Event-B Support for RTOS Applications

Miguel Costa[1] and Fábio Sousa[2]
Supervisor: Leonel Braga (Altreonic)
Co-Supervisor: José Nuno Oliveira (DI Uminho)

University of Minho

Abstract. OpenComRTOS is a commercial RT operating system for embedded systems. Rodin is an Eclipse-based tool for formal modelling in Event-B. Both are the subject of a project proposed by Altreonic to MFES students of University of Minho whose main goal is to create a bidirectional connection between these two tool-sets.

1 Introduction

OpenComRTOS [1] is a commercial network-centric, formally developed real-time operating system, aimed primarily at the embedded systems market. Rodin[1] [2] stands for *Rigorous Open Development Environment for Complex Systems* and it is an Eclipse-based tool for formal modelling in Event-B.

This document reports on the work developed in a project proposed by Altreonic to the students of the Formal Methods for Software Engineering (MFES) course of the University of Minho. The main goal of this project is to create a *bidirectional connection* between OpenComRTOS Visual Designer and Rodin/Event-B [2]. The main purpose of OpenComRTOS is to provide a software runtime environment supporting a coherent and unified systems engineering methodology based on so-called *interacting entities*. In order to develop applications for this *RTOS*, Altreonic provides a tool named OpenComRTOS Visual Designer, which purpose is to allow the software developer to define the topology and the application of his system. The Rodin platform is based on Eclipse.

The subject proposed by the Altreonic company includes the exploitation of the B/Event-B language capabilities, and the integration of the latter with OpenComRTOS Visual Designer in such a way that properties of Real-time (RT) systems (such as eg. deadlocking) can be checked in Event-B. In the initial steps of the work the subject was refined to assess the correctness of system interactions, ensured by checking safety property conditions related to the interacting entities.

At the heart of this project lays the implementation of an automated process for converting OpenComRTOS Visual Designer projects onto models in the Event-B language (and vice-versa). The goal of this transformation is to assess the correctness of the system interactions, and to correct any undesired behaviour identified, thus implicitly offering a bidirectional connection between OpenComRTOS Visual Designer and Rodin.

[1] `http://www.event-b.org/install.html`

2 Introduction to OpenComRTOS Visual Designer

OpenComRTOS Visual Designer is toll designed to help the user (a software engineer) to develop and deploy real-time applications for OpenComRTOS. OpenComRTOS is a distributed RTOS and it provides a build-in router and communication layer. While hidden from the application programmer, this allows tasks to synchronise and to communicate transparently across a network of processing nodes. This provide that one node can be part of local network that is connected through internet with another cluster at the other side of the world. This supports a transparent distributed operation however is an option that does not prevent using OpenComRTOS Visual Designer on a single CPU.

Put in a simple way, a Task will be running on a computing device (CPU + RAM + Peripherals + etc.), called a "Node". There may exist many tasks running on a single node. These tasks may be run independently or may synchronise and communicate with each other.
The means of communication, synchronisation and to data exchange is provided by the OpenComRTOS through the meta-concept of hubs, each hub has specific behaviour and they are independent from the tasks.

OpenComRTOS Visual Designer supports the following types of hubs: port, event, semaphore, resource, *FIFO*, and memory pool. Each hub provides the following distinct synchronisation semantics: wait, non wait, and wait with timeout. In order to build a new application the user can simply drag and drop onto an empty canvas the hubs he needs for his application. The hubs can be later be linked to the tasks which will make use of them. The result of this process is the generation of source code with the configurations and the definitions of the functions that will be part of the application.

3 Introduction to Event-B in Rodin

Event-B is a formal method for system-level modelling and analysis. Event-B is a notation and method developed from the B-Method and it is intended to be used with an incremental style of modelling. The idea of incremental modelling has been taken from modern programming languages that come with integrated development environment that make it easy to modify and improve programs. Key features of Event-B are: the use of set theory as a modelling notation, the use of refinement to represent systems at different abstraction levels, and the use of mathematical proof to verify consistency between refinement levels.

The Rodin Platform is an Eclipse-based IDE for Event-B that provides effective support for refinement and mathematical proof. The platform is open source, contributes to the Eclipse framework and is further extendable with plug-ins. This work has been sponsored by the DEPLOY project[2].

[2] `http://www.deploy-project.eu/`

3.1 Event-B Modelling in Rodin

Every Model in Robin is composed of 2 separate components, the *context* and the *machine*. However, it is possible to use just the machine component as the main and only element of the model. The contexts describes static elements of a model, elements that will be set in each model trace executed. Machine describes the dynamic behaviour of a model by means of variables whose values are changed by events. A central aspect of modelling a machine is to prove that the machine never reaches an invalid state, i.e., the variables always have values that satisfy the invariant. Machines can also make use of the sets, constants, and axioms declared in contexts.

A context has the following components:

- Sets, using constants.
- Constants, in which each constant must be given axioms.
- Axioms, are a list of predicates. They describe what can be taken for granted when developing a model. The axioms can be later used in proofs that occur in components that use this context. Each axiom has a label attached to it.
- Theorems, Axioms can be marked as theorems. In that case, is declared that the predicate is provable by using the axioms that are written before the theorem. Theorems can be used later in proofs just like the axioms.
- Extends, a context may extend an arbitrary number of other contexts. Extending another context A has the effect that we can make use of all the constants and axioms declared in A and we can add new constants and axioms. As in object oriented programming languages.

The order of the axioms and theorems matter because the proof of a theorem or the degree to which an expression is well-defined may depend on the axioms and theorems that are already written. This is necessary to avoid circular reasoning.

An machine consists briefly of:

- Sees, can use the context's sets, constants and axioms in a machine by declaring it in the Sees section. The axioms can be used in every proof in the machine as hypothesis.
- Variables, the values of the variables are determined by an initialisation and can be changed by events. Together they constitute the state of the machine. Each variable must be given a type. This definition is achieved by the use invariants.
- Invariants, these are predicates that should be true for every reachable state. Each invariant has a label.
- Events, an event can assign new values to variables. The guards of an event specify under which conditions it might occur. The initialisation of the machine is a special case of an event which set ups the machine's initial configurations.

A good tutorial can be found here[3]

4 Development

The work was scheduled as follows. The first stage of this project wast to conduct a research on the Event-B language concerning its expressiveness power and tooling-support. Documented in 4.1. Parallel to this, work was carried out on understanding the OpenComRTOS Visual Designer tool, by modelling the behaviour of some *Hubs* (base elements found in the OpenComRTOS Visual Designer and OpenComRTOS) and correspondent connection methods to be used in the construction of small models, with no more than 3 to 4 elements. Some of these models can be found in the appendix section 8.2 and a example of a OpenComRTOS Visual Designer diagram in appendix section 8.1.

During the analysis of the models created in Visual Designer, the presence of undesired behaviour was immediately identified and described in section 4.2.

Modelling a global model containing all the OpenComRTOS Visual Designer elements in a arbitrary number was the next step taken. The step was immediately followed by automatising the process of converting OpenComRTOS Visual Designer projects onto Event-B models compatible with Rodin. The creation of this global model is described in the section 4.3 and the automation process is described in the section 4.5.

Such automatised process allows an increase and diversification on the number of cases to be studied. It allows also the evaluation of the capacity of Event-B in detecting undesired behaviour that is described in section 4.4

So that the *bidirectional connection* between OpenComRTOS Visual Designer and Rodin can be done one of the following two different paths can be chosen:

- Using the model created in Event-B and identify the problems and correct it directly in the OpenComRTOS Visual Designer.
- Using the model created in Event-B and identify the problems and correct in the Event-B model, and afterwards convert it to the C programming language to be interpreted by OpenComRTOS Visual Designer.4.6

4.1 Event-B: Syntax and Tooling-support

Event-B language is a very powerful language that relies on set theory and first order logic, such as B Language, but it uses event logic, which is what differentiates it from the B-language.

Several platforms were found during the execution of this project, namely Rodin[2], Atelier B[3] and ProB[4]. The chosen modelling tool for Event-B for this project was Rodin, for the simple fact that it is better documented in the Event-B.

[3] `http://www.atelierb.eu/outil-atelier-b/`

[4] `http://www.stups.uni-duesseldorf.de/ProB/index.php5/The_ProB_Animator_and_Model_Checker`

To learn how to use this language in the context of this tool a tutorial[5] for Event-B was followed, and a Cheat sheet[6] was kept in hand the.

4.2 Problem Analysis

From a first analyse it became clear that in each OpenComRTOS Visual Designer project, the diagrams representing the connections between tasks may contain undesired behaviour. Let us consider as a first example a diagram with four tasks and four ports using the wait synchronisation mechanism (see figure 1). As we shall soon see, in this simple example, it is possible to reach a state of deadlock.

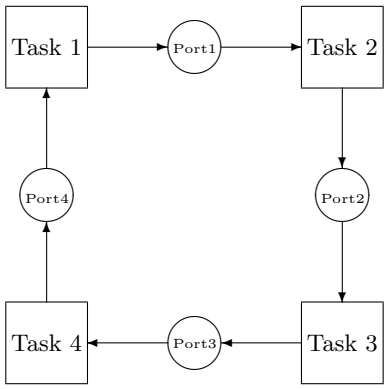


Fig. 1: Diagram with 4 Tasks and 4 Ports

Let us now suppose that the following four events occur:

- Task 1 requests to write on Port 1.
- Task 2 requests to write on Port 2.
- Task 3 requests to write on Port 3.
- Task 4 requests to write on Port 4.

As can be seen in figure 2, after the events mentioned above the system blocks because all the tasks are waiting to synchronise but none of them can continue without being unlocked.

[5] `http://handbook.event-b.org/current/files/EventB-Summary.pdf`
[6] `http://handbook.event-b.org/current/files/EventB-Summary.pdf`

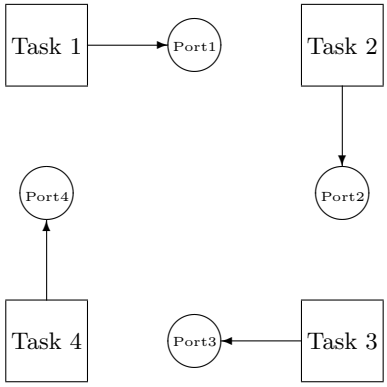


Fig. 2: System blocked - deadlock

A system that will not block does not ensure a well behaved system (such as*live lock*). In the previous example, if it is considered any type of synchronisation mechanism with the repeated behaviour such as defined in 4.2, the system will achieve a state of live lock.

4.3 Event-B Specification of OpenComRTOS Projects

OpenComRTOS Visual Designer projects contain different elements, but just some of them were the focus of our work in this project. This targeted/studied elements can be divided in three different types:

- Tasks
- Hubs
  - Port
  - Fifo
  - Semaphors
- Synchronisation mechanisms
  - Wait
  - Wait and timeout
  - No Wait

After the analysis of some models, in section 4.2, the next step was start to model the Hubs in Rodin. For all of the synchronisation mechanisms(Wait, Wait with timeout, Non wait). It was created one model for each hub, respectively Port(Tasks and Ports), Semaphore(Tasks and Semaphores.), and Fifo(tasks and Fifos). In the end, they were merged into a single Event-B model capable of representing an arbitrary number of tasks and hubs.

Hubs and Synchronisation Modelling Methods The use of models will enable a deeper analysis of the behaviour of the system. For this purpose was decided that the following concepts must be modelled:

- Representation of different connections.
- Representation of different synchronise methods.
- Representation of connected nodes.
- The effect of synchronisation methods on the tasks.

To achieve this, synchronisation methods are the most important elements in the models and the ones that need to be model more concretely as possible. Time duration of synchronisation is not important point but the order in which the system call and events happens.

Task The operations related only to tasks were abstracted and model as variables which hold the state in which the task is. That state can be:

- Idle, the task is executing something and is not trying synchronise.
- Wait, state in which the tasks wants to synchronise with a semaphore using the wait semantics.
- WaitT, state in which the task wants to synchronise with a semaphore using wait with the timeout semantics.
- Non Wait, state in which the tasks wants to synchronise with a semaphore using the non wait semantics.
- Wait_w, state in which the tasks wants to synchronise with a fifo or a port using the wait semantics to write.
- WaitTime_w, state in which the tasks wants to synchronise with a fifo or port using wait with timeout semantics to write.
- NoWait_w, state in which the tasks wants to synchronise with a fifo or port using non wait semantics to write.
- Wait_r, state in which the tasks wants to synchronise with a fifo or port using wait semantics to read.
- WaitTime_r, state in which the tasks wants to synchronise with a fifo or port using wait with timeout semantics to read.
- NoWait_r, state in which the tasks wants to synchronise with a fifo or port using non wait semantics to read.

Port Ports were model as simple connection channels. A port does not preserve order of call requests and the synchronisation needs to exist when one task wants to read and another one to write. The events that can be triggered from a task are:

- Desire to synchronise using the wait semantics is trigger by the follow methods:
  - PortReadWait, to receive information.
  - PortWriteWait, to send information.

The task that requests this method can only unblock if it is able to synchronise with other task.

- Desire to synchronise using the wait with timeout semantics is trigger by the follow methods:
  - PortReadWaitTime, to receive information.
  - PortWriteWaitTime, to send information.

  The task that requests this method can only unblock if it is able to synchronise with other task, or by the one of the following methods:
  - PortTaskTimeOutW,
  - PortTaskTimeOutR,

  These methods introduce the concept of a timeout, which indicates how much time is the caller willing to wait to achieve synchronisation.

- Desire to synchronise using non wait semantics is trigger by the follow methods:
  - PortTask_Read_NW, to receive information.
  - PortTask_Write_NW, to send information.

  The task that requests this method does not block and it will only synchronise if it exists other task waiting to synchronise. In the event of not existing tasks waiting to synchronise, the following triggers will be available.
  - PortTask_Write_NW_fail.
  - PortTask_Read_NW_fail.

  These methods represent that request has timed out.

- In every case, if the synchronisation is successful the trigger is
  - PortSync.

Fifo FIFO Hub was modelled as a bounded circular array. The first elements entering in the array are the first ones to be removed.

- Desire to synchronise using the wait semantic is trigger by the follow methods:
  - fifo_rd_w, to receive information.
  - fifo_wr_w, to send information.

  The task that requests this method can only unblock if it is able to synchronise with the FIFO. Synchronisation depends on the existence of an empty slot for a new element in a send information event, or on the existence of data in the array if trying to read information.

- Desire to synchronise using the wait with timeout semantics is trigger by the follow methods:
  - fifo_rd_wt, to receive information.
  - fifo_wr_wt, to send information.

  The task that did synchronise using one of this methods can only unblock if synchronise successful with the FIFO or by one of the follow methods
  - fifo_sync_wr_wt_fail.

  - fifo_sync_rd_wt_fail.

  These methods represent the times out of the request.

- Desire to synchronise using the non wait semantics is trigger by the follow events:
  - fifo_sync_rd_nw, to receive information.
  - fifo_sync_wr_nw, to send information.

  The task that requests this method does not block and it can only synchronise if it exists other task waiting to synchronise. If there are no tasks with which to to synchronise the following triggers that represent that request timed out will be available:
  - fifo_sync_rd_nw_fail.
  - fifo_sync_wr_nw_fail.

  These methods represent that request has timed out.

- In all cases if synchronisation is successful is trigger by
  - fifo_syn_wr_w_wt.
  - fifo_syn_rd_w_wt.

Semaphore Semaphores are simple structures with a variable, this variable is the number of increments made to that semaphore. A task can "ask" a semaphore to increment the variable or to share/give the information about the variable.

- Desire to synchronise using the wait semantics is trigger by the follow events:
  - Task_Want_2_Sync_Sem_Wait, if the semaphore is free the task just asks to synchronise;
  - Task_Sync_AfterWait_Sem_Wait, if the semaphore is busy the task just stays in queue to synchronise;
  - Task_Sync_Sem_Wait, Semaphore allows task to synchronise.

  The task that requests this method can only unblock if the synchronisation with the Semaphore is possible.

- Desire to synchronise using the wait with timeout time semantics is trigger by the follow events:
  - Task_Sync_Sem_WaitTime_FREE, if the semaphore is free the task asks to synchronise with a given timeout;
  - Task_Sync_AfterWait_Sem_WaitTime, if the semaphore is busy the task stays in queue to synchronise for a given timeout;
  - Task_Sync_Sem_WaitTime_BUSY, if the semaphore is waiting to synchronise with a task that just timed out.;

  The task that requests this method can only unblock if the synchronisation with the Semaphore is possible. If the tasks times out the following triggers will be available
  - Sem_Sync_TimedOut
  - Task_TimedOut

  These methods represent that request has timed out.

- Using the non wait semantics when tasks ask directly to the semaphores to increment or to share its variable value.
  - Sem_ShowInfo_NoWait_Free and SemIncrement_NoWait_Free, if the semaphore is free this events can occur at the moment;
  - Sem_ShowInfo_NoWait_Busy and SemIncrement_NoWait_Busy, if the semaphore is busy this events can't occur at the moment.

To synchronise the Semaphore will need to be free and after a successful synchronisation the task can ask the Semaphore to increment its variable or to share its value before.

4.4 Safe Condition

To ensure the system is well behaviour one needs to ensure that the cases such as the ones found in the section 4.2 do not happen. Its only possible to enquire such properties using Temporal Logic.

Rodin has support for Temporal Logic through the plugin "ProB". Documentation about the use of Temporal Logic in Rodin is almost nonexistent, the only reference that could be found was `http://www.stups.uni-duesseldorf.de/ProB/index.php5/LTL_Model_Checking` and it is not related to Rodin.
Initial tests show that the plug-in does not work properly in the matter of Temporal Logic and for that reason it is impossible to verify such properties. This is a big setback, blocking part of the development of this project. An new set of tools will be needed to pursue this objective.

The results that we observed when trying to check LTL properties made us wonder if the problem is related to the initialisation event. An counter-example was always popping up, an empty set implying some expression.

Before the initialisation event the variables were not assigned. So it is assumed that the error stems from that event.

Linear-time temporal logic (LTL)[7] is a modal temporal logic with modalities referring to time. In LTL, one can encode formula about the future of paths, e.g., a condition will eventually be true, a condition will be true until another fact becomes true.

Pseudo examples could be:
$G((\forall t.TaskStatus(t) == wait) ==> F(event(PortSync\ with\ t)))$, this would mean: for all states if task t enters in wait status, eventually the event of synchronisation referent to task t will happen .

4.5 From Visual Designer to Rodin

To allow the conversion of OpenComRTOS Visual Designer projects into Rodin models two programs have been created.

[7] `http://en.wikipedia.org/wiki/Linear_temporal_logic`

The main program(named *rtos2rodin.c*) is written in C code(*C Programming Language*[8]). *rtos2rodin.c* is responsible for generating the Rodin model files (contexts and machines) from OpenComRTOS Visual Designer project files.
The other program is a *bash script* file which is responsible for extracting the information about the elements(Hubs, tasks) and their connections, from the OpenComRTOS Visual Designer project files.

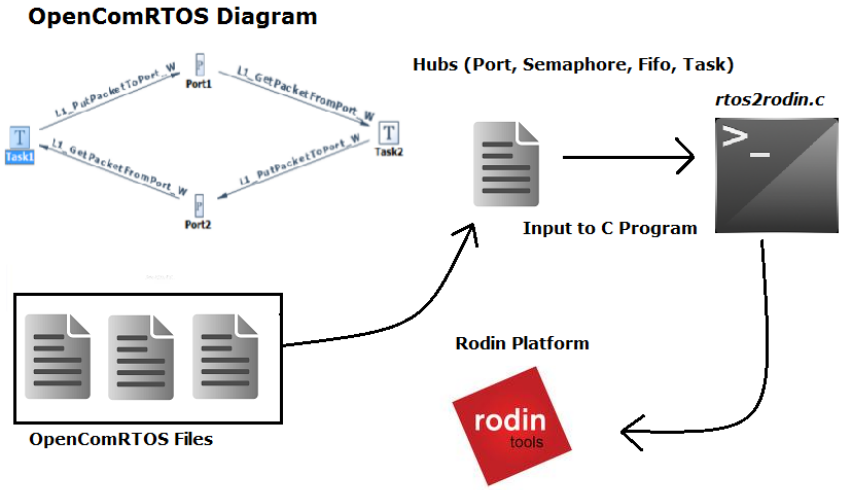


Fig. 3: Conversion Process

Extract the Data from the projects There are two different ways to extract data from a Visual Designer project:

1. Dynamic filter
2. Static filter

A *dynamic filter* would analyse every XML file from the OpenComRTOS Visual Designer project and then get the information about every element, connection and task.

A *static filter* retrieves the information directly from C files. This one is the implemented and was chosen over dynamic filter as the difficulty to parse the XML over C code would be greater for us and the information retrieved would be the same.
The implementation of this filter is based on a bash script. Using the *egrep* command, usually found in *Unix* systems. This command is a simple text filter which looks for patterns which match the system calls to hubs in C file.

[8] `http://en.wikipedia.org/wiki/C_(programming_language)`

The bash script removes also repeated information and extracts the data to individual files. This individual files contain the information of each element, task or connection in the project.

For instance, if a project contains tasks, fifos, ports, semaphores, and the tasks are connected using any of the hubs, the filter will write the following files:

- t.txt
- p.txt
- f.txt
- s.txt
- c.txt

Here is a example of a t.txt(tasks) file:

```
TaskEntryPoint1
Task2
Taskers
TaskOfDuty
```

Conversion Process After creating a project in OpenComRTOS Visual Designer some files related to that project will be created. Therefore the user should copy only the C files created to the folder where the conversion program is currently located.

The conversion program(*rtos2rodin*) can be executed in terminal as follows:

```
$ ./rtos2rodin.c
```

During the execution of the program the user will be asked to give a name to the Rodin model to be generated.

Afterwards the program will finish the conversion and a new folder will appear in the current directory, with the name given by the user. The new folder can now be imported to Rodin platform.
Once generated, the project can be imported to Rodin. Next there is a list of steps to achieve this task:

1. Run the Rodin platform executable
2. Press 'File' tab
   (a) Import
   (b) General → Existing Projects into Workspace
   (c) Browse the file
   (d) Press 'Finish'.
3. Press 'Project' tab
   (a) Select the project to clean
   (b) Press 'OK'.
4. Build the project
   (a) Right click in the project
   (b) Press 'Build Project' option.

These steps can be also followed from the figures in the appendix section 8.4.

***rtos2rodin* - Functional Behaviour** The program rtos2rodin will only work properly with the help of the filter(bash script). If the filter is missing the program will not work properly and any attempt to generated a the Rodin project will generated models without any task or element.

After a few investigations over the Rodin model files it was easy to notice that Rodin model files are written in XML(Figure 10).
With the knowledge of the XML structure of the Rodin files, devising a method to generate the model files is became simpler. With the information about the elements, tasks and connections the program rtos2rodin will create the files context.buc and machine.bum. These files contain the information about the context and the machine, respectively.
An example of a Rodin tags.

```
<org.eventb.core.guard name="*" org.eventb.core.label="grd2"
org.eventb.core.predicate="new_value = TRUE ->
peds_go = FALSE"/>
</org.eventb.core.event>
```

4.6 From Rodin to Visual Designer

The accomplishment of this task was highly dependent on the plugins available to Rodin whose goal is to generate C source code from Event B models.

Unfortunately, this point could not be achieved because of the lack of working source code generators. A further explanation on this matter will be given in section 5.2.

5 Rodin Platform Tool Evaluation

5.1 Global Aspects

Event-B is a very powerful formal modelling language, which Rodin takes leverage of it. The syntax of Rodin is simple, intuitive and allows the user to easily create complex expressions. Nevertheless, some limitations and constraints during the course of this project were found. Specifically, additional features to this tool and interactions with external tools, such as source code generation and the integration with temporal logic provided by the tool ProB.

A minor aspect, definitely not crucial, is the editor. In the early use of the editor this presents itself helpful, but shortly after the beginning of the project it becomes unusable. Simple operations like copy and paste of elements, adding new fields, need a correct clicking in specific points in the editor. Also the auto-interpretation that combines characters sometimes works erroneously, which might be counter productive. An existence of an option to interpret pure ASCII text would be interesting.
A new version of Rodin was released while creating this report.

5.2 Code Generation

One of the goals of this project was to convert Rodin models to OpenComRTOS Visual Designer projects. These projects are constituted of C source code files. Due to this reason, Rodin plug-ins[9] to generate that C code from Rodin models were the focus of our search.

Code Generators available for Rodin :

- EB2ALL[10];
- Tasking Event-B[11];
- B2C[12].

EB2ALL supports automatic source code generation from Event-B to C, C++,Java and C#.

From the official website[13], "The EB2ALL tool is still in development stage, we are releasing all the EB2C, EB2C++, EB2J and EB2C# tools as beta version before final release of the EB2ALL for collecting the bugs."web page[14]

From the list of modules provided by this plug-in, the suitable one is EB2C. The steps for the installation of this tool are provided in the section "Download" → "EB2C" → "Installing The EB2C Plug-In".[15]

We were able to install EB2C using the guidelines provided in the official documentation, but unfortunately we could not have the same success running it. An error message would pop up, as you visible in Figure 4. Several Rodin models were used with the purpose of discarding any incompatibilities with Event-B models, but the result would be always the same: an error message and no output files generated.

From an official paper[4]: "This is our first step towards source code generation from the Event-B formal specification, and our aim is to improve this tool to meet the industrial requirements".
It would be helpful to know what should be understood by "industrial requirements" in this context, but we could not get more details with regards to this matter.

Tasking EventB supports generation of multi-tasking Java, Ada, and OpenMP C code from Event-B.

We haven't found any manual for the installation of this tool and the files available on the GIT repository seem to be plug-ins that are already installed, which did not help in the installation of this feature in Rodin.

[9] `http://wiki.event-b.org/index.php/Rodin_Plug-ins`
[10] `http://eb2all.loria.fr/`
[11] `http://wiki.event-b.org/index.php/Code_Generation`
[12] `http://wiki.event-b.org/index.php/B2C_plugin`
[13] `http://eb2all.loria.fr/`
[14] `http://eb2all.loria.fr/`
[15] `http://eb2all.loria.fr/`

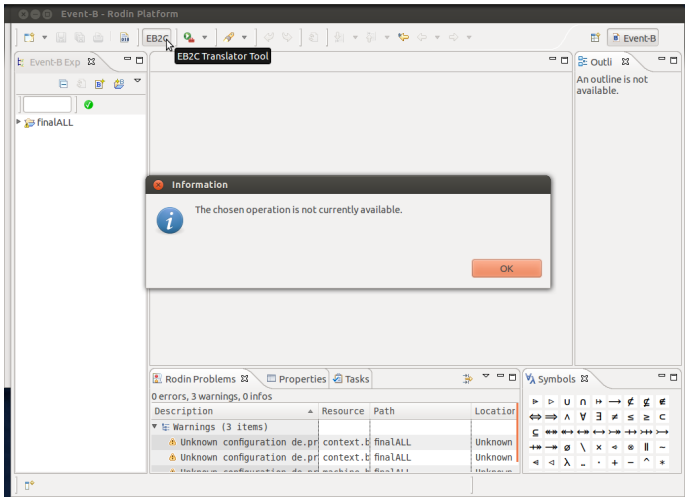


Fig. 4: Error when running EB2C tool.

B2C translates Event-B models to C source code, which may then be compiled using external C development tools [5].

By Following the manual[16] [6], the installation process became an easy process, but for a proper installation it is recommended the usage of considerable old versions of Rodin and Java.

Installing Rodin is easy but the installation of java was a bit tough and the system warned that the installation might not succeed because it was old for the computer system and that version of java would not work properly in new Microsoft Windows systems, as the tool is only available under Microsoft Windows Platforms.

Although the installation of the Rodin platform and java was completed, there was no success while running it unfortunately, as we can see in the Figure 5. The manual does not provide any other way to run this source code generation, so this tool couldn't also be used.

5.3 Choosing a source code generator

From the experience with the tools we found during this project to perform source code generation we would make the following recommendation. If a new and working version of any of these tools were released we would strongly recommend to use the EB2ALL tool for source code generation. Even knowing that the tools aren't working it's possible to analyse the method they generate the code, just by following their papers. The tool Tasking Event-B don't provide a paper with the method they generate code, so just by analysing the two remaining tools (EB2ALL and B2C) the EB2ALL tool has a easier way to generate

[16] `http://deploy-eprints.ecs.soton.ac.uk/84/2/Midas_Deploy.pdf`

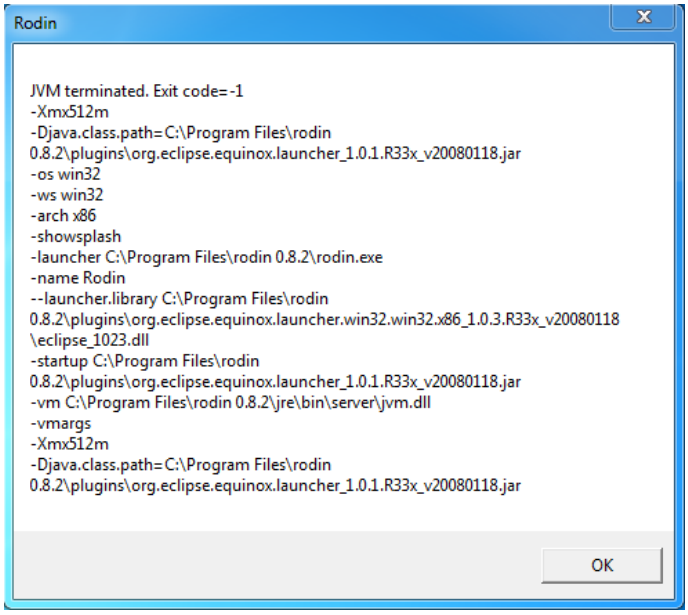


Fig. 5: Error when running Rodin.

code. This generation method can be found in their papers. The method used to generate in this code can be found in[4].

6 Useful tips and tricks

6.1 ProB tool

After downloading the Rodin platform it is highly recommended to install in the Rodin environment the tool ProB[17].

To install this tool in Rodin environment just open the "Help" tab and press "Install New Software..."(Figure 6). Afterwards type in the 'Work with' section the tool you desire to be added to Rodin(Figure 7).

These tools provide additional features to users like: the possibility to run a animation/model checking, but the most important is to check certain properties like find the existence of blocked states(deadlocks), invariant violations, etc(Figure 20).

The figures about the execution of ProB in Rodin can be found in the appendix section 8.5.

[17] `http://wiki.event-b.org/index.php/ProB`

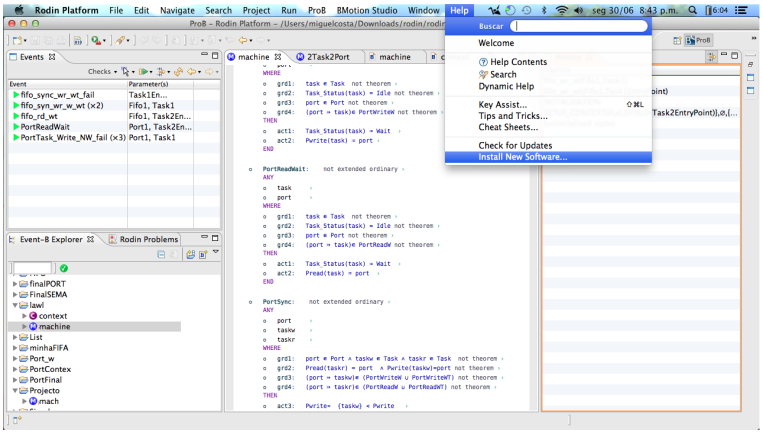


Fig. 6: Help → Install New Software

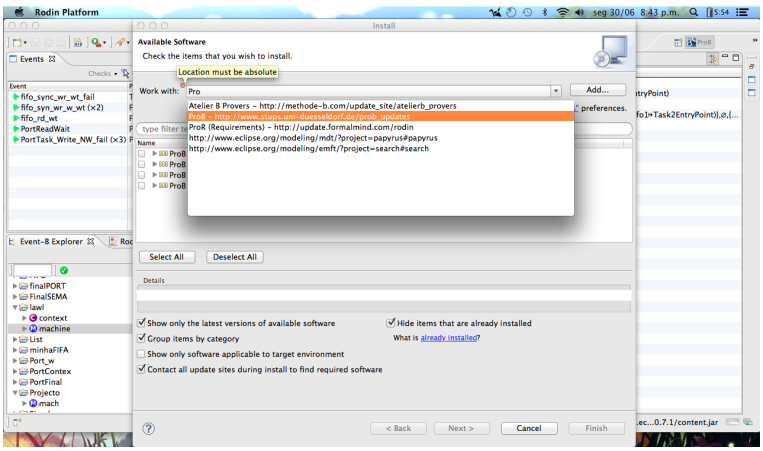


Fig. 7: Installing the new tools

7 Conclusion and Future Work

The research that was put in place to integrate OpenComRTOS Visual Designer with Rodin and Event-B led us to model the various hubs and interactions. This has allowed us in the identification of undesired behaviours. Such behaviours came from synchronisation methods and communication relation between tasks.

A set of tools have been developed to implement the bidirectional transformations between OpenComRTOS Visual Designer and Rodin with the intent to check the correctness of system although the bi-directionality was not achieved due to lack of proper tools to generate proper source code from formal models written in Event-B and the functionality to check Temporal Logic properties.

As future work will be to extend this project to the remaining Hubs that weren't modelled yet, which also implies upgrade the filter to recognise the new hubs added. Also add the feature recognise the connection names and system calls from XMLs created by OpenComRTOS Visual Designer, such feature will allow rename of connection names and system call by Altreonic without need to change the filter source code. Its also important to overcome barriers founded by the complemented tools in Rodin, even if that means change the modelling language being used so it would be possible to generate source code and check LTL properties.

References

1. Altreonic: OpenComRTOS-Suite Manual and API Manual
2. Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: an open toolset for modelling and reasoning in Event-B. STTT **12**(6) (2010) 447–466
3. Jastram, M.: Rodin User's Handbook
4. Singh, N.K.: Using Event-B for Critical Device Software Systems. Springer (2013)
5. Wright, S.: Automatic generation of c from event-b. In: Workshop on Integration of Model-based Formal Methods and Tools. (February 2009)
6. Wright, S.: MIDAS: A Formally Constructed Virtual Machine

8 Appendix

8.1 Example of a OpenComRTOS Visual Designer diagram

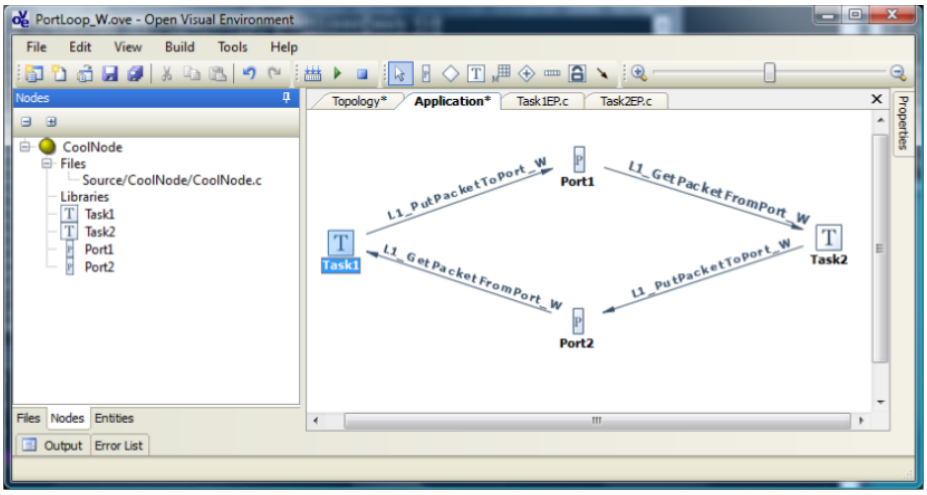
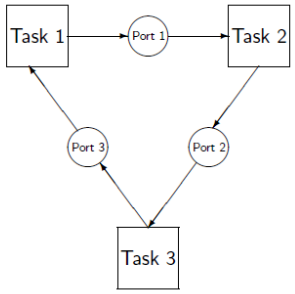
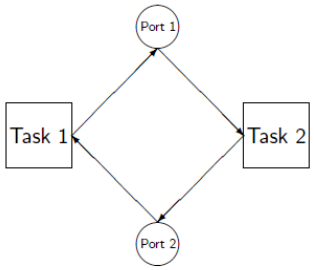


Fig. 8: Example of a OpenComRTOS Visual Designer diagram

8.2 Other defined models



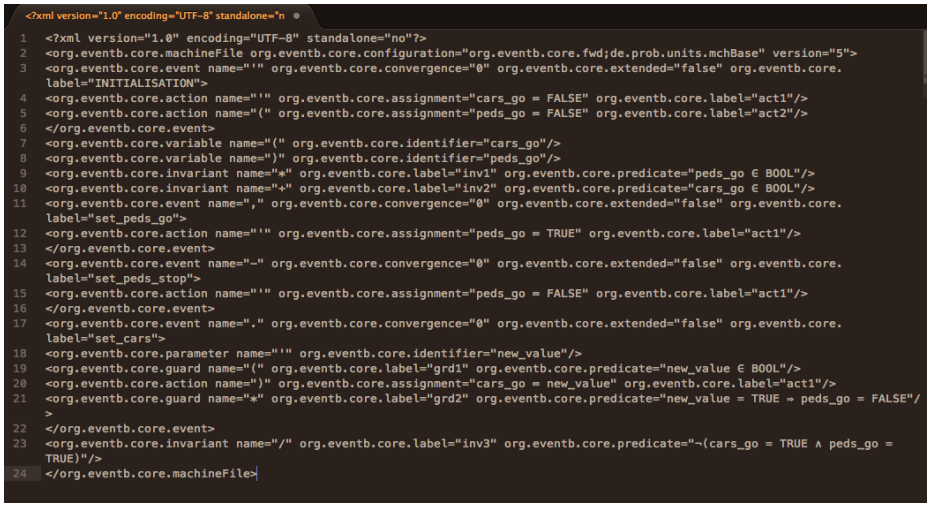
(a) Diagram of model with 2 Ports/Tasks



(b) Diagram of a model with 2 Ports/Tasks

Fig. 9: Other model examples

8.3 A XML Rodin file



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<org.eventb.core.machineFile org.eventb.core.configuration="org.eventb.core.fwd;de.prob.units.mchBase" version="5">
<org.eventb.core.event name="'" org.eventb.core.convergence="0" org.eventb.core.extended="false" org.eventb.core.
label="INITIALISATION">
<org.eventb.core.action name="'" org.eventb.core.assignment="cars_go = FALSE" org.eventb.core.label="act1"/>
<org.eventb.core.action name="(" org.eventb.core.assignment="peds_go = FALSE" org.eventb.core.label="act2"/>
</org.eventb.core.event>
<org.eventb.core.variable name="(" org.eventb.core.identifier="cars_go"/>
<org.eventb.core.variable name=")" org.eventb.core.identifier="peds_go"/>
<org.eventb.core.invariant name="*" org.eventb.core.label="inv1" org.eventb.core.predicate="peds_go ∈ BOOL"/>
<org.eventb.core.invariant name="+" org.eventb.core.label="inv2" org.eventb.core.predicate="cars_go ∈ BOOL"/>
<org.eventb.core.event name="," org.eventb.core.convergence="0" org.eventb.core.extended="false" org.eventb.core.
label="set_peds_go">
<org.eventb.core.action name="'" org.eventb.core.assignment="peds_go = TRUE" org.eventb.core.label="act1"/>
</org.eventb.core.event>
<org.eventb.core.event name="-" org.eventb.core.convergence="0" org.eventb.core.extended="false" org.eventb.core.
label="set_peds_stop">
<org.eventb.core.action name="'" org.eventb.core.assignment="peds_go = FALSE" org.eventb.core.label="act1"/>
</org.eventb.core.event>
<org.eventb.core.event name="." org.eventb.core.convergence="0" org.eventb.core.extended="false" org.eventb.core.
label="set_cars">
<org.eventb.core.parameter name="'" org.eventb.core.identifier="new_value"/>
<org.eventb.core.guard name="(" org.eventb.core.label="grd1" org.eventb.core.predicate="new_value ∈ BOOL"/>
<org.eventb.core.action name=")" org.eventb.core.assignment="cars_go = new_value" org.eventb.core.label="act1"/>
<org.eventb.core.guard name="*" org.eventb.core.label="grd2" org.eventb.core.predicate="new_value = TRUE ⇒ peds_go = FALSE"/
>
</org.eventb.core.event>
<org.eventb.core.invariant name="/" org.eventb.core.label="inv3" org.eventb.core.predicate="¬(cars_go = TRUE ∧ peds_go =
TRUE)"/>
</org.eventb.core.machineFile>
```

Fig. 10: Example of a Rodin file (machine.bum)

8.4 Importing a Rodin project

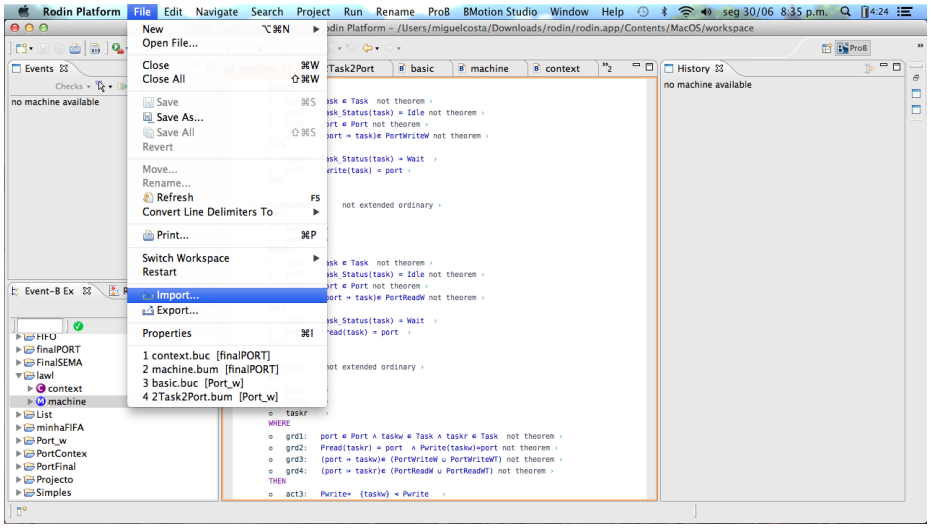


Fig. 11: 'File' tab → Import...

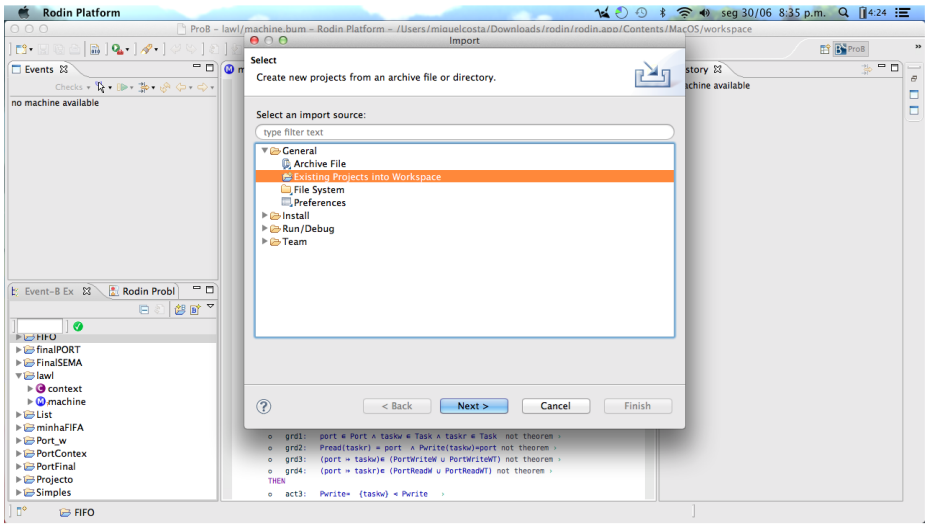


Fig. 12: Existing project

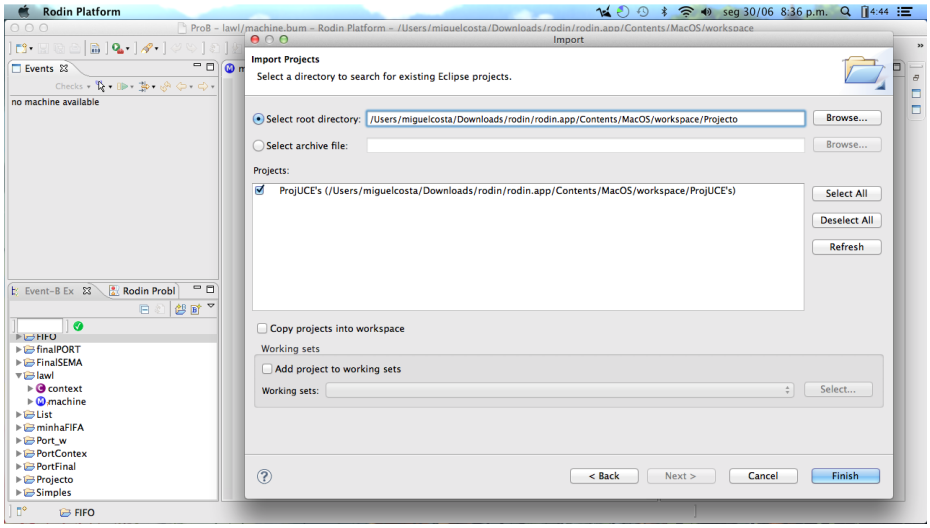


Fig. 13: Browsing the project

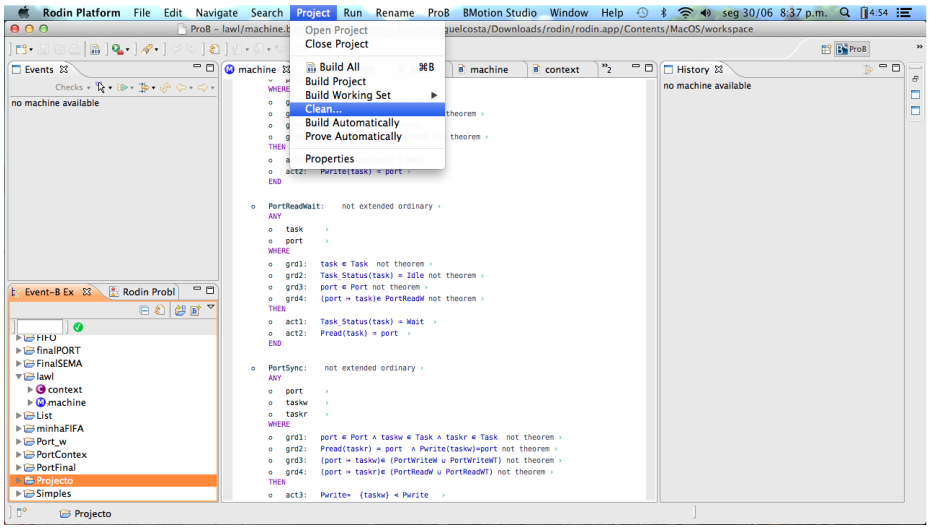


Fig. 14: 'Project' tab → Clean...

8.5 Execution of ProB

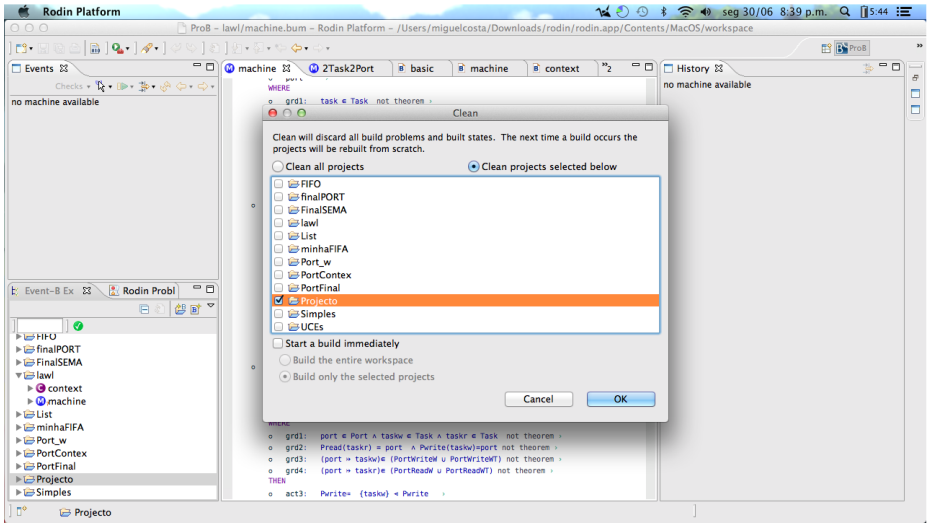


Fig. 15: Cleaning the project

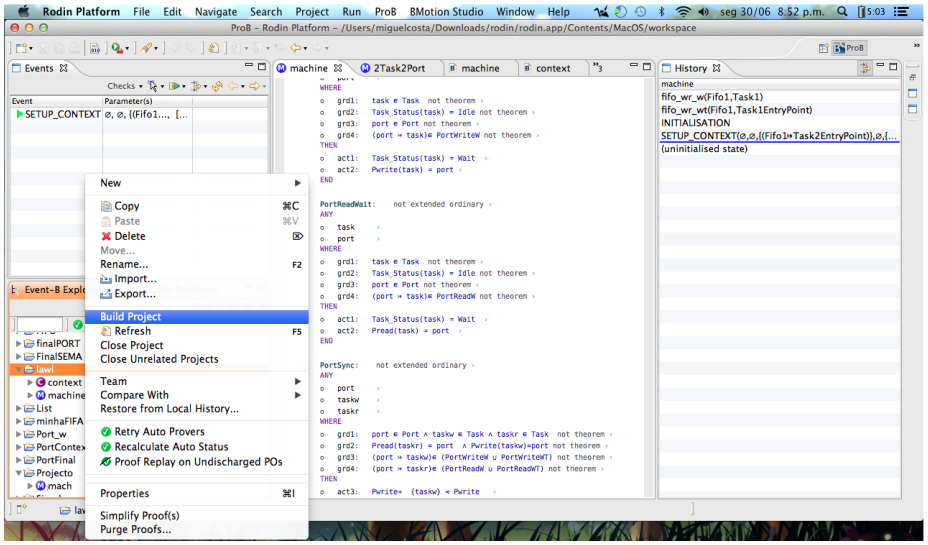


Fig. 16: Build the project

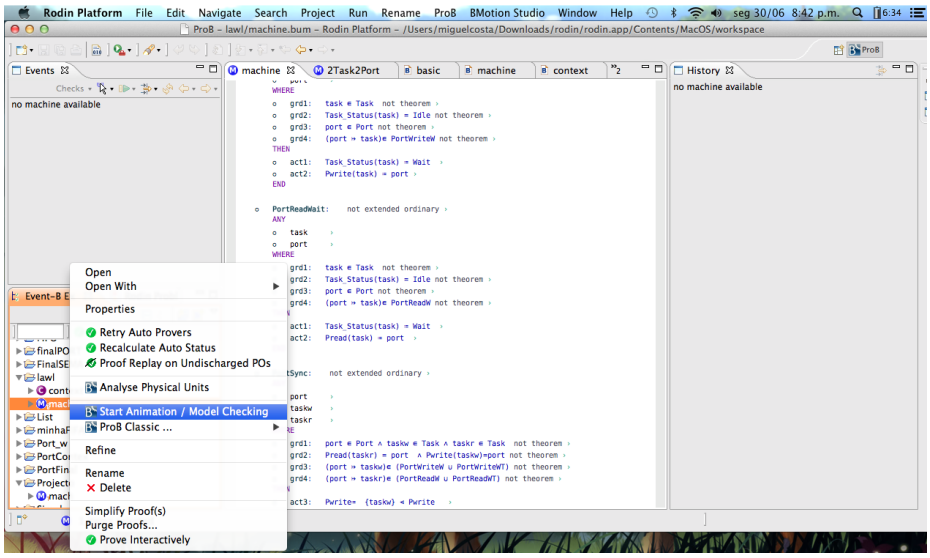


Fig. 17: Run ProB for a model

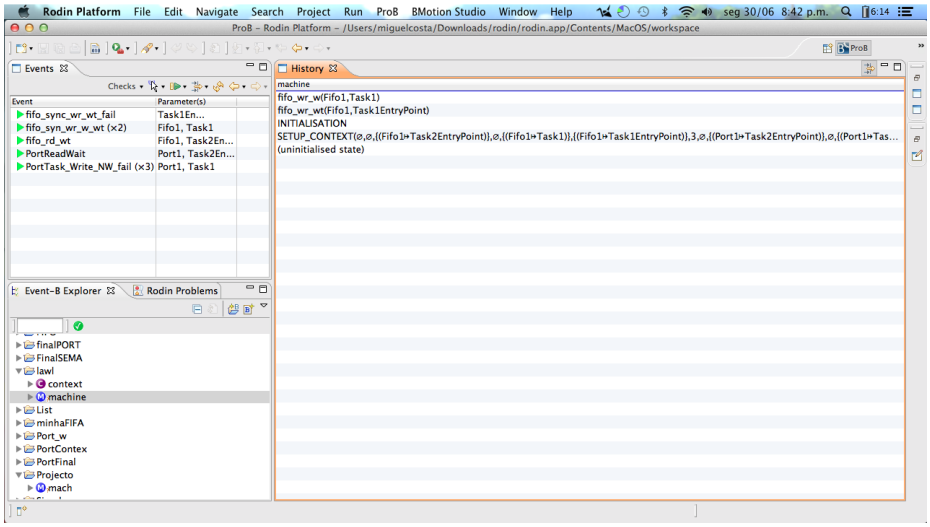


Fig. 18: Animation of the ProB execution

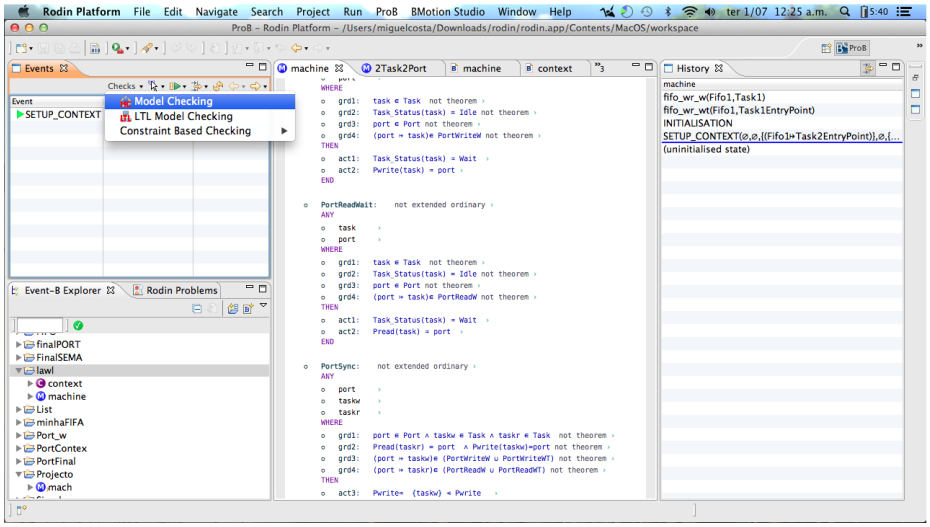


Fig. 19: Check Proprieties

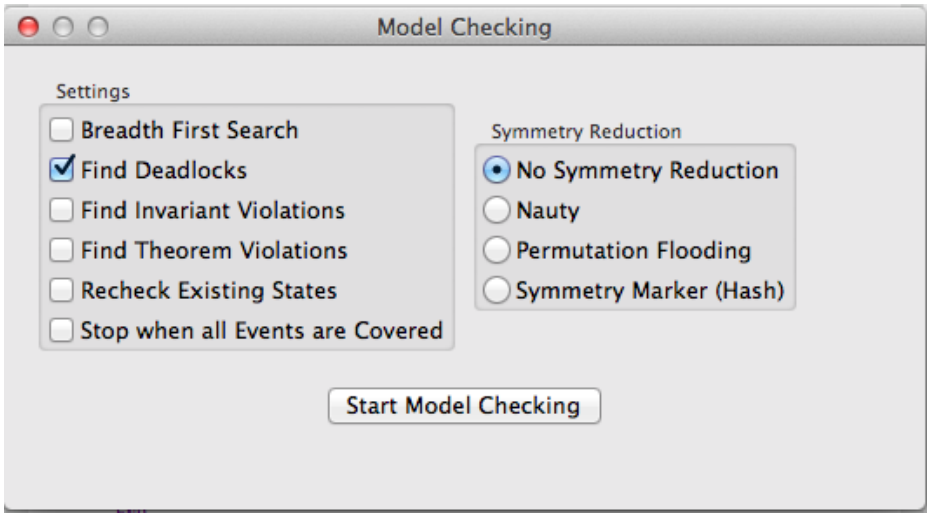


Fig. 20: A few proprieties that can be checked