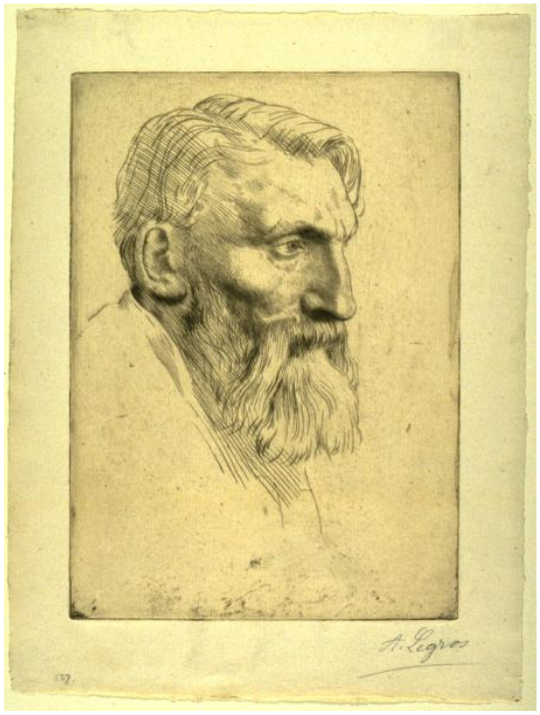
Project IST-511599
RODIN
“Rigorous Open Development Environment for Complex Systems”



RODIN Deliverable D16

Prototype Plug-in Tools

Editor: Michael Butler, University of Southampton

Public Document

28 February 2006

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

Contributors:

Michael Butler (University of Southampton)
Ledina Hido (University of Southampton)
Victor Khomenko (University of Newcastle)
Maciej Koutny (University of Newcastle)
Thierry Lecomte (ClearSy)
Apostolos Niaouris (University of Newcastle)
Martin Ross (University of Southampton)
Colin Snook (University of Southampton)
Robert Stops (University of Southampton)
François Terrier (ETH Zurich)

Table of Contents

1 Introduction

The RODIN open tools platform being developed in Workpackage 3 (WP3) will allow other parties to integrate their tools, such as model checkers and theorem provers, as plug-ins to support RODIN methods. WP4 of RODIN is developing a collection of plug-in tools to be integrated in the RODIN platform. Developing these plug-in tools has two major aims:

- To provide extra functionality on top of the core platform to support more fully the application of the RODIN methodology being developed in WP2
- To validate the open architecture of the platform by populating it with a collection of plug-in tools covering a range of functionalities.

This deliverable provides an overview of some prototype tools that have been developed by RODIN WP4 over the last six months. The deliverable consists of this overview report together with the prototype software. Our initial requirements and designs for these plug-ins where described in RODIN Deliverable D11. Since then much prototyping of tools has taken place in WP4. This has taken place in parallel with the development of the RODIN open platform in WP3. Although the developments in WP4 have been informed by the design decisions of WP4, it has not been possible at the time of writing to integrate these plug-in tools properly with the RODIN platform as the open platform prototype is being released at the same time as this deliverable (Month 18). Over the next few months WP3 and WP4 will work together on integration of the prototype plug-ins with the RODIN open platform.

Not all the plug-ins described in D11 have been prototyped at this stage. We decided to focus on a subset of the tools identified in D11. Other tools will be developed in the next stages of the project. The prototype software being delivered as part of this deliverable are

- An Eclipse-based U2B tool
- A requirements manager tool associated with U2B
- A code generation tool
- A document generation tool
- A model animation tool
- A pi-calculus to Petri-net translator

2 Eclipse-based U2B

The U2B plugin translates a UML-B specification into B. The prototype version accepts, as input, UML2[1] models, that have had the UML-B profile attached. Profiles are a feature of the UML2 metamodel which allow modelling elements to be specialised and extended with customised property fields. The UML-B profile specialises and extends the UML2

[1] UML2 is an eclipse project plugin. It is an EMF based implementation of the UML 2.0 metamodel and provides a basis for UML eclipse tools. The Eclipse UML2 Project

metamodel for UML-B modelling. We currently use Rational Software Architect (RSA) to create models for input to U2B, but any such tool could be used. U2B has no interaction with or dependency on RSA.

2.1 Tool Outline

The prototype consists of the following packages:

1. **ac.soton.umlb.u2b** The main u2b conversion package that utilises the other packages. This package also contains the plugin code and extension code (only the pop-up menu is extended).
2. **ac.soton.umlb.umlb_metamodel** A package that implements a UML-B metamodel. The UML-B metamodel is UML-like but designed to be more appropriate to UML-B modelling. The package will create an instance of itself from a UML2 model with UML-B profile.
3. **ac.soton.umlb.u2b.b_metamodel** A package that implements a B metamodel. The package will create an instance of itself given a UML-B metamodel instance.
4. **ac.soton.umlb.u2b.classic_b** A package that converts an instance of the B metamodel into a collection of B text files.
5. **ac.soton.umlb.u2b.ui** A package that can be instantiated to provide simple user interfaces. The interfaces are used by the other packages for logging progress, errors and warnings.
6. **ac.soton.umlb.preferencePage** An extension to the eclipse preferencePage facility
7. **ac.soton.umlb.preferencePage.preferences** The actual preference page code. Currently provides preferences for folder locations for B output and logfiles.

Hence the translation is performed by a constructor of the B metamodel given an instance of the UML-B metamodel that has been constructed from a UML2 model. The prototype has limited functionality. It implements the translation of UML-B structural and data elements but without many of the alternative options that are envisaged for the final version. For example state machine regions are translated into relations whereas an alternative translation to a variable for each state is planned in the final version. Some handling of textual guards and actions is provided but no uB translation is provided in the prototype. The following features are currently supported:

1. Packages are translated into B machines (containing the variable and behavioural aspects of the package) and contexts (containing the contextual sets and constant values of the package). A context will be generated automatically and referenced for each model. A package can also be explicitly identified as a context when its entire contents will be treated as sets and constants. A dependency from model to context package can be specified using a specialised dependency arrow.
2. Refinement relationships between packages can be indicated using a specialised dependency arrow.
3. Classes are translated into a set of instances and a variable representing the currently existing instances. All (non-static) elements owned by the class are lifted to the set of instances.

4. Attributes are translated into variables or constants depending on their variance property. Note that Association roles are recorded as Attributes in UML2 and hence are covered by the processing of attributes. The translation provides the appropriate type and constraint invariants for each kind of association multiplicity (see Appendix A)
5. Statemachine regions are translated into a set representing the states and a variable representing the current state. Each parallel or nested region generates a separate set and variable. No support is provided as yet for behavioural aspects.

A separate plug-in is provided to programmatically generate the UML-B profile. The program is easier to examine and maintain than the profile itself and the program enables quick installation and regeneration.

2.2 Installation and Use

Requirements:
To create UML-B models suitable for translation, an eclipse based modelling tool, such as RSA, that supports the use of UML2 profiles is needed. RSA comes in an installation package that includes eclipse and the UML2 metamodel plugin. We have found that the eclipse versions used by RSA and UML-B are incompatible. We therefore recommend installing a separate eclipse environment for running U2B. This should include eclipse 3.2.0, emf 2.2.0 and uml2 2.0.0.
To install, just unzip the profile generator plugin and the u2b plugin into your eclipse installation directory (e.g. C:\eclipse). Restart eclipse. A new menu button and a drop-down menu should appear. Use either of these to run the UML-B profile generator. The file containing the profile can be put anywhere on your file system but it is recommended that it be put in a folder outside of the UML modelling tools workspace. (This is because RSA strips the model of any profile information before exporting it if the profile is within its workspace).
To create models suitable for translation by U2B, create a class diagram model and apply the UML-B profile to it. Do not overwrite or re-create this profile once it is has been applied to a model as the model will then contain a profile application to a profile that no longer exists. (Even if the profile has the same content it contains unique identification information from its creation). The profile adds stereotypes to all existing model elements and any new elements when they are added to the model.
A UML-B model should consist of packages (representing B constructs) contained within the outer level package. Each of these construct packages may contain a class diagram and own classes. When the model is completed it can be exported from RSA into another workspace that will be accessible from the eclipse installation where U2B is installed.
To translate the model to B it must be imported into the eclipse environment (import from file system). The profile that was applied to the model should also be imported into the same eclipse project folder to ensure it is accessible to the model. First set the folder locations for the log file and for the B text output. The model can then be opened using the UML2 reflective editor (a browser style editor). Expand the model package and select the model inside it. Right click on the model to start the pop-up menu and select U2B – translate to B. U2B will ask for a project name. The project will appear in the

eclipse browser view and will contain a pair of text files (model + context) corresponding to each construct package from the UML2 model.

2.3 Platform Integration Plans

The U2B tool is intended to act as a link between the Rodin B tools and a front end UML-B modelling interface provided by a drawing tool. Our current method of inputting models is hampered by version differences between the eclipse platform required by the U2B plug-in and that required by the drawing tool.[2] This means that we have to export the UML-B model from one eclipse platform and import it into another for linking to the U2B plugin. In the future we expect these problems to be resolved by new versions of the drawing tools. We are also investigating the feasibility of providing a dedicated UML-B drawing tool that will bypass the use of UML2 and store models directly in the UML-B metamodel. Once these issues are resolved we will make U2B a model listener so that it can be invoked from the drawing tool. It will register B repository elements that are listened to by the Rodin B tools (for example syntax checker) so that these tools run automatically after U2B has updated the B models. Feedback of results will be provided by a similar process in reverse, relying on linking pointers in the model elements to identify the reverse translation paths.
We expect the packages to change in the following way

1. **ac.soton.umlb.u2b** This package will be developed to provide closer integration with alternative front end drawing tools. The package will also be expanded to accommodate additional packages handling plug-in usage facilities such as project properties, U2B help pages, improved dedicated error handling views.
2. **ac.soton.umlb.umlb_metamodel** This package will be developed to support direct data insertion. This will enable alternative (i.e. non UML2) input formats.
3. **ac.soton.umlb.u2b.b_metamodel** This package will be modified to utilise the Rodin repository for storing its B elements. This will be the main package involved in integration with the Rodin repository.
4. **ac.soton.umlb.u2b.classic_b** We expect this package to be removed after integration with the Rodin repository. A B text output facility will be provided as part of the Rodin B tools.
5. **ac.soton.umlb.u2b.ui** This package will be improved (and probably replaced by many packages) to support better user interface facilities by extension of eclipse ui features.
6. **ac.soton.umlb.preferencePage** This package is unlikely to change since it only provides the extension mechanism
7. **ac.soton.umlb.preferencePage.preferences** This package will be developed to include any new preferences as they arise.

3 Requirements Manager

The Requirements Manager (RM) plugin aids instance data management in the context of product line engineering. Product line engineering arises when multiple instances of

[2] Currently we use Rational Software Architect as this was the only available tool to support UML2. However, we intend to investigate the use of other tools such as Together's.

fundamentally indistinguishable software systems are required. RM interfaces with Rational Software Architect, an Eclipse based UML modelling tool, and provides a database for storing and validating correctness of such data. An abstract B specification of the system was produced and model-checked in ProB [HRS06a].

3.4 Tool Outline

Figure 1 shows the structure of Requirements Manager. After completing the properties, preferences, and selecting a current schema, a database can be automatically generated from the class diagram. This is then populated through bulk upload and / or manual data processing. User-friendly feedback is provided to the user through pop-up messages and the error views. When there are no multiplicity errors in the instance data, the UML-B profile can be populated using this. Finally, the diagram together with the data can be input to U2B to produce an instantiated B specification of the class diagram.

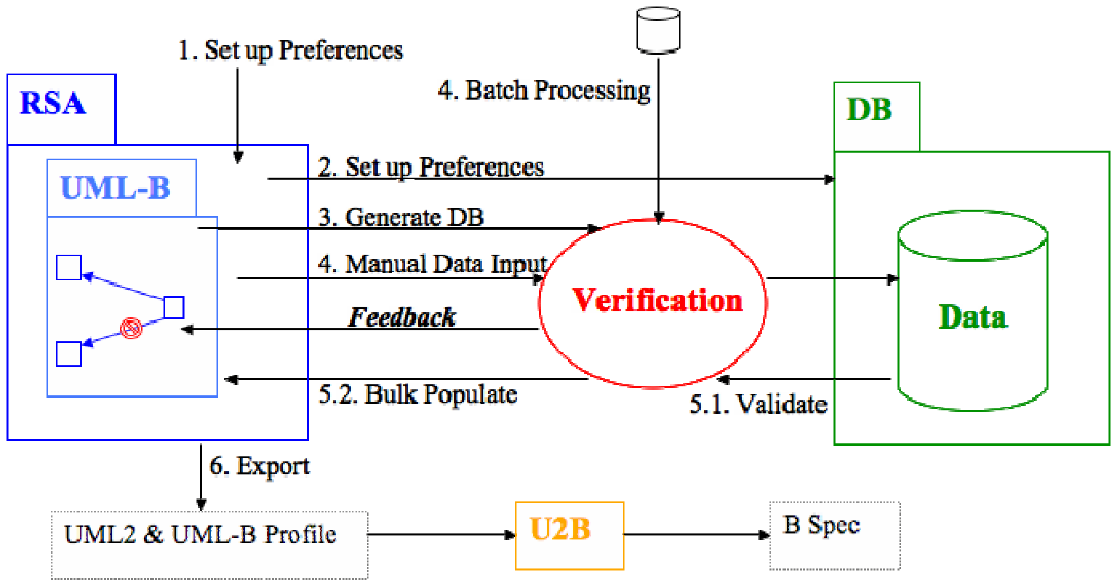


Figure 1: Tool structure

3.5 Platform Requirements

Primarily due to the requirements of RSA, the tool can only run on a Windows machine, required to have:

Processor	Minimum: Recommended:	Pentium™ III, 800MHz Pentium™ 4, 1.40GHz or higher
Memory	Minimum: Recommended:	768MB RAM 1GB RAM

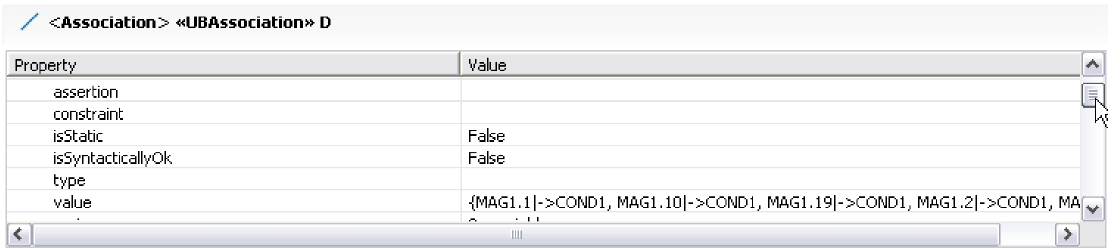
In order for the tool to work, Rational Software Architect and the PostgreSQL Database Management System need to be first installed. RSA comes in an installation package including the Eclipse platform and the UML2 metamodel plug-in. Steps on how to install PostgreSQL are detailed in the System Manual [HRS06b]. The tool also requires the UML-B profile to be applied to class diagrams (see section 2).

Details of using the tool may be found in the User Manual [HRS06c].

3.6 Platform Integration Plans

The tool has been designed and implemented to exhibit a high degree of genericity by adopting a component-based approach, using well defined interfaces and non-proprietary protocols/systems. This not only makes it easily extensible but also allows for it to re-integrated with alternative components.

The Requirement Manager tool acts as an intermediate database between the U2B tool and a front end UML-B modelling interface (currently provided through the Windows version of RSA). The data stored in the database is used to populate the *instances* and *value* fields of the UML-B profile attached to each class diagram (see Figure 2). The values of these fields, represented as sets of instances or mappings between instances can be exported from RSA together with other information from the UML diagram and imported into a separate eclipse installation where it is converted into an instantiated B Specification by U2B. In the future the Requirements Manager will be integrated into the same eclipse platform as U2B.



<Association> «UBAssociation» D

Property	Value
assertion	
constraint	
isStatic	False
isSyntacticallyOk	False
type	
value	{MAG1.1|->COND1, MAG1.10|->COND1, MAG1.19|->COND1, MAG1.2|->COND1, MA

Figure 2: UML-B Association

It would be desirable for the RM tool to be less dependent on RSA either by providing interfaces to other proprietory eclipse based UML tools or by developing a dedicated UML-B drawing tool. When porting the tool to a new drawing tool, its database component will be completely unaffected and most of the user interface components would require only minor alterations, as the RSA specific code has been deliberately isolated to single 'wrapper' classes.

It is possible that a different Database Management System could be used. In this case, the new classes handling the data can simply implement the well-defined interface between the existing database component and the rest of the tool, specifying the contract between the two. Alternatively, the database functionality can be extended, by augmenting this interface with new methods, which can then be called from within RM. Such possible extensions include providing a querying mechanism, more detailed logging or provision of user-specific views of different product lines.

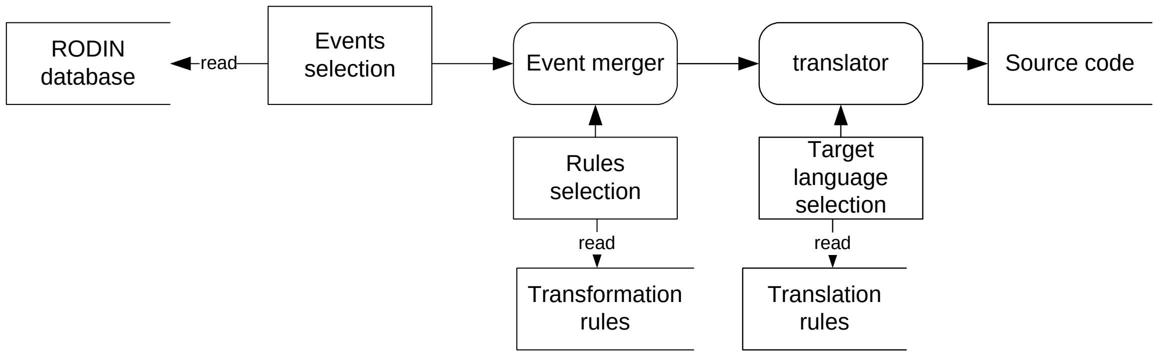
4 Code Generation

This tool helps to transform a subset of events into an a piece of code, by applying transformation (aggregation) rules on events to obtain a monolithic event and by translating this event into target source code . This tool is to be used when the software model is not large and already integrates all algorithmic aspects. The development of large software is covered by the Atelier B tool.

4.1 Tool Outline

This tool proceeds in several steps:

- The user selects the events to translate,
- The user applies a set of transformation rules to merge those events,
- The resulting merged event is translated into target source code, using simple translation rules.



Both transformation and translation rules are hard-coded in the respective implementations. Event selection is done by means of a wizard for given model. Then a list-based editor is opened , in which eventd and transformation rules are selected and applied.Finally, target translators are C Ansi, Ada and Java.

4.2 Platform Integration Plans

Integration of this tool within the platform requires to:

- connect to the RODIN database to have access to models (events, variables, …) in read-only mode.

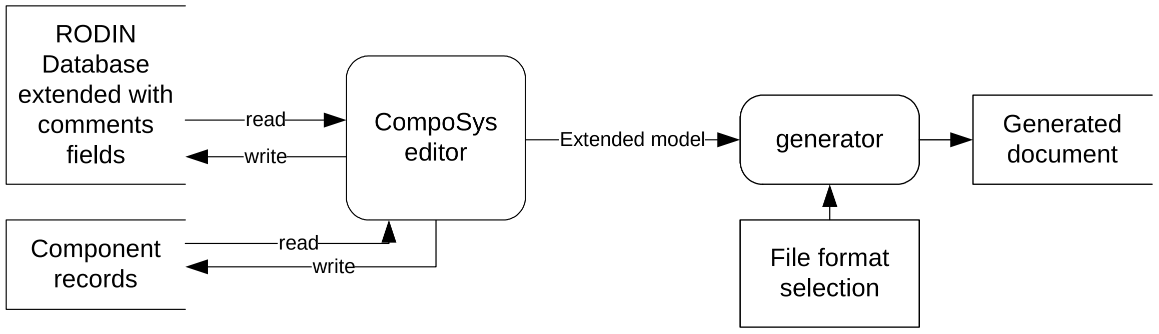
The UI is yet to be integrated in Eclipse 3.1. Their execution requires the B-kernel (the B-kernel implements the B language and comes along with Atelier B and B4free tools. This program is named krt. All Atelier B proof tools are built on top of the B-kernel).

5 Document Generation (CompoSys)

5.1 Tool Outline

This tool, named CompoSys, is designed for generating documentation for a B model, based on a component-based decomposition. The resulting document is intended to ease the understanding of the underlying B model, by providing a more representation of the underlying B model.
This tool comes along with several features:

- component based-description of a system: a system is broken down in components. Events are associated to one component.
- Comments: every single part of a model may be commented and documented.
- Verification: usage of variable is checked in order to establish in which component a variable is read and where it is modified. Variables never used or never modified after initialization are exhibited and error messages are displayed.
- Documentation generation: based on component allocation and comments, B models (up to several models within one refinement column) are transformed in chapters, one per component. Paragraphs are related to events allocated to the current component.



Supported file formats are PDF, HTML and Star Office.

5.2 Platform Integration Plans

This tool is integrated with Eclipse 3.1, but is independent from the Rodin platform. In its current state, it generates documentation from:

- B files (machine and refinement): these files are normal B files, without extra information.
- and dictionary file: this file contains comments, component description and component allocation.

AtelierB B compiler and cross-referencers are used to determine variables usage.

Integration requires to:

- connect the tool with the RODIN database, to gain access to models (variables, events, …) in read/write mode;
- extend the RODIN database with comments fields and component allocation information (events are allocated to sub-systems, composing the whole system). Separate information related to components should be stored aside the RODIN database, and synchronized with the it.

6 Animation tool

This tool brings animation capabilities to formal B model developers, easing model debug phases as well as demonstrating the B model to non B specialists.

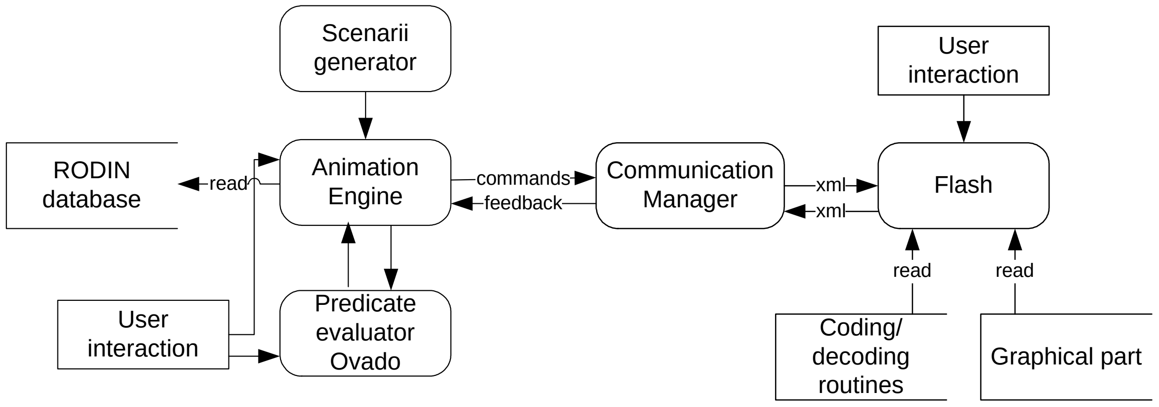
6.1 Tool Outline

The animation tool is composed of several parts:

- *Animation Engine*,
- Communication Manager,
- Graphical part, based on Flash.

The *Animation Engine* uses statically checked models from the RODIN database. From such a model, the *Animation Engine* creates its own set of objects, independent from the RODIN database, by using the Ovado predicate evaluator. For the time being, the *Animation Engine* doesn't listen to RODIN database modifications and behave independently. The *Animation Engine* is multi-threaded and can be commanded via a set of commands.
Available commands are:

- "*execute an event*" : the user may choose an event whose guard is enabled whatever the level of refinement.
- "*assign a value to a variable*": the user may assign the variables of any level of refinement.



In case of non-deterministic choices ("becomes such that" or ANY), the formula evaluator tries to find a correct value. In case of failure, the user is asked to enter a correct value.
The *Animation Engine* is also capable of generating and playing scenarios, by firing enabled events (the choice function would also be probabilistic).
The heart of the *Animation Engine* is minimal; its only function is to receive commands and to execute them. Several elements are hooked by using the provided extension points. Thus, observers (views, animation servers …) or robots (scenario generator) can be easily added.
As the *Animation Engine* is running in its own thread, the thread sending commands to it is not necessarily interrupted while a command is being evaluated. Similarly, if a command is taking too much time for execution, the *Animation Engine* may decide to abort its execution.

The Communication Manager is responsible for sending and receiving information/commands from/to Animation Engine/Flash-based graphical part. This communication part is socket-based and
Messages from Animation Engine to Graphical Part are:

- event fired,
- new variable value.

Messages from Graphical Part are:

- event played,
- new variable value displayed,
- user interaction.

The graphical part is a Flash-based animation, connected to the Communication Manager and exchanging XML flows. This graphical part is developed using FlashMX. Animations are set up independently then connected to the underlying model, by specifying specific behavior upon reception of commands. The decoding routines, transforming XML flow in Flash commands, are common to all Flash animations using the Animation Engine.

6.2 Platform Integration Plans

Integration in the RODIN platform only requires having access to statically checked models (parsing, type-checking) in read-only mode. For the time being, only unchecked models are supported as the required services are not yet implanted in the RODIN platform.
Any further functionality would be implemented as an extension to the Animation Engine plug-in, by contributing to any of its extension points.

7 Pi-Calculus to Petri Net (mobility plug-in)

As planned in the original project proposal, the mobility plug-in was to be developed and primarily evaluated in the context of Rodin's Ambient Campus case study. Following this, our work on the Petri net based model-checking has been conducted in close cooperation with this case study. In particular, it was decided about a year ago to use the Ambient Campus programming notation CAMA (context-aware mobile agents [Iliasov'05, Iliasov'05a]) as the main specification notation. At the present moment, CAMA is still under intensive development and, in fact, is expected to lead to a new programming notation which is planned to be completed in the second year of the project. As a result, the final development of the mobility plug-in will need to be postponed until this new programming notation is available. Having said that, it is clear that this notation (and so input to the mobility plug-in) will be based on concepts and constructs coming from (or being based on) π-calculus, Event B and KLAIM. We have therefore focused on those aspects of these three models which we will support in the final model-checking plug-in. In each case, the key issue which needs to be considered is a behaviour preserving translation of a given specification into a high-level Petri net, and optimisation of the resulting high-level Petri net in order to utilise all potential concurrency present in the original specification by the model-checking engine based on net unfolding [Khomenko'03].
The work on model-checking π-calculus specifications has been based on the translation of finite π-calculus terms described in [Devillers'06]. However, the theoretical translation was not suitable as a direct input to the model-checking engine as it relied on the so-called read arcs. We have therefore developed a novel technique for simulating read arcs and applied it in the implemented translation of π-calculus terms. Note that this translation is specifically aimed at full utilisation of potential concurrency in the model-checking procedure. Another advantage is that the translation utilises inherent symmetries in the analysis of state spaces of systems evolving according to π-calculus specifications. The existing translation is sufficient for bounded model-checking of π-calculus specifications, and in the next step we plan to also address some aspects of recursive (or iterative behaviour), as described in [Devillers'05a].
In the work on model-checking Event B specifications, we have developed an initial translation into high-level Petri and carried out a number of performance tests. The results are clearly promising, and in the next step we will explore the extent to which Petri net based model-checking could be used, at least for a subset of instances, to improve the efficiency of the existing Event-B model-checking.

In the work on model-checking KLAIM specifications, we proposed a theoretical translation from its recursive subset to high-level Petri nets [Devillers'05b]. As it also uses read arcs, further work is still needed to turn it into a suitable tool for producing input to our model-checking engine, and we plan to follow here our experiences gained during the work on the translation of π-calculus terms.

Further plans include incorporating recent improvements of the efficiency of unfolding based model-checking technique presented in [Khomenko'05a, Khomenko'05b].

The work on model-checking π-calculus and Event B specifications has reached the prototype implementation phase, and the folder accompanying the submission contains two corresponding sub-folders, where prototype implementations, example input files and documents explaining the main technical details and test results can be found.

7.3 Tool Outline

We provide prototype translations in the standalone format. The key components of the target Petri net based mobility plug-in will be as follows:

- *Translator* from the programming notation used in the modelling of mobile systems in the Ambient Campus case study to high-level Petri nets;
- formula *editor* where the user specifies the property to be verified;
- *unfolder* for deriving a finite prefix of the unfolding of the translated Petri net;
- *verifier* which establishes, by working with the finite prefix, whether the formula property is true of the original input.

7.4 Platform Integration Plans

We see two main ways of integrating our work with the other parts of the platform: (i) through the programming notation used in the modelling of mobile systems in the Ambient Campus; and (ii) as a verification option which can be called by a developer working with Rodin B notation (for example, through Pro-B interface).

8 Other Plug-ins

8.5 ProB model checker for B

This tools pre-dates the RODIN project though it has continued to be developed during the project. ProB is freely available from

http://www.ecs.soton.ac.uk/~mal/systems/prob.html

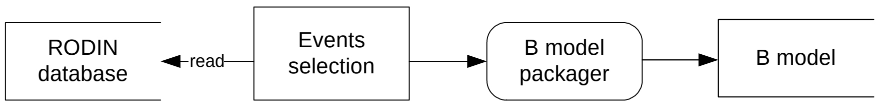
The immediate plan with ProB is to produce an Eclipse version integrated with the RODIN open platform. In this way, the animation and model checking functionality of ProB can be invoked as part of a formal development in the RODIN environment.

8.6 Model based testing

We are currently exploring a number of approaches to model-based testing in WP4. One approach that we are investigating (Nokia, Aabo Akademi and Southampton) involves combining the UML-B profile and U2B with the Confomiq UML-based testing tool. The Conformiq Test Generator automates the generation of tests from UML state diagrams (www.conformiq.com). Another approach we are investigating (Aabo Akademi, Southampton and Dusseldorf) is the use of ProB to generate finite coverage graphs form Event B models. These would then be used to generate test cases for Java programs along the lines described in [SatpathyEtAl05]. Naturally any plug-ins developed as part of this work will be integrated with the RODIN open platform.

8.7 Connection with Atelier-B

This tool provides a direct way from the system-level RODIN platform to the software-level Atelier B tool, thus enabling the development of large software in a dedicated environment. This tool is not yet complete as it is heavily dependent on having a prototype of the Rodin platform (D15) which is being made available at the same time as this deliverable.
At the end of a decomposition phase, a system-level model is usually decomposed into several modules/components/subsystems, representing hardware, software or mixed parts. This tool can be seen as an export wizard for software-only components issued from this phase. It helps to select parts, package and generate B software models.



The user is invited to select events that should be part of the specification of the software to develop. Then the tool generates the B model containing those events, renamed in operations, plus its sequencer, using different execution models: linear (single execution), looped.
Exported data are constants, properties, variables, events, invariant and initialisation.
The resulting package (Atelier B archive) is ready to be used as a starting point in a software development with Atelier B.
Integration requires having access to the RODIN database in a read-only mode.

9 Concluding

The prototype plug-ins included in this deliverable provide extra functionality on top of the core Open platform to support the goal of a rich development environment for the RODIN methodology. Our efforts over the next period in WP4 will focus on proper integration of the plug-ins with the RODIN open development environment. This will be followed by evaluation of the plug-ins in the case studies of WP1.

10 References

[Abrial96] J.-R. Abrial. *The B-Book*. Cambridge University Press, 1996

[Devillers'04] R.Devillers, H.Klaudel and M.Koutny: *Petri Net Semantics of the Finite π-Calculus.* FORTE (2004)

[Devillers'05a] R.Devillers, H.Klaudel and M.Koutny: *A Petri Translation of π-Calculus Terms.* CS-TR-887, University of Newcastle (2005)

[Devillers'05b] R.Devillers, H.Klaudel and M.Koutny: *A Petri Net Semantics of a Simple Process Algebra for Mobility.* EXPRESS (2005)

[HRS06a] Ledina Hido, Martin Ross, Robert Stops (2006) *Requirements Manager Final Report*, Master's Group Design Project, ECS, University of Southampton, UK.

[HRS06b] Ledina Hido, Martin Ross, Robert Stops (2006) *Requirements Manager System Manual*, Master's Group Design Project, ECS, University of Southampton, UK.

[HRS06c] Ledina Hido, Martin Ross, Robert Stops (2006) *Requirements Manager User Manual*, Master's Group Design Project, ECS, University of Southampton, UK.

[Iliasov'05] A.Iliasov, L.Laibinis, A.Romanovsky and E.Troubitsyna: *Towards Formal Development of Mobile Location-Based Systems*. WREFT (2005)

[Iliasov'05a] A.Iliasov,V.Khomenko, M.Koutny and A.Romanovsky*: On Specification and Verification of Location-based Fault Tolerant Mobile Systems*. WREFT (2005)

[Khomenko'03] V.Khomenko: *Model Checking Based on Prefixes of Petri Net Unfoldings.* PhD Thesis, University of Newcastle upon Tyne (2003)

[Khomenko'05a] V.Khomenko, A.Kondratyev, M.Koutny and V.Vogler: *Merged Processes - a New Condensed Representation of Petri Net Behaviour.* CONCUR (2005)

[Khomenko'05b] V.Khomenko: *Computing Shortest Violation Traces in Model Checking Based on Petri Net Unfoldings and SAT.* WREFT (2005)

[SatpathyEtAl05] Satpathy, M., Leuschel, M. and Butler, M. (2005) ProTest: An Automatic Test Environment for B Specifications. *Electronic Notes in Theoretical Computer Science* 111:pp. 113-136.

[SnookButler04] Snook, C. and Butler, M. (2004) *U2B - A tool for translating UML-B models into B*, in Mermet, J., Eds. UML-B Specification for Proven Embedded Systems Design, chapter 6. Springer.

[SnookButler06] Snook, C. and Butler, M. (2006) *UML-B: Formal modelling and design aided by UML*. ACM Transactions on Software Engineering and Methodology (to appear).

Appendix A

Translation of a UML association to B

r is the role name of an association from class A to class B, Ai and Bi are the instances sets of class A and B respectively

UML-B association properties					**B invariant**			
multiplicity	**surj**	**uniq**	**reqd**	**sing**	**additional constraints**			**type**
0..n → 0..n								`r: Ai <-> Bi`
1..n → 0..n	*				`ran(r)=Bi`			`r: Ai <-> Bi`
0..1 → 0..n		*				`r~: Bi +-> Ai`		`r: Ai <-> Bi`
1..1 → 0..n	*	*			`ran(r)=Bi`	`r~: Bi +-> Ai`		`r: Ai <-> Bi`
0..n → 1..n			*				`dom(r)=Ai`	`r: Ai <-> Bi`
1..n → 1..n	*		*		`ran(r)=Bi`		`dom(r)=Ai`	`r: Ai <-> Bi`
0..1 → 1..n		*	*			`r~: Bi +-> Ai`	`dom(r)=Ai`	`r: Ai <-> Bi`
1..1 → 1..n	*	*	*		`ran(r)=Bi`	`r~: Bi +-> Ai`	`dom(r)=Ai`	`r: Ai <-> Bi`
0..n → 0..1				*				`r: Ai +-> Bi`
1..n → 0..1	*			*	`ran(r)=Bi`			`r: Ai +-> Bi`
0..1 → 0..1		*		*		`r~: Bi +-> Ai`		`r: Ai +-> Bi`
1..1 → 0..1	*	*		*	`ran(r)=Bi`	`r~: Bi +-> Ai`		`r: Ai +-> Bi`
0..n → 1..1			*	*			`dom(r)=Ai`	`r: Ai +-> Bi`
1..n → 1..1	*		*	*	`ran(r)=Bi`		`dom(r)=Ai`	`r: Ai +-> Bi`
0..1 → 1..1		*	*	*		`r~: Bi +-> Ai`	`dom(r)=Ai`	`r: Ai +-> Bi`
1..1 → 1..1	*	*	*	*	`ran(r)=Bi`	`r~: Bi +-> Ai`	`dom(r)=Ai`	`r: Ai +-> Bi`