). Extensive functional and system testing has been developed for the Event-B tools. An area for improvement is testing of the user-interface, which has not been done yet (this is because the user interface is developed in a very exploratory way and is not yet stable enough to be tested).

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

Grade: [5]

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 two compare two versions of the tools.

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

Grade: [N/A]

The kernel tools do not admit self-analysis. They allow to 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.

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

Grade: [5]

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.

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

Grade: [N/A]

It is currently too early to apply an analysis to the fault detection history.

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

Grade: [4]

The tools have been developed in a quite defensive way, so that they do 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.

6.4.4 Input (source) Language Issues

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

Grade: [3]

The current version of the tools doesn't support fully the event-B language. We're working on improving that by extending the tools. It's expected that at the end of the RODIN project full support will be provided.

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)

Grade: [N/A]

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

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

Grade: [5]

Source documents can be analysed at any point in time.

6.4.5 Output Format

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

Grade: [5]

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.

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

Grade: [5]

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.

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

Grade: [5]

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..

7 Plug-in Assessments

RODIN is developing a collection of plug-in tools to be integrated in the RODIN platform. In Year 2, the plug-in work focused on prototyping of tools and integration of tools with the RODIN platform. Between months 18 and 24, further development of the tools continued with a special focus on integration with the RODIN platform. U2B, Brama, Compsys and B2Rodin have achieved a good degree of integration with the platform (B2Rodin is a tool for converting Classical B models to Event-B models). For example, U2B can run alongside the RODIN tools in Eclipse. It allows the construction of UML-B models from which it generates Event-B models which can then be analysed by the RODIN static checker and proof obligation generator. An Eclipse version of ProB has been developed by Michael Leuschel and his team at Düsseldorf and this is being extended so that it can read RODIN models directly. Work on a model-based testing tool and the Mobility checker also continued during Year 2.

This deliverable contains assessments of 4 plug-in tools that made considerable progress during Year 2: the mobility checker, the ProB model checker and animator, the B2J code generator and the Brama graphical animator.

Previous standalone versions of the U2B plug-in for linking UML and B, and the Composys document generation system for B were assessed in Year 1 as part of D14. Work in Year 2 on these tools has focused on integration of the tools with the RODIN platform. Prototypes of integrated version of both these tools exist but they are not sufficiently functional or robust to be assessed in year 2 and will therefore be assessed in year 3 in conjunction with the case studies. Also in Year 2 ProB was extended to support test case generation from B models.

Work has started on using this extension of ProB to produce a model based testing plug-in for the RODIN environment. We expect ot have a prototype in early 2007 that will then be assessed in conjunction with the case studies.

7.1 Mobility Model Checker

7.1.1 Introduction

Mobile systems are highly concurrent thus causing a state space explosion during model checking. One should therefore use approach which alleviates this problem; in our case, based on partial order semantics of concurrency and the corresponding Petri net unfoldings [24]. 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 [23], 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 usually exhibit a lot of concurrency, their unfolding prefixes are often much more compact than the corresponding state graphs.
There exist several programming notations and frameworks proposed in the past to formally model and analyse the locality and movement of components, such as data and code. They are often based on, or directly inspired by, the π-calculus [25], which in itself is an extension of the CCS process algebra used in a variety of situations where reasoning about communication among distributed components is needed. As a result, π-calculus plays a foundational role in the continuing development of theories and methods for mobile and dynamically reconfigurable computing systems. Within RODIN, the original plan was to develop a model checker based on Petri net unfoldings aimed at verifying π-calculus specifications.

The work on the Petri net based model checking was planned to be conducted in close cooperation with the RODIN Ambient Campus case study. As a result, due to the technical decisions made in the design and implementation of latter, the scope of the Petri net based model checking has been extended to cover features related to asynchronous message passing (in addition to the synchronous message passing supported by the π-calculus). In concrete terms, it has been decided to add a capability of model checking designs expressed in a process algebra supporting constructs coming from by the KLAIM system [18]. The details of this extension are contained in [22]. The main architectural components of the target plug-in will remain the same, but they will be based on much more expressive input language relevant, in particular, to the development work within the Ambient Campus case study.

7.1.2 Current Status

The theoretical and algorithmic foundations of the compositional translation from the π-calculus to Petri nets, first developed for its finite fragment [19], have been extended to a full recursive variant of π-calculus [20]. Another strand of work aims at extending the previous developments to the KLAIM based process algebra [21].

Further development of algorithms needed for efficient implementation of the model-checking kernel of the mobility plug-in has been proposed in [26]. 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. Moreover, this representation is sufficiently similar to the traditional unfoldings, so that a large body of results developed for the latter can be re-used. Experimental results indicate that the proposed representation of a Petri net's behaviour alleviates the state space explosion problem to a significant degree and is suitable for model checking.

For a system developer, a crucial part of any model checking approach is information about system traces leading to an error state. Moreover, in order to be useful debugging information, such a trace should be as short as possible. The paper [29] describes a new efficient method for computing the shortest violation traces in the Petri net unfolding approach.

Based on the theoretical translation described in [19] and an existing efficient unfolder and verifier [23, 30], prototype tools have been developed. It should be stressed that the development of the prototype tools was not a straightforward task. Several problems had to be addressed starting from the adjustments to the input format required by the unfolder. Other problems concerned the infinite branching in the operational semantics of π-calculus expressions, and the specific implementation of the read arcs used by the theoretical translation. The developed prototype tools are suitable for the task of model-checking of finite π-calculus terms. To lift it to the recursive (or iterative) case requires further investigation as this cannot in general be done (since the full π-calculus is Turing powerful), and so one needs to identify sufficiently rich yet still manageable fragment of the full process algebra. Detailed descriptions of the problems together with the concrete solutions we adopted in order to deal with them, as well as initial experimental results in support of our specific design choices are presented in [31]. It should be emphasized that the efficiency of the prototype model checking tools appears to be much better than that provided by the standard Mobility Workbench tool which has been developed for the π-calculus.

7.1.3 Progress

The current progress against objectives in the RODIN DoW is already significant. In particular, in addition to the progress reported a year ago, in Year 2 of the project we have successfully implemented the key functionalities of the mobility plug-in for the π-calculus which has already proved to be much more efficient than similar tools provided by the standard π-calculus tool implementation (Mobility Workbench). In general, the work progresses according to the milestones listed in the RODIN DoW.

The experimental results based on our Year 2 tool implementations indicate that, in addition to achieving the planned results stated in the RODIN DoW, the work on the mobility plug-in will exceed our initial expectations.

7.1.4 Metrics

Requirements and functionality

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

Grade: [5]

As a tool designed with the purpose of efficiently model checking π-calculus specification, it delivered experimental results which were much better (by several orders of magnitude) than those provided by the standard tool developed for the π-calculus. This confirmed our initial hypothesis that the previous advancements made using Petri net based model checking can be successfully applied to mobile systems.

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

Grade: [5]

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

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

Grade: [4]

This is an expected contribution based on the previous advancements made using Petri net based model checking.

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

Grade: [4]

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 by a significant amount, though the main benefit is in verifying correctness criteria rather than relying on, e.g., testing.

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)

Grade: [4]

The necessary time for using the tool efficiently by someone familiar with the specification language should be considered to be small. The required steps to operate the tool, after building the specification, are automatic and require minimal user interaction.

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

Grade: [4]

The main engine of the tool has a proven performance record in the model checking community and initial experiments in model checking π-calculus specifications show much better performance when compared with other 'state of the art' tools. Some further improvements are currently under investigation.

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

Grade: [4]

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

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

Grade: [5]

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.

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

Grade: [5]

The released version of the tool will be free.

Development of the tool

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

Grade: [5]

The translator code developed within the project is relatively small and well documented, and it is quite easy to identify and fix bugs. (The main engine has been stable for some years now, and changes to it would require more significant effort.)

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

Grade: [N/A]

We expect that this will be relatively fast process since the output of the model checking tool provides trace information leading to an error state; however, we do not yet have experimental data to support this.

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

Grade: [3]

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

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

Grade: [N/A]

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

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

Grade: [4]

There are currently Windows and Unix versions of the prototype tool.

Testing and verification

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

Grade: [3]

We are still in the process of testing the tool using examples developed in the context of the Ambient Campus case study. We also plan to run the tool for other, non-RODIN, benchmark examples.

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

Grade: [5]

The different versions admit the same input so comparing their relative performance is straightforward.

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

Grade: [N/A]

The tool has not been designed for this purpose.

4. How reliable is error tracking and regression testing? (U, S)

Grade: [4]

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

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

Grade: [N/A]

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

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

Grade: [3]

It is possible to detect faults in the construction of a Petri net from a given process algebra expression, such as isolated places and duplicate arcs.

Source language

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

Grade: [4]

The tool is suitable for the task of model checking of finite π-calculus terms. Support for the recursive (or iterative case) requires further investigation.

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)

Grade: [4]

The tool requires as input a well-formed π-calculus expression and any finite π-calculus expression can be converted to this format.

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

Grade: [5]

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.

Output form

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

Grade: [4]

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

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

Grade: [4]

The user has a substantial level of control over the output volume of the tool.

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

Grade: [5]

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

7.2 ProB model checking and animation tool

7.2.1 Tool and Plug-in Metrics

The open source tool and plugins that result from WP3 and WP4 will be evaluated in two respects. The first way will be in the context of the applicable case studies. In these circumstances, the evaluation criteria will be as specified in each case study. We anticipate that the tool (WP3) will be evaluated for each case study, but not all plugins will be applicable to all case studies and so the case study evaluations will be applied differently to each plugin.

The second form of evaluation will be as formal methods tools in general. For these, we apply metrics that we have developed after a study of other formal methods tools, based on the principles identified in Section 2. Our overall aims are to measure how well the tool and plugins achieve their objectives. The stated WP3 objective is to develop a kernel tool that supports formal developments in Event-B, and the stated WP4 objective is to develop a range of plugin tools to support the RODIN methodology from WP2, integrating with the WP3 kernel.

7.2.2 Measurable Properties

We the kernel and plugins should be regarded as *effective* if:

- they support development of real-world systems (**usability**);
- they are commercially attractive to developers (**cost-effectiveness**);
- they can be used effectively and run efficiently in day-to-day operations (**extensibility**, **usability**); and
- they have a rigorous definition and mode of operation (**soundness**, **openness**).

Each metric is marked with a C, U, S, O and/or X where there is a correspondence to one of the metrics objectives from section 2.3.

Requirements and Functionality

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

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

The tool can animate B formal models. It can also be used for automated consistency checking and refinement checking. The tool has been proven useful on various industrial case studies (some of which within RODIN, such as MITA from Nokia). For example, it can animate all the 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” [Abrial]. The 13th refinement has 163 operations.

Grade: [4]

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

A rigorous mathematical generation admits more rigorous testing, and possibly 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]

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

A tool is unlikely to be taken up commercially unless it provides a significant gain in system correctness, due to the cost of tool acquisition and user training.

As the various case studies have shown, ProB has successfully identified a series of errors very quickly. Due to the automated nature of the tool, very 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 the amount of system re-testing after tool application to decrease.

ProB does contain a very prelminary test case generation component; but for the moment this cannot yet be applied to realistic models. Development and testing are diminished only to the extent that the various formal models are much more likely to be correct and exhibit the desired functinality. 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)

This is perhaps the prime usability criteria; the initial decision by a development team as to whether or not to use a certain formal tool will depend on a trial use. If the tool proves too difficult to learn relatively quickly then it is unlikely to be selected.

Somebody versed in B will be able to use the tool straightaway. The tool has actually been used at various places to teach B, highlighting the fact that it can help 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 frustration will be caused to the developer.

This question is difficult to answer for this tool, due to the exponential state explosion problem (``how long is a string''). However, anecdotal evidence seems to indicate 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 more and more difficult to locate and the tool will require more and more time, and due to the exponential state explosion problem an exhaustive check may be infeasible.

Grade: [3]

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

Specifications are typically built up incrementally, and in an industrial development are usually adjusted throughout the project due to requirements change. A usable formal tool must be able to accept these changes without undue overhead in re-analysis and rework.

A new specification can be very quickly reloaded. In our new Eclipse version of the tool the parser is actually re-run 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 run-of-the-mill PC is totally adequate to animate all of the B models from the case studies.

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 is 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 licencing plan has not yet been investigated.

Grade: [4]

Development and Integrity of the Tool

The results of these metrics are liable to change significantly during the development phase of RODIN. Their main use will be in measuring the relative maturity of the tools during their development. Once the tools are released, they provide a different service; namely, measuring how usable and extendable the tools are 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 output (defining the bug), accessibility of the tool's source code (finding the bug) and the re-analyse (fixing the bug) and re-test (checking the fix) speed. Also consider whether more than one fix can be applied to the tool at once.

Most bugs are fixed within a few days, usually less.

Grade: [3]

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 developed unit testing and regression testing framework. 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.

Grade: [3]

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

This measure relates to the tool's overall design; it is hard to define a good design, but this test is one significant measure. If even minor features require substantial tool rewriting then the tool is not flexible. If major features can be added in a relatively straightforward manner then this indicates a very well-designed code structure.

Several features have been added to the tool in the past year: integration with CSP, automated refinement checking, Flash-based animation engine, symmetry reduction, support for records and other new B features. This seems to indicate 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 provide extension points and have used them ourself to extend ProB.

Grade: [3]

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

This measures the accessibility of not only 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-routining and constraint solving. For the moment mainly a single person is developing and maintaining the core (Michael Leuschel), even though a new PhD student was able to add a new datatype (free types for Z) after about 6 months of starting his PhD. Work on other components is more accessible, and various extensions have been implemented by about a dozen different people. The new Eclipse version will make 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 harder. 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. It would be possible to generate Solaris binaries (but there has been no demand). A platform-independent version is also available, which requires a valid SICStus licence, however.

Grade: [4]

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

There is still no substitution for testing. Opinions on the efficacy of unit testing vary, but functional and system tests of the tool are hard to better as a demonstration of what the tool currently does and does not do.

The Prolog source code has a custom developed unit testing and regression testing framework. The tool currently has 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 (there are for example 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).

Grade: [4]

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

Tied in 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 goes a long way here.

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, 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"?

For the moment:no. The tool is not meant for developing code; it is meant for analysing formal models. Building a formal tool of parts of the tool is not unreasonable; but modelling the core constraint solving engine of ProB in B probably lies outside the capabilities of B at the moment.

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: [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.

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 is 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 particularly 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.

95 %

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 changes required 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. The earlier that a specification can be analysed and an error detected, the cheaper the error is to fix.

The new Eclipse version analyses the specification after 10 seconds of inactivity; so very quickly !

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. If the tool output is not easy to parse then it will make this difficult. The ideal is for the output to be in a suitable standardised format e.g. XML, allowing 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). The tool also provides .dot output as well as Postscript/PDF views of the state space.

Grade: [3]

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 between 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.

The tool produces no sound 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 which exhibit the erroneous behaviour. Traces can be save to an ASCII file.

Grade: [4]

7.3 Code Generation

7.3.1 Introduction

This section assesses the Work Package 4: Code Generation - plugin.

7.3.2 Current Status

The B2J translator has been set up and experimented on small size case studies. This translator combines an event recomposer and a Java code generator.

7.3.3 Metrics

Requirements and Functionality

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

Grade: [3]

The tool allows the transformation of a set of events, into a piece of a java code, by composing events, using transformation rules, and by generating java code in accordance with formal algorithm obtained so far. For the time being, the tool is a just a standalone application, as the RODIN platform doesn't support yet refinement.

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

Grade: [3]

The tool is defined in term of transformation rules and translation schemes.

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

Grade: [3]

This tool provides an executable representation of the software part of the system that can be inserted into a system level simulator.

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

Grade: [N/A]

Not covered yet.

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)

Grade: [3]

Combining events relies on transformation rules. Using the tool is a just a matter of selecting rules and events, and verifying that resulting proof obligations are true.

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

Grade: [4]

Immediate.

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

Grade: [3]

The complete procedure should be applied again, as model modification may prevent any previously applied transformation rule to be applicable.

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

Grade: [4]

The tool runs on any standard PC, without any need for some fancy hardware or software.

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

Grade: [3]

The tool is free for project partner. Commercial plan has not been yet envisaged.

Development and Integrity of the Tool

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

Grade: [3]

The tool has been developed using the THEORY language (Prolog-like language used by AtelierB proof tools). The kernel implementing this language comes along with some debugging facilities.

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

Grade: [4]

Test cases are embedded in the source code, hence easing user side debugging.

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

Grade: [3]

This tool has been designed and implemented with extensibility in mind.

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

Grade: [3]

Experiments have been conducted recently, with various skilled and unskilled users, leading to a positive conclusion concerning use of use.

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

Grade: [4]

The tool is supported by Windows and Linux platforms.

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

Grade: [3]

Unit tests are embedded in the application, and cover all functions. Functionnal testing is based on some tests cases.

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

Grade: [4]

The tool is managed with CVS and comes along with an extensive history.

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

Grade: [N/A]

The tool does not admit self-analysis

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

Grade: [4]

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.

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

Grade: [N/A]

It is currently too early to apply an analysis to the fault detection history.

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

Grade: [4]

The tools have been developed in a quite defensive way, so that they do a lot of internal integrity checks. All these checks give rise to logging of all cases where an internal inconsistency has been detected.

Input (source) Language Issues

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

Grade: [3]

100 % event B language.

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)

Grade: [N/A]

There is no external definition of event-B, nor any other tool implementation of the notation.

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

Grade: [5]

Source documents can be analysed at any point in time.

Output Format

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

Grade: [5]

Outuput is generated java code.

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

Grade: [3]

No control is possible.

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

Grade: [4]

All transformations made by the tools (most notably proof obligation generation) are fully traced to their source.

7.4 Brama

7.4.1 Tool and Plug-in Metrics

The open source tool and plugins that result from WP3 and WP4 will be evaluated in two respects. The first way will be in the context of the applicable case studies. In these circumstances, the evaluation criteria will be as specified in each case study. We anticipate that the tool (WP3) will be evaluated for each case study, but not all plugins will be applicable to all case studies and so the case study evaluations will be applied differently to each plugin.

The second form of evaluation will be as formal methods tools in general. For these, we apply metrics that we have developed after a study of other formal methods tools, based on the principles identified in Section 2. Our overall aims are to measure how well the tool and plugins achieve their objectives. The stated WP3 objective is to develop a kernel tool that supports formal developments in Event-B, and the stated WP4 objective is to develop a range of plugin tools to support the RODIN methodology from WP2, integrating with the WP3 kernel.

7.4.2 Metrics

We the kernel and plugins should be regarded as *effective* if:

- they support development of real-world systems (**usability**);
- they are commercially attractive to developers (**cost-effectiveness**);
- they can be used effectively and run efficiently in day-to-day operations (**extensibility**, **usability**); and
- they have a rigorous definition and mode of operation (**soundness**, **openness**).

Each metric is marked with a C, U, S, O and/or X where there is a correspondence to one of the metrics objectives from section 2.3.

Requirements and Functionality

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

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

The tool can animate B formal models. The tool has been proven useful on various case studies and comes along with animation examples: bridge, press, train. It has been applied on an industrial case study (Ariane 5 launcher) for CNES. The predicate evaluator, heart of the tool, has been extensively validated as it has been embedded in an application for verifying trackside safety critical invariant data for line 13 new automatic metro in Paris (RATP).

Grade: [4]

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

A rigorous mathematical generation admits more rigorous testing, and possibly mathematical analysis of the tool's operation.

The tool animates B models according to the standard mathematical B semantics, with valuated sets and constants.

Grade: [4]

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

A tool is unlikely to be taken up commercially unless it provides a significant gain in system correctness, due to the cost of tool acquisition and user training.

Brama helps to verify a B model or a refinement column, as open and close guards are clearly displayed. That way, formal model can be checked against real system.

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 the amount of system re-testing after tool application to decrease.

The development phase of the formal models itself is diminished, as the simulation phase helps to debug and tune them more efficiently.

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)

This is perhaps the prime usability criteria; the initial decision by a development team as to whether or not to use a certain formal tool will depend on a trial use. If the tool proves too difficult to learn relatively quickly then it is unlikely to be selected.

Somebody versed in B will be able to use the tool straightaway. The tool has actually been used at various places to teach B, highlighting the fact that it can help 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 frustration will be caused to the developer.

The tool behaves satisfactorily on simulating small and medium models, with very good response time. Managing history of large models leads sometimes the tool to hang. Optimisations are being investigated.

Grade: [3]

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

Specifications are typically built up incrementally, and in an industrial development are usually adjusted throughout the project due to requirements change. A usable formal tool must be able to accept these changes without undue overhead in re-analysis and rework.

Immediate.

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 sufficient for running Brama animations. Designing animations could be more demanding (flashmx, eclipse). 512 MB is minimum memory, 1GB is recommended.

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 is 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.

The tool is free to RODIN project participants. A commercial licensing plan is been investigated.

Grade: [4]

Development and Integrity of the Tool

The results of these metrics are liable to change significantly during the development phase of RODIN. Their main use will be in measuring the relative maturity of the tools during their development. Once the tools are released, they provide a different service; namely, measuring how usable and extendable the tools are 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 output (defining the bug), accessibility of the tool's source code (finding the bug) and the re-analyse (fixing the bug) and re-test (checking the fix) speed. Also consider whether more than one fix can be applied to the tool at once.

Bugs are fixed within a few days.

Grade: [3]

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.

Brama plainly uses Eclipse test capabilities (unit testing, integration testing). Adding a new test is straightforward and requires only minutes for being setup. Debug facilities (dedicated debugger and perspective) allow to precisely track errors in the java source code.

Grade: [3]

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

This measure relates to the tool's overall design; it is hard to define a good design, but this test is one significant measure. If even minor features require substantial tool rewriting then the tool is not flexible. If major features can be added in a relatively straightforward manner then this indicates a very well-designed code structure.

Brama is plug-in oriented, as most of the eclipse platform. Adding a new feature is just adding a new plug-in, contributing to existing extension points.

Grade: [3]

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

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

1 month is required to train an engineer, to be able to maintain and extend the tool.

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 harder. 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.

Brama supports Windows and Linux platforms.

Grade: [4]

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

There is still no substitution for testing. Opinions on the efficacy of unit testing vary, but functional and system tests of the tool are hard to better as a demonstration of what the tool currently does and does not do.

The tool currently has 130 unit tests implemented. Functional testing is mainly performed while replaying existing animations.

Grade: [4]

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

Tied in 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 goes a long way here.

An history is distributed with the tool, explaining changes between the various versions. The tool is managed with CVS.

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, 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"?

The tool doesn't support self-analysis.

Grade: [0]

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.

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. A log of open issues is maintained.

Grade: [1]

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 particularly 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 verify animation replay.

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.

100 % of event B language supported by the RODIN platform.

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)

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

There is no ISO standard.

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. The earlier that a specification can be analysed and an error detected, the cheaper the error is to fix.

The new Eclipse version analyses the specification after 10 seconds of inactivity; so very quickly !

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. If the tool output is not easy to parse then it will make this difficult. The ideal is for the output to be in a suitable standardised format e.g. XML, allowing use of off-the-shelf parsers and conversion tools.

Output is easily accessible, self-contained animation.

Grade: [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 between 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.

No control is possible.

Grade: [1]

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).

Errors during animation (incorrect valuation, broken invariant) are clearly displayed.

Grade: [4]

8 References

1. Procedures for Technical Review and Assessment, RODIN deliverable D6 / D7.1, 25th May 2006.
2. Contract for specific targeted research project, Rigorous Open Development Environment for Complex Systems (RODIN), Proposal no. 511599, Annex I "Description of Work"
3. "Definitions of Case Studies and Evaluation Criteria for Case Studies", RODIN deliverable D2 / D1.1, 30th November 2004
4. "Software Engineering with Formal Metrics", Lem O. Ejiogu, pub. QED Technical Publishing Group, 1991, ISBN 0-07-707455-6
5. "Object-Oriented Design Measurement", S.A. Whitmire, pub. John Wiley & Sons Inc, October 1997, ISBN 0471134171
6. "Software Science Revisited: A Critical Analysis of the Theory and Its Empirical Support", V. Schen, S. D. Conte, H. E. Dunsmore, *IEEE Transactions on Software Engineering*, vol. SE-9, number 2, March 1983
7. "Initial Report on Case Study Developments", RODIN deliverable D8/D1.3
8. "Preliminary report on the methodology", RODIN deliverable D9/D2.1
9. "Project Annual Progress Report", RODIN deliverable D13/D6.2A
10. "Prototypes of basic tools and platform", RODIN deliverable D15/D3.4
11. "Prototype plug-in tools", RODIN deliverable D16/D4.2
12. "Intermediate report on case study developments", RODIN deliverable D18/D1.4
13. "Intermediate report on methodology", RODIN deliverable D19/D2.2
14. "Project annual progress report", RODIN deliverable. D21/D6.2B
15. Formal Model-Driven Development of Communicating Systems. Laibinis, E. Troubitsyna, S. Leppänen, J. Lilius and Q. Malik.
    Proceedings of ICFEM 2005 - 7th International Conference on Formal Engineering Methods, Nov 2005, Manchester, UK.
16. Towards a methodology for rigorous development of generic requirements C. Snook, M. Poppleton, and I. Johnson. Accepted for Workshop on Rigorous Engineering of Fault Tolerant Systems, REFT, Newcastle, 2005
17. Automated boundary testing from Z and B. In Formal Methods Europe, B. Legeard, F. Peureux, and M. Utting, 2002.
18. The KLAIM Project: Theory and Practice, L.Bettini et al.: LNCS 2874 , 2003
19. Petri Net Semantics of the Finite pi-calculus Terms. Fundamenta Informaticae, R.Devillers, H.Klaudel and M.Koutny, 2006, [Devillers'06a]
20. A Petri Translation of π-Calculus Terms. Proc. ICTAC, R.Devillers, H.Klaudel and M.Koutny, 2006, [Devillers'06b]
21. A Petri Net Semantics of a Simple Process Algebra for Mobility, Electronic Notes in Theoretical Computer Science, R.Devillers, H.Klaudel and M.Koutny, 2006), [Devillers'06c]
22. On Specification and Verification of Location-based Fault Tolerant Mobile Systems. Proc. REFT Workshop at FME05, A.Iliasov, V.Khomenko, M.Koutny and A.Romanovsky*:* 2005, [Iliasov'05]

23. Model Checking Based on Prefixes of Petri Net Unfoldings. PhD Thesis, University of Newcastle upon Tyne, V.Khomenko, 2003
24. Symbolic Model Checking: an Approach to the State Explosion Problem. PhD Thesis, Carnegie Mellon University, K.L.McMillan, 1992
25. A Calculus of Mobile Processes. Information and Computation, R.Milner, J.Parrow and D.Walker, 1992
26. Merged Processes - a New Condensed Representation of Petri Net Behaviour, In: Proceedings of CONCUR 2005 (August 2005). to appear in Lecture Notes in Computer Science. V. Khomenko, A. Kondratyev, M. Koutny and W. Vogler
27. A Petri Net Semantics of a Simple Process Algebra for Mobility. In: Proceedings of EXPRESS 2005 (August 2005). To appear in the ENTCS journal. R. Devillers, H. Klaudel, M. Koutny
28. CONCUR 2005 - Concurrency Theory 16th International Conference, CONCUR 2005, San Francisco, CA, USA, Proceedings Series: Lecture Notes in Computer Science, Vol. 3653 Abadi, Martín; de Alfaro, Luca (Eds.) 2005
29. Computing Shortest Violation Traces in Model Checking Based on Petri Net Unfoldings and SAT. Proc. REFT Workshop at FME05, V.Khomenko, 2005
30. Branching Processes of High-Level Petri Nets. Proc. TACAS, V. Khomenko and M. Koutny, 2003
31. Applying Petri Net Unfoldings for Verification of Mobile Systems. Proc. MOCA, V. Khomenko, M. Koutny and A. Niaouris: 2006