RoboChart and RoboTool

Modelling, Verification and Simulation for Robotics

Alvaro Miyazawa Pedro Ribeiro

Ana Cavalcanti

Contents

I Tutorial

1. RoboTool Instalation

This section contain the instruction to download and install RoboTool.

1.1 Requirements

RoboTool (v.2.0.0) is a collection of Eclipse plugins. Its requirements for installations are:

1. Operating System: Linux, Windows or macOS[1];
2. Java 11 or above;
3. Eclipse 2021-06 IDE; and
4. FDR4 refinement checker (Optional).

1.2 Download and Update Site

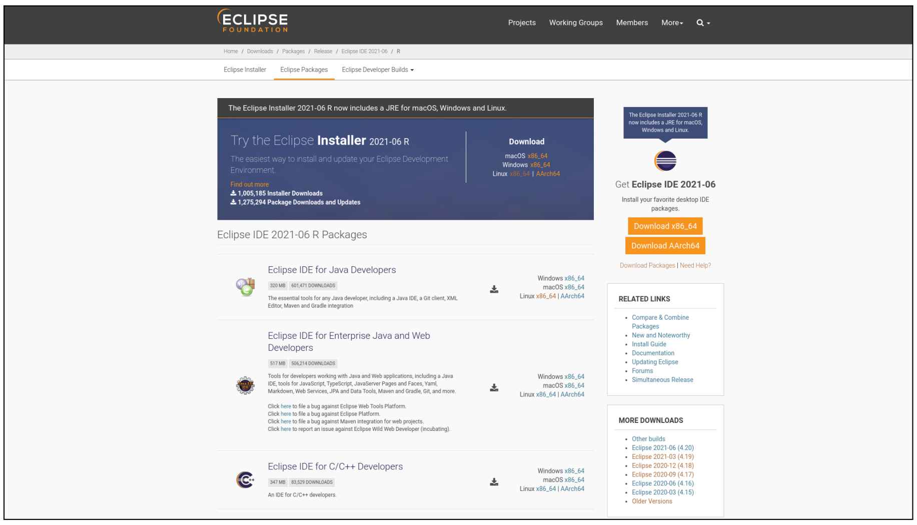
If you are already an Eclipse user, you can install RoboTool from its update site located in `https://www.cs.york.ac.uk/robostar/robotool/update/`. Detailed instructions for installation from the update site are given in the next section.

[1]While the RoboTool works on all these operating systems, it is only actively developed for Linux. Furthermore, models created in one operating system may lose formatting when open on different operating systems.

1.3 Installation

If you wish to install RoboTool from the update site, follow the steps below.

1. In order to install RoboTool, first download and install eclipse. We recommend starting with the package *Eclipse IDE for Java Developers*.



The Eclipse Installer 2021-06 R now includes a JRE for macOS, Windows and Linux.

Try the Eclipse **Installer** 2021-06 R

The easiest way to install and update your Eclipse Development Environment.

1,005,185 Installer Downloads
1,275,294 Package Downloads and Updates

Get **Eclipse IDE 2021-06**

Install your favorite desktop IDE packages.

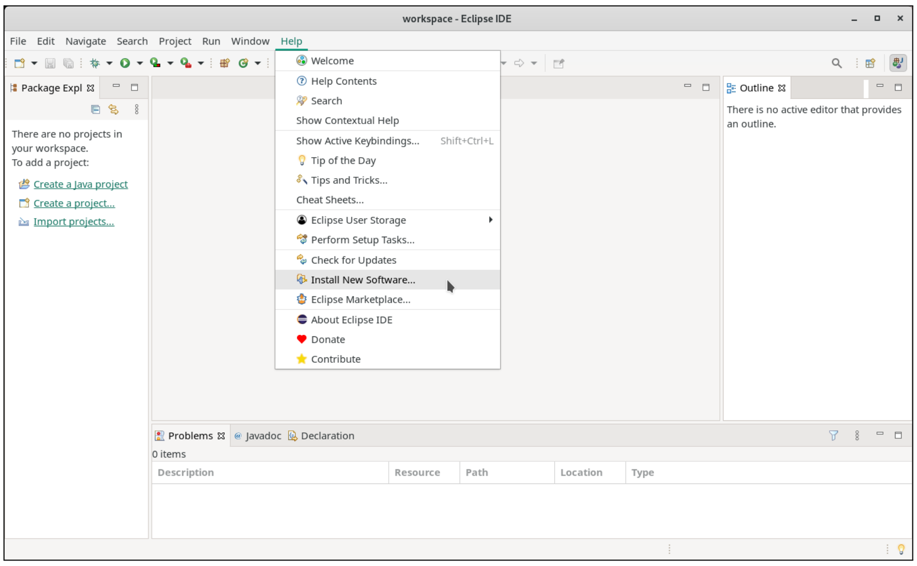
Eclipse IDE 2021-06 R Packages

Eclipse IDE for Java Developers
320 MB 601,471 DOWNLOADS
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration
Windows x86_64
macOS x86_64
Linux x86_64 | AArch64

Eclipse IDE for Enterprise Java and Web Developers
517 MB 506,214 DOWNLOADS
Tools for developers working with Java and Web applications, including a Java IDE, tools for JavaScript, TypeScript, JavaServer Pages and Faces, Yaml, Markdown, Web Services, JPA and Data Tools, Maven and Gradle, Git, and more.
Click here to file a bug against Eclipse Web Tools Platform.
Click here to file a bug against Eclipse Platform.
Click here to file a bug against Maven integration for web projects.
Click here to report an issue against Eclipse Wild Web Developer (incubating).
Windows x86_64
macOS x86_64
Linux x86_64 | AArch64

Eclipse IDE for C/C++ Developers
347 MB 83,529 DOWNLOADS
An IDE for C/C++ developers.
Windows x86_64
macOS x86_64
Linux x86_64 | AArch64

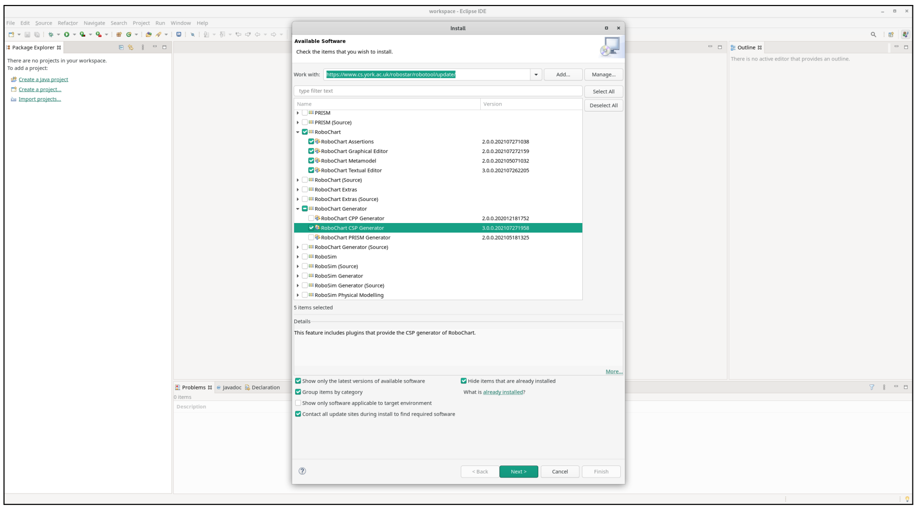
2. Start eclipse, and select the menu item `[Help > Install New Software...]`.



There are no projects in your workspace.
To add a project:

There is no active editor that provides an outline.

3. Enter the RoboTool update site `https://www.cs.york.ac.uk/robostar/robotool/update/` in the field `Work with`, wait for the available features to load, select the feature you require, and click `Next`. For the purposes of this tutorial, we recommend the following features:
   (a) `RoboChart Assertions`
   (b) `RoboChart Graphical Editor`
   (c) `RoboChart Metamodel`
   (d) `RoboChart Textual Editor`
   (e) `RoboChart CSP Generator`



Available Software

Check the items that you wish to install.

Work with: https://www.cs.york.ac.uk/robostar/robotool/update/

Name	Version
☐ PRISM	
☐ PRISM (Source)	
✓ RoboChart	
✓ RoboChart Assertions	2.0.0.202107271038
✓ RoboChart Graphical Editor	2.0.0.202107272159
✓ RoboChart Metamodel	2.0.0.202105071032
✓ RoboChart Textual Editor	3.0.0.202107262205
☐ RoboChart (Source)	
☐ RoboChart Extras	
☐ RoboChart Extras (Source)	
▬ RoboChart Generator	
☐ RoboChart CPP Generator	2.0.0.202012181752
✓ RoboChart CSP Generator	3.0.0.202107271958
☐ RoboChart PRISM Generator	2.0.0.202105181325
☐ RoboChart Generator (Source)	
☐ RoboSim	
☐ RoboSim (Source)	
☐ RoboSim Generator	
☐ RoboSim Generator (Source)	
☐ RoboSim Physical Modelling	

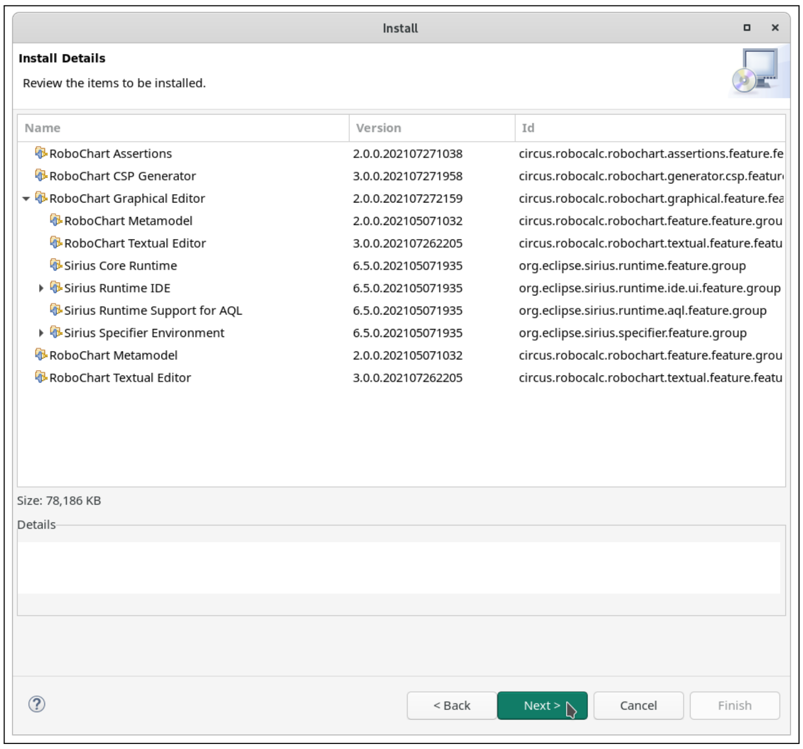
5 items selected

Details

This feature includes plugins that provide the CSP generator of RoboChart.

✓ Show only the latest versions of available software
✓ Hide items that are already installed
✓ Group items by category
What is already installed?
☐ Show only software applicable to target environment
✓ Contact all update sites during install to find required software

4. Eclipse will present the list of dependencies that will also be installed. Click `Next`.



Install Details

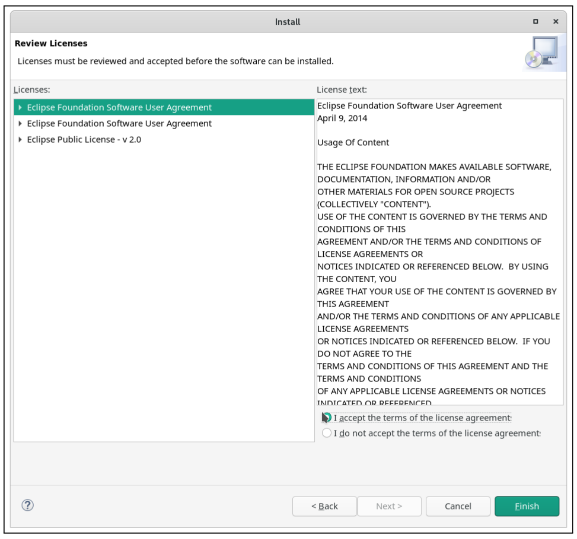
Review the items to be installed.

Name	Version	Id
RoboChart Assertions	2.0.0.202107271038	circus.robocalc.robochart.assertions.feature.fe
RoboChart CSP Generator	3.0.0.202107271958	circus.robocalc.robochart.generator.csp.featur
RoboChart Graphical Editor	2.0.0.202107272159	circus.robocalc.robochart.graphical.feature.fea
RoboChart Metamodel	2.0.0.202105071032	circus.robocalc.robochart.feature.feature.grou
RoboChart Textual Editor	3.0.0.202107262205	circus.robocalc.robochart.textual.feature.featu
Sirius Core Runtime	6.5.0.202105071935	org.eclipse.sirius.runtime.feature.group
Sirius Runtime IDE	6.5.0.202105071935	org.eclipse.sirius.runtime.ide.ui.feature.group
Sirius Runtime Support for AQL	6.5.0.202105071935	org.eclipse.sirius.runtime.aql.feature.group
Sirius Specifier Environment	6.5.0.202105071935	org.eclipse.sirius.specifier.feature.group
RoboChart Metamodel	2.0.0.202105071032	circus.robocalc.robochart.feature.feature.grou
RoboChart Textual Editor	3.0.0.202107262205	circus.robocalc.robochart.textual.feature.featu

Size: 78,186 KB

Details

5. Accept all licenses, and click `Finish`.



Review Licenses

Licenses must be reviewed and accepted before the software can be installed.

- Eclipse Foundation Software User Agreement
- Eclipse Foundation Software User Agreement
- Eclipse Public License - v 2.0

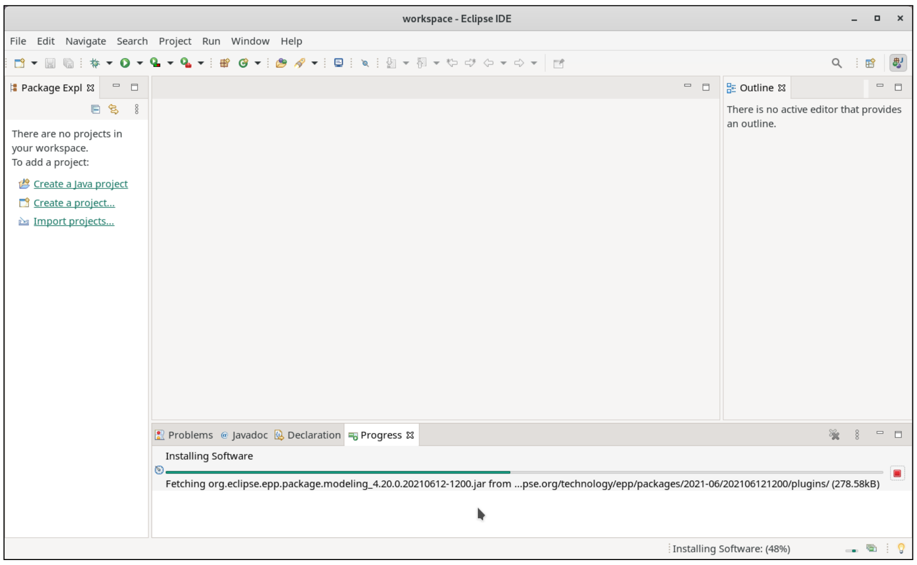
Eclipse Foundation Software User Agreement
April 9, 2014

Usage Of Content

THE ECLIPSE FOUNDATION MAKES AVAILABLE SOFTWARE, DOCUMENTATION, INFORMATION AND/OR OTHER MATERIALS FOR OPEN SOURCE PROJECTS (COLLECTIVELY "CONTENT"). USE OF THE CONTENT IS GOVERNED BY THE TERMS AND CONDITIONS OF THIS AGREEMENT AND/OR THE TERMS AND CONDITIONS OF LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW. BY USING THE CONTENT, YOU AGREE THAT YOUR USE OF THE CONTENT IS GOVERNED BY THIS AGREEMENT AND/OR THE TERMS AND CONDITIONS OF ANY APPLICABLE LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW. IF YOU DO NOT AGREE TO THE TERMS AND CONDITIONS OF THIS AGREEMENT AND THE TERMS AND CONDITIONS OF ANY APPLICABLE LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED

[✓ I accept the terms of the license agreement]
I do not accept the terms of the license agreement

6. Wait for Eclipse to finish downloading all plugins.



There are no projects in your workspace.
To add a project:

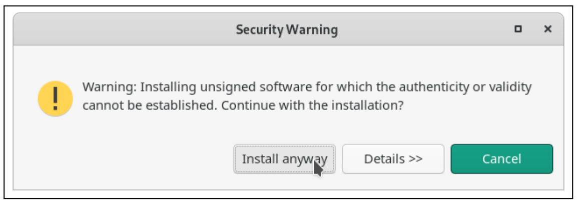
There is no active editor that provides an outline.

Installing Software

Fetching org.eclipse.epp.package.modeling_4.20.0.20210612-1200.jar from ...pse.org/technology/epp/packages/2021-06/202106121200/plugins/ (278.58kB)

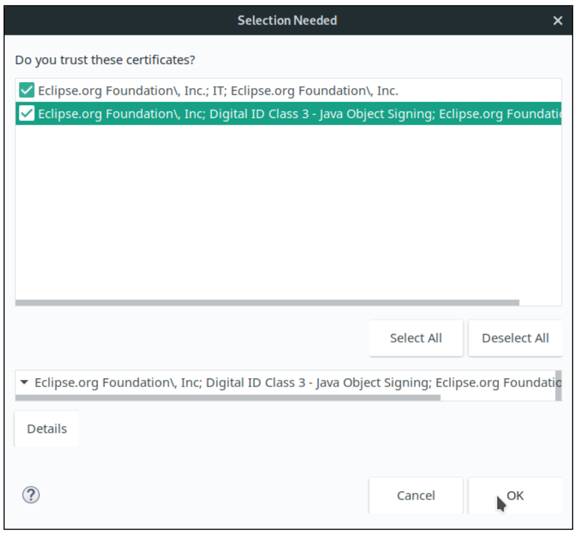
Installing Software: (48%)

7. Eclipse will show a security warning indicating that the software is unsigned. Click `Install anyway`.



Warning: Installing unsigned software for which the authenticity or validity cannot be established. Continue with the installation?

8. If prompted, select all certificates as trusted, and click `OK`.



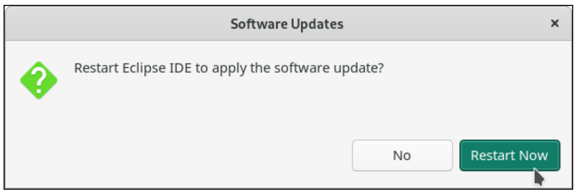
Selection Needed

Do you trust these certificates?

✓ Eclipse.org Foundation\, Inc.; IT; Eclipse.org Foundation\, Inc.
✓ Eclipse.org Foundation\, Inc; Digital ID Class 3 - Java Object Signing; Eclipse.org Foundati

Eclipse.org Foundation\, Inc; Digital ID Class 3 - Java Object Signing; Eclipse.org Foundatic

9. Finally, restart Eclipse.

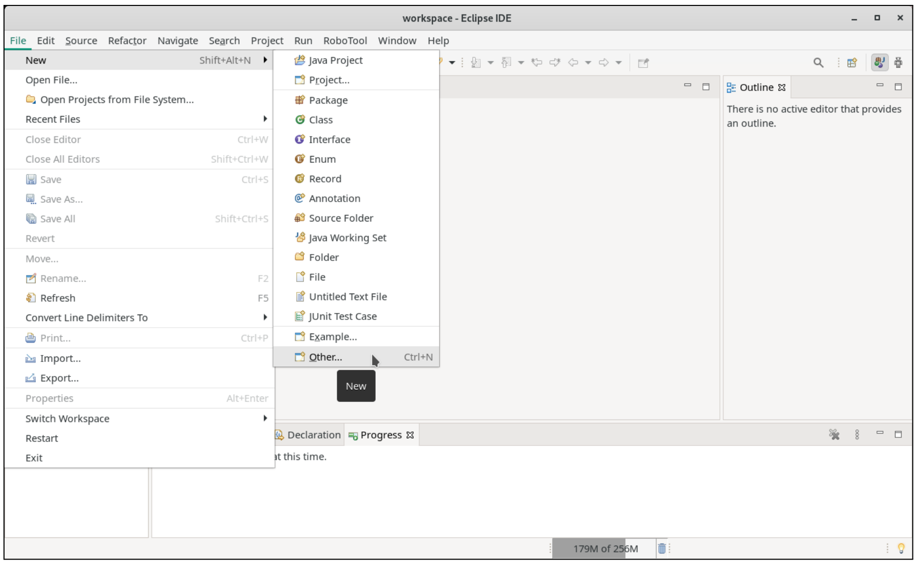


Software Updates

Restart Eclipse IDE to apply the software update?

1.4 Creating a new RoboChart Project

1. Click on the menu item `[File > New > Other...]`.

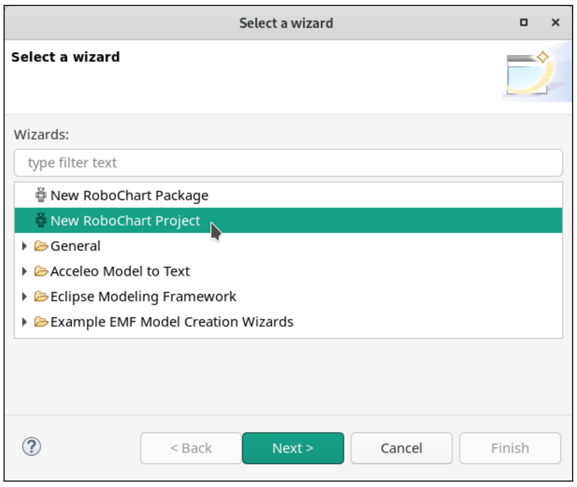


workspace - Eclipse IDE

There is no active editor that provides an outline.

...t this time.

2. Select the wizard `New RoboChart Project`, and click `Next`.


Select a wizard

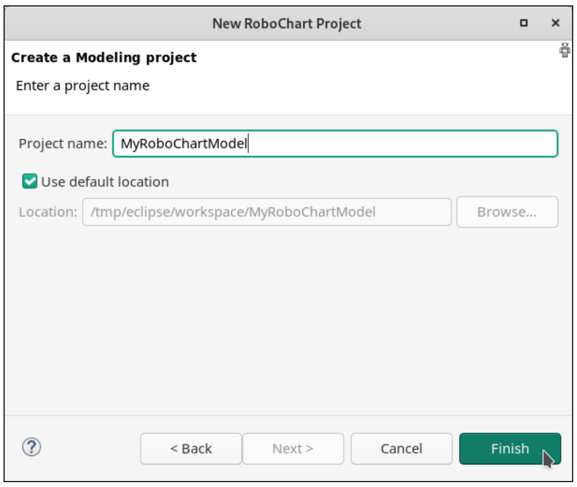
Select a wizard

Wizards:

type filter text

- New RoboChart Package
- New RoboChart Project
- General
- Acceleo Model to Text
- Eclipse Modeling Framework
- Example EMF Model Creation Wizards

3. Give the project a name, optionally change the default location, and click `Finish`.


New RoboChart Project

Create a Modeling project

Enter a project name

Project name: MyRoboChartModel

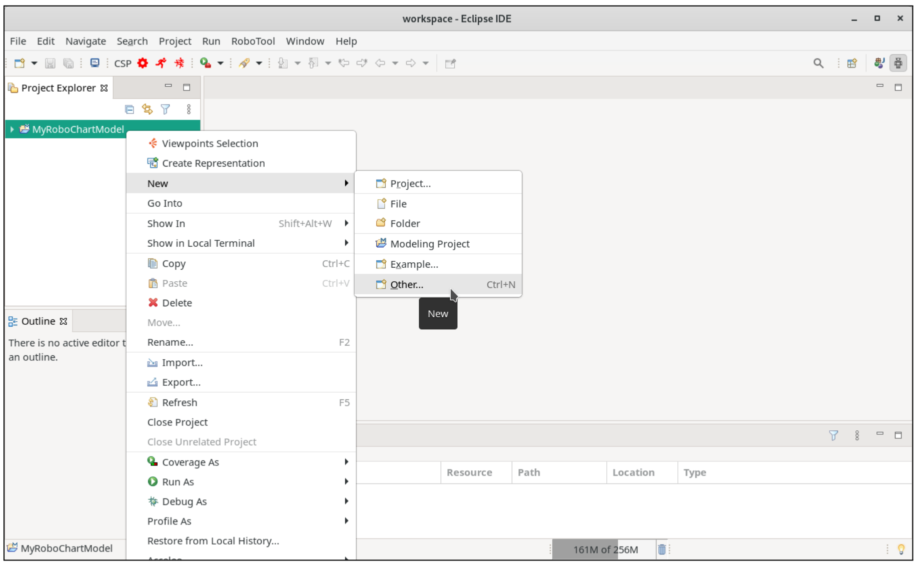
[✓] Use default location

Location: /tmp/eclipse/workspace/MyRoboChartModel

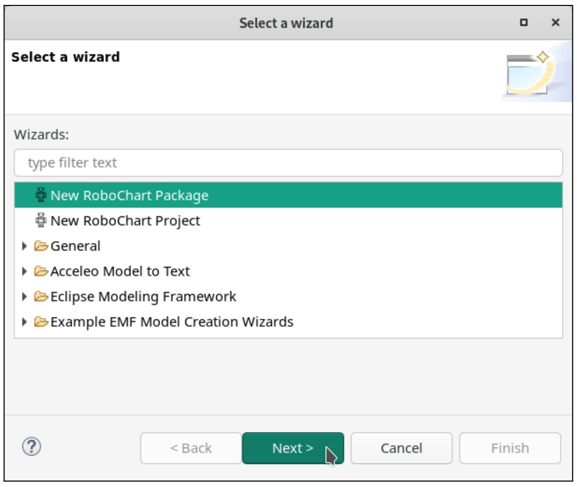
The first time a project is created, Eclipse will switch to the RoboChart perspective indicated by the RoboTool icon on the top-right corner of the window. This perspective is based on the modelling perspective and organises the Eclipse window to include the properties tab at the bottom, the project explorer at the top-left and the outline at the bottom-left corner. Additionally, it adds shortcuts to `new RoboChart project` and `RoboChart package` directly on the `File > New` menu item.

1.5 Creating a new RoboChart diagram

1. Right click on the project, and click on the item `[New > Other...]`.



2. Select the wizard `New RoboChart Package`, and click `Next`.


Select a wizard

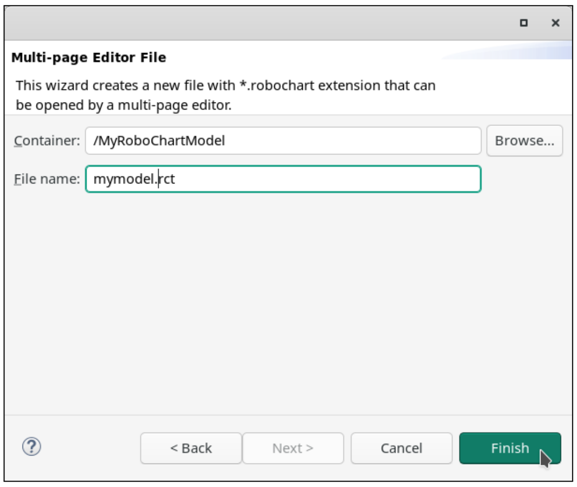
Select a wizard

Wizards:

type filter text

- New RoboChart Package
- New RoboChart Project
- General
- Acceleo Model to Text
- Eclipse Modeling Framework
- Example EMF Model Creation Wizards

3. Select a file name and make sure a container (e.g. your project) is selected. Click `Finish`.

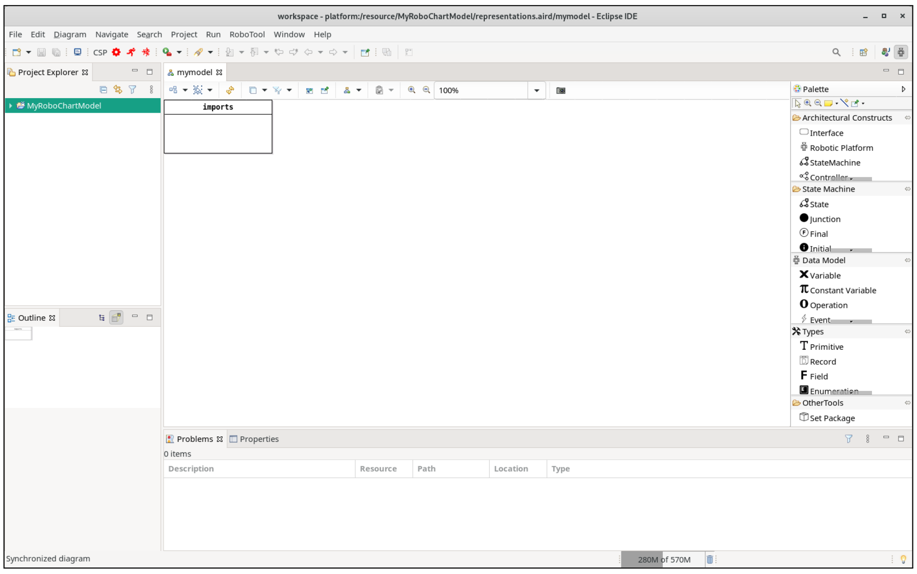

Multi-page Editor File

This wizard creates a new file with *.robochart extension that can be opened by a multi-page editor.

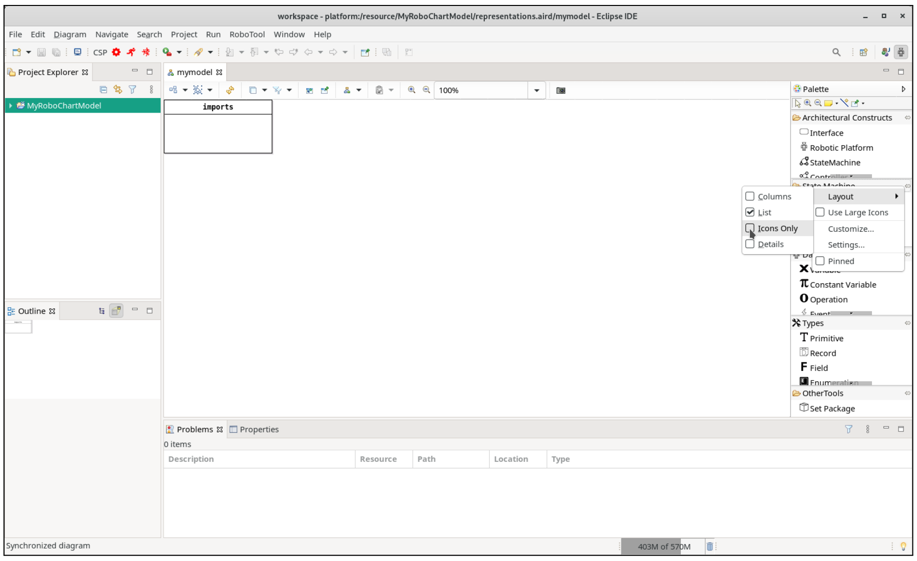
Container: /MyRoboChartModel

File name: mymodel.rct

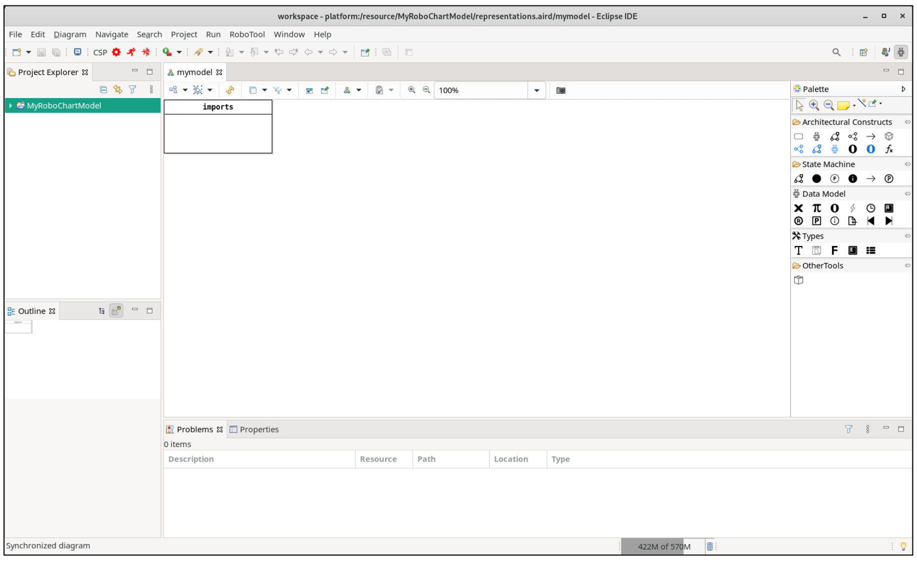
4. The editor will open the newly created diagram.



5. (Optional) The palette at the right does not show all diagram edition tools, and you must expand and contract the sections as necessary. It is possible to configure the palette to show only icons. Right click on the palette, and select `[Layout > Icons Only]`.



6. (Optional) With the `icons Only` layout selected, the palette can show all the edition tools. To find out the name of the tool, you can hover you mouse over the icons for a few seconds.

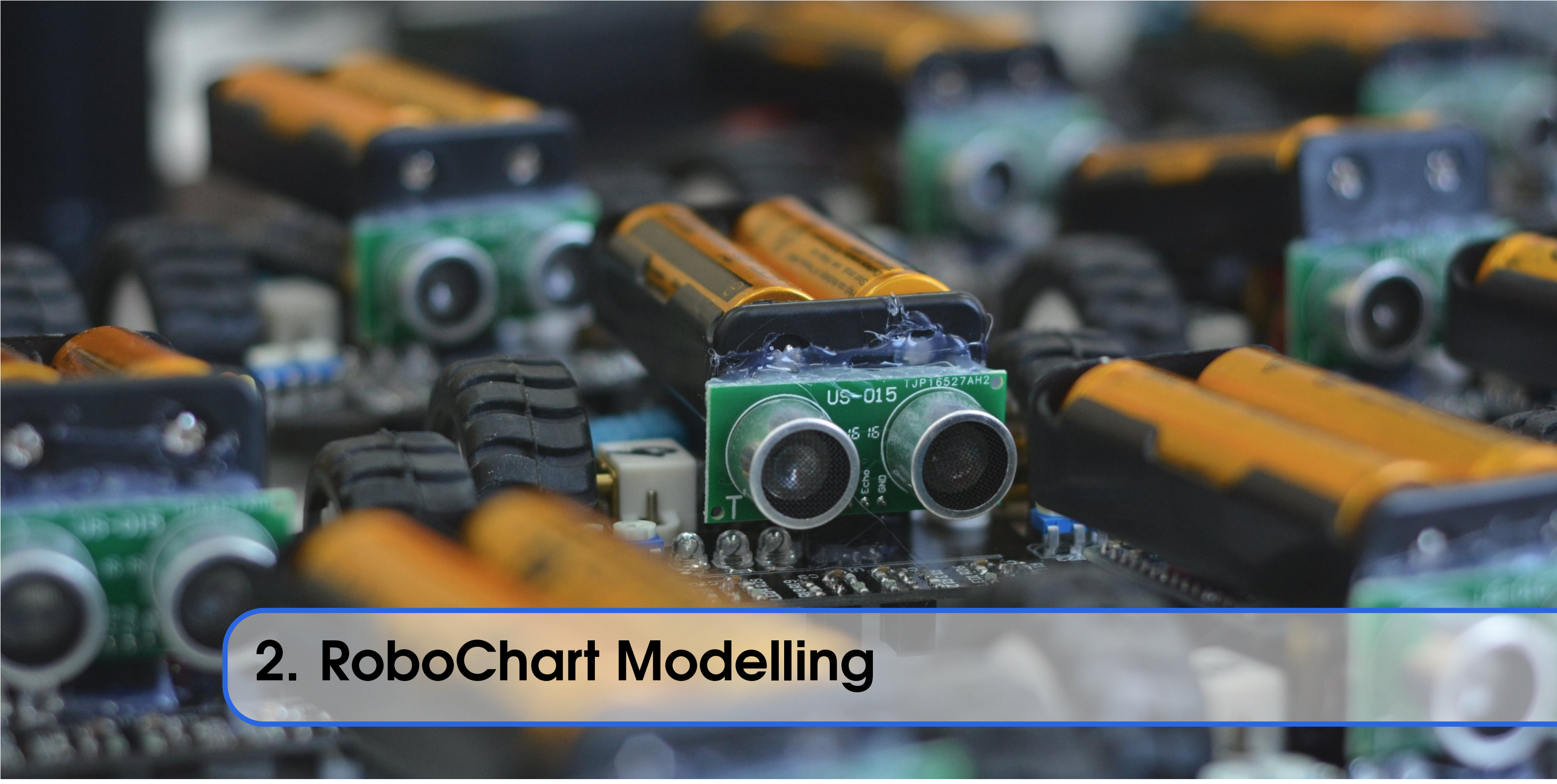


1.6 Exercises

Exercise 1.1 Download Eclipse and install RoboTool.

Exercise 1.2 Switch to the modelling perspective.

Exercise 1.3 Create a new RoboChart project and an empty RoboChart package.



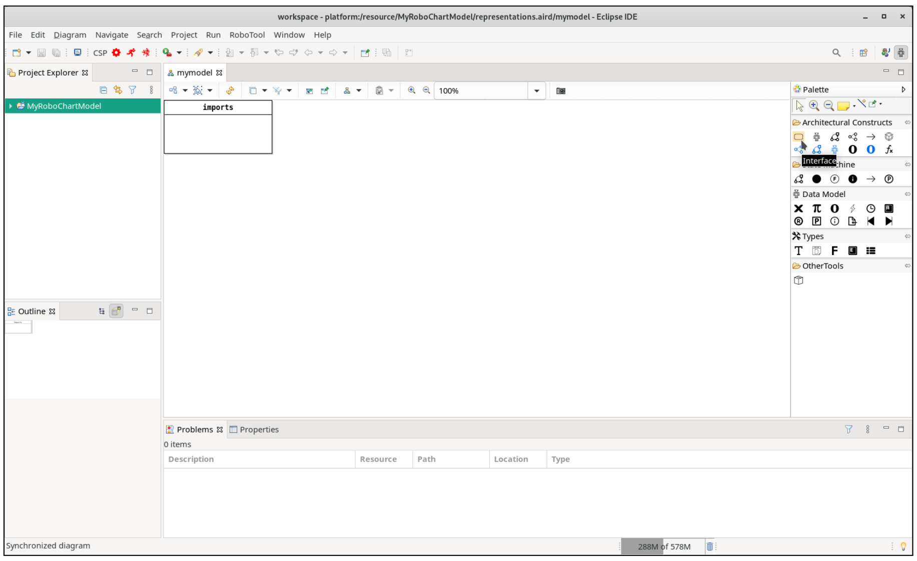
2. RoboChart Modelling

This chapter includes step-by-step instruction to build a simple RoboChart model.

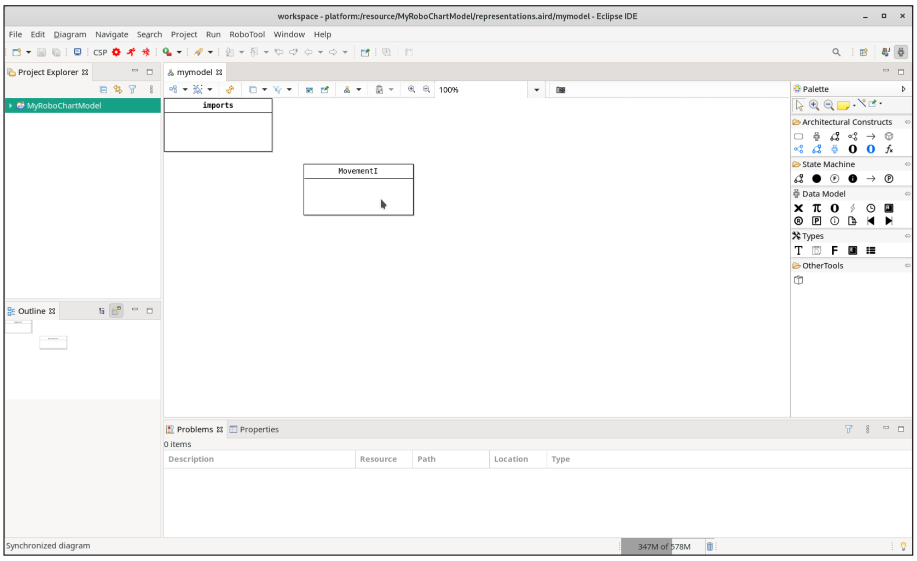
2.1 Creating Interfaces

Interfaces in RoboChart can contain events, variables and operations. Below are the steps used to create two interfaces, the first containing an operation `move`, and the second containing an event.

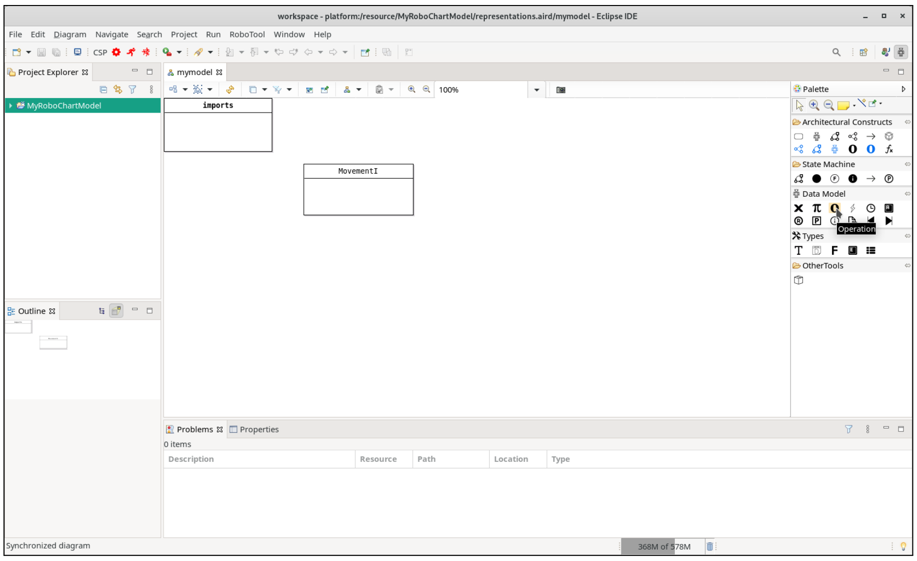
1. Select the interface tool ▭ in the Architectural Constructs section of the palette, and click on the editor to position the interface.



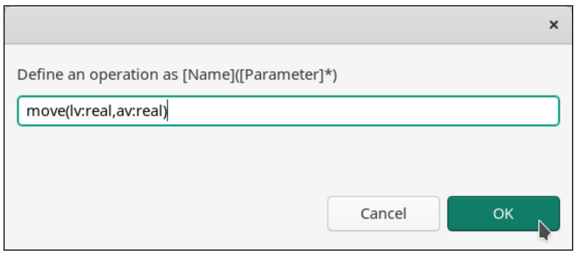
2. With the interface selected, press F2 to rename the interface.



3. Add an operation by selecting the operation signature tool **O** in the `Data Model` section, and clicking on the interface.



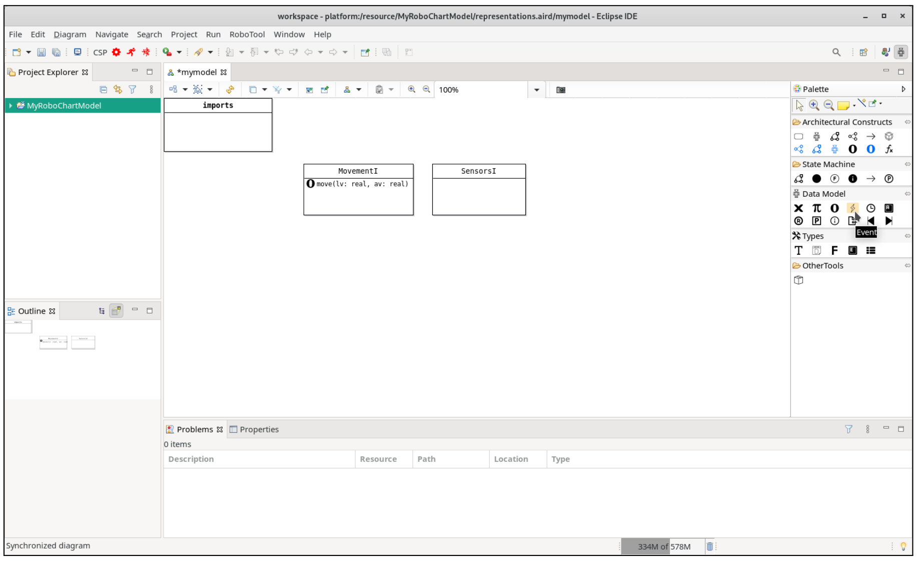
4. RoboTool will open a dialog window to input the operation signature. Type the operation signature[1] and click `OK` to add the operation to the interface.


Define an operation as [Name]([Parameter]*)

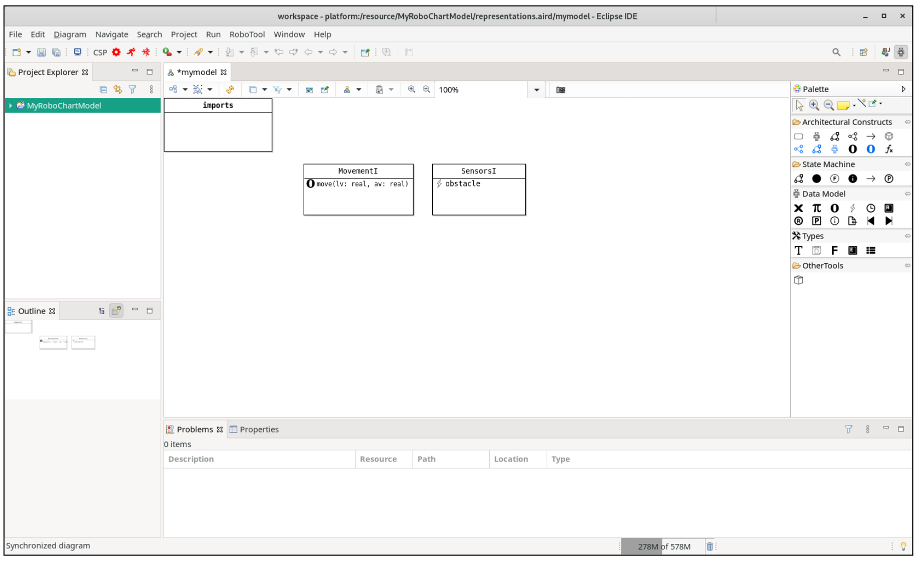
move(lv:real,av:real)

[1]Syntax errors are listed in the dialog box.

5. Add another interface called `SensorsI` and add an event to it by selecting the event tool ⚡ in the `Data Model` section, and clicking on the interface. A similar dialog as in Step 4 will open. Input the event definition and click `OK`.



6. The final model with two interfaces `MovementI` and `SensorsI` should be as follows.



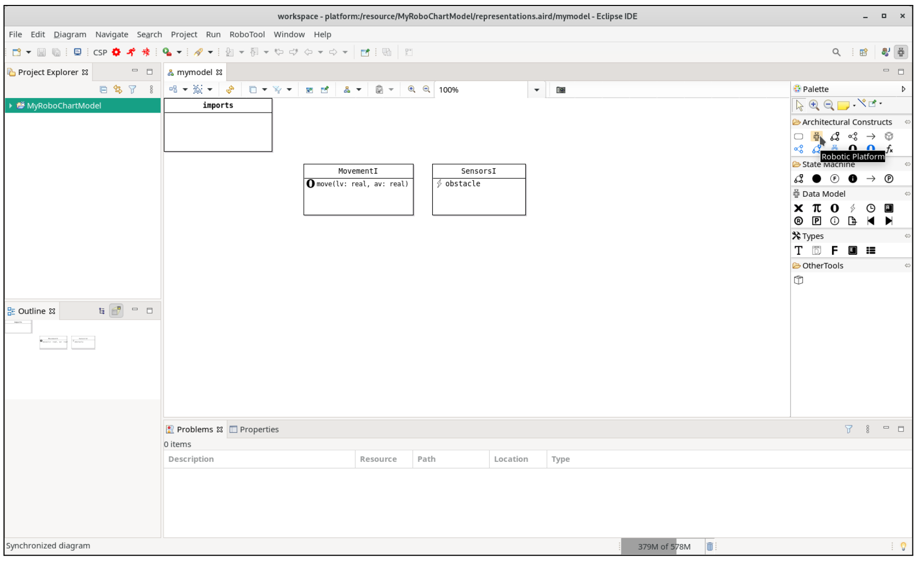
2.2 Creating Robotic Platforms

A robotic platform represents an abstraction of the robot in terms of the variables, events and operations available to the software. These elements can either be added directly to the robotic platform, or they can be indirectly added by providing Ⓟ, requiring Ⓡ or defining ⓘ interfaces.

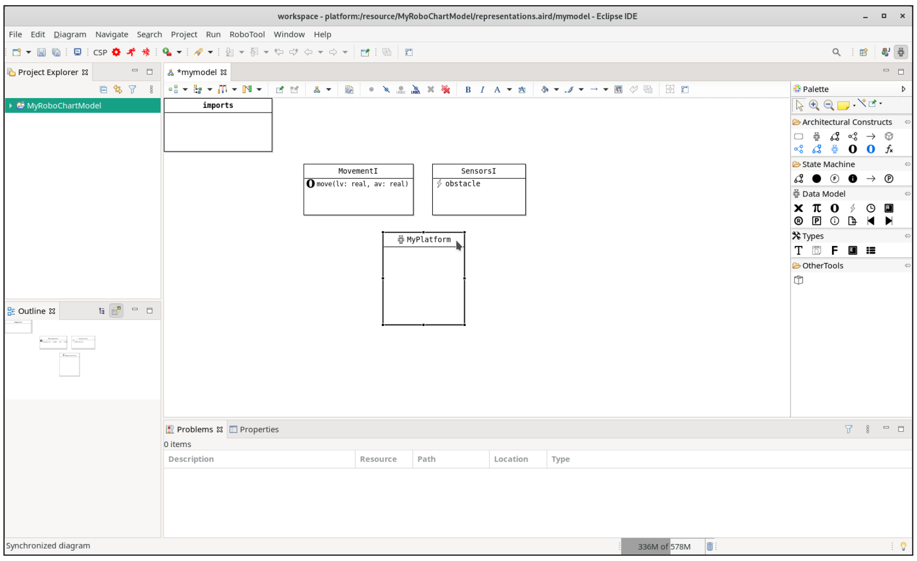
Required and provided interfaces represent dependencies between platforms, controllers and state machines, and can contain only variables and operations, which can be shared between components. Events are locally defined and are connected explicitly. Defined interfaces simply define its variables and events on platforms (controllers and state machines) as if they were declared individually. Defined interfaces cannot contain operations.

The steps to create a robotic platform using the interfaces defined in the previous section are shown next.

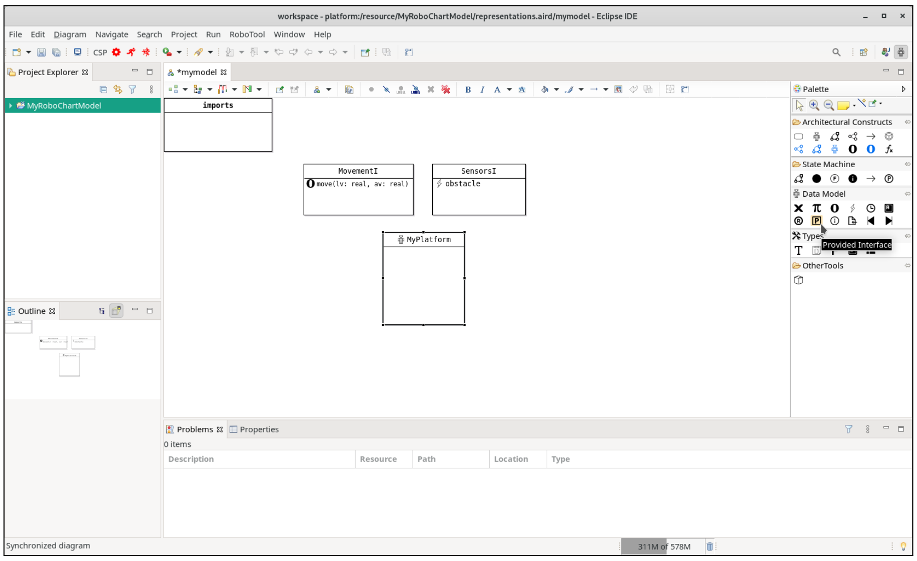
1. Select the robotic platform tool 🤖 in the Architectural Constructs section of the palette, and click on the editor to position the platform.



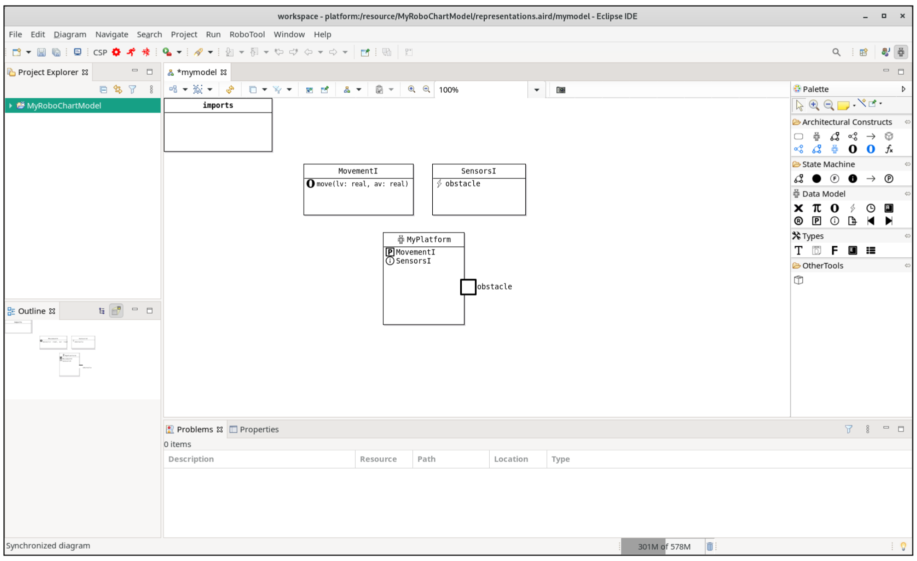
2. With the platform selected, press F2 to rename the robotic platform.



3. Select the provided interface tool 🄿 (required Ⓡ or defined ⓘ) in the Data Model section of the palette, and click on the platform. Input the name of an interface and click `OK`.



4. The result of providing the `MovementI` interface, and defining the `SensorsI` interface is shown below.

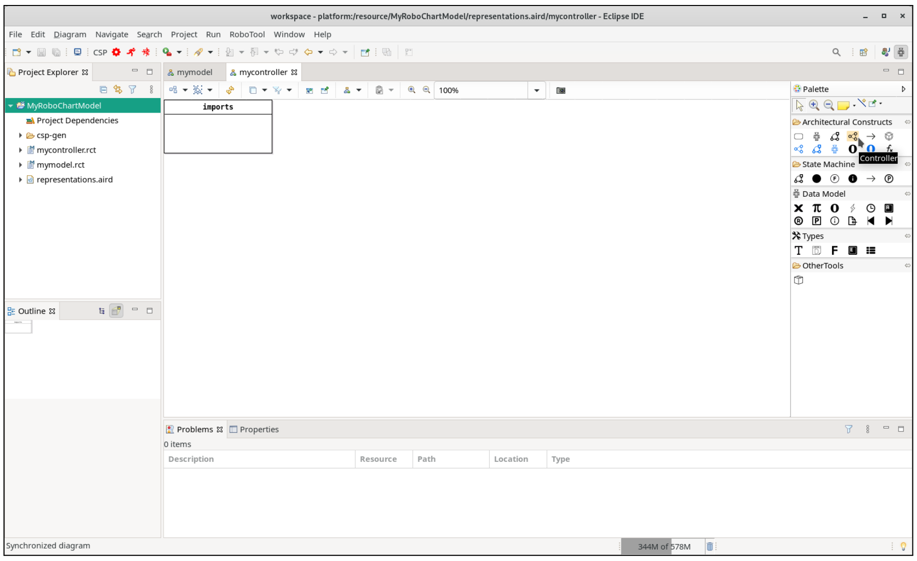


Additionally, events, variables and operations can be added to platforms in an identical fashion as for interfaces.

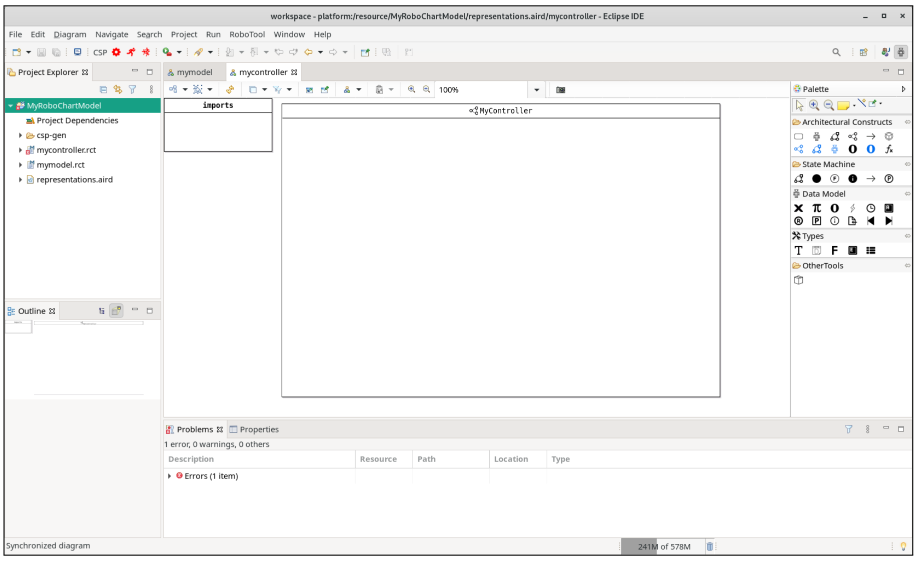
2.3 Creating Controllers

A controller describes possibly parallel behaviours, containing one or more state machines.

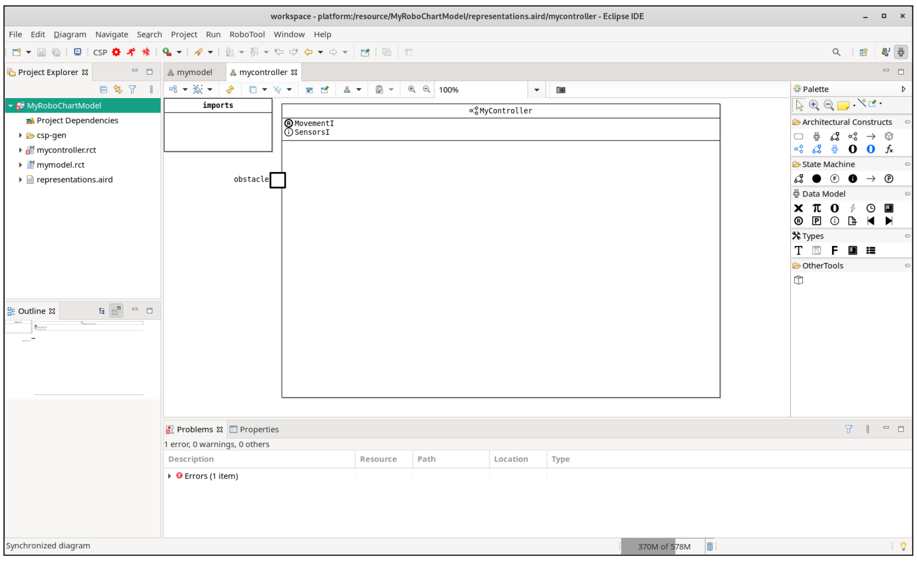
1. Select the controller tool in the Architectural Constructs section of the palette, and click on the editor to position the controller.



2. After clicking, you can drag the mouse pointer to size the controller box. The resulting diagram is shown below.



3. Use the required and defined interface tool to provide `MovementI` and define `SensorsI`.

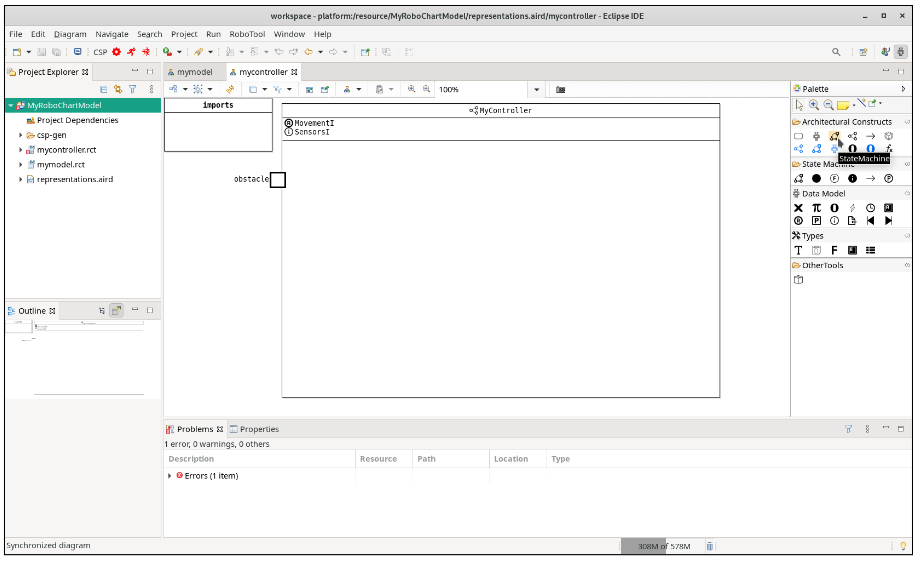


1 error, 0 warnings, 0 others
Errors (1 item)

2.4 Creating State Machines

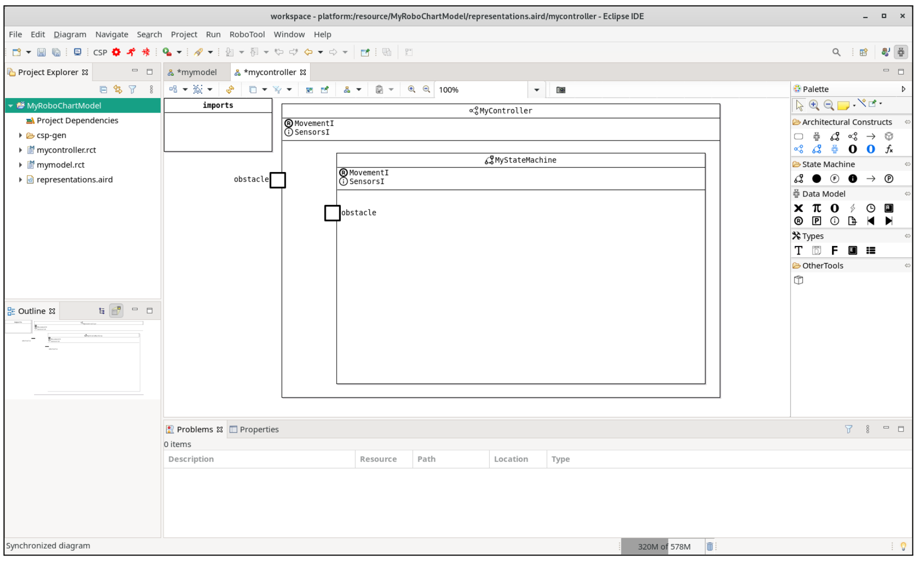
In this section, we will create a simple state machine with two states `Moving` and `Turning` inside the controller created in the previous section.

1. Select the state machine tool in the Architectural Constructs section of the palette, and click inside the controller to position the state machine.

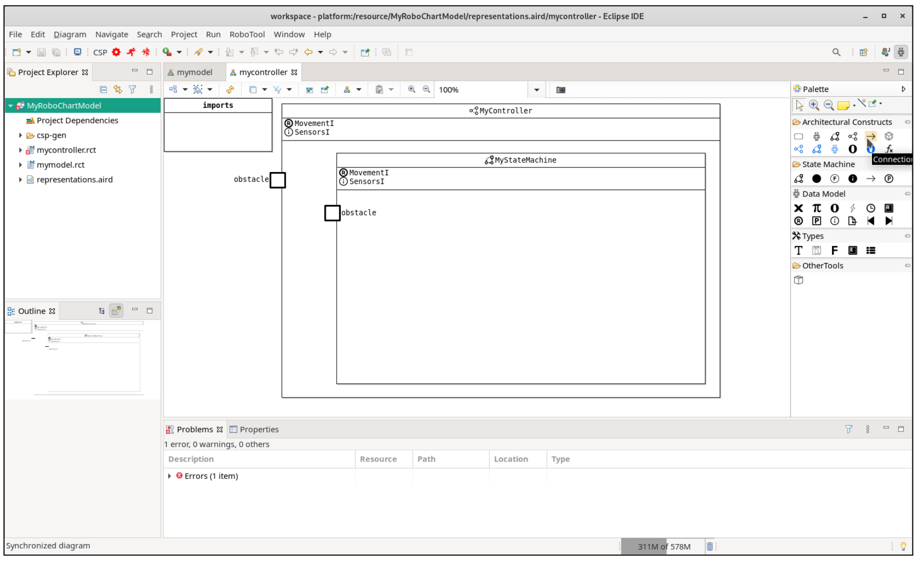


1 error, 0 warnings, 0 others
Errors (1 item)

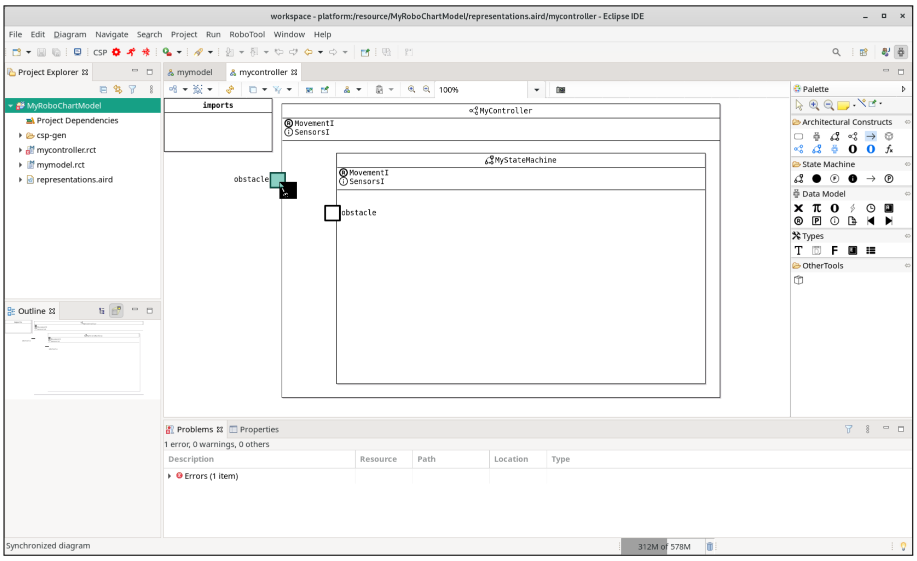
2. Rename it and add the required and defined interfaces as in the previous section.



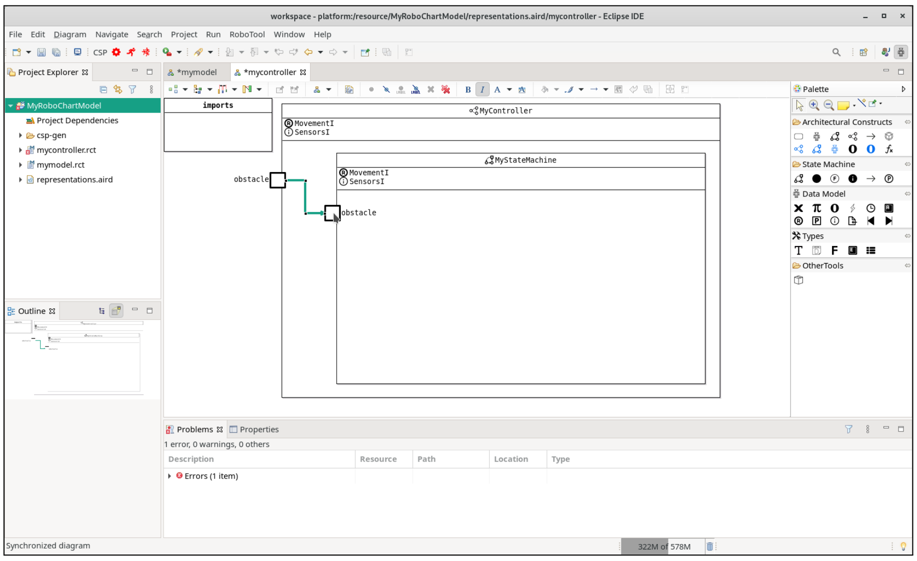
3. In order to connect the events of the controller and state machine, select the connection tool $\rightarrow$ in the Architectural Constructs section.



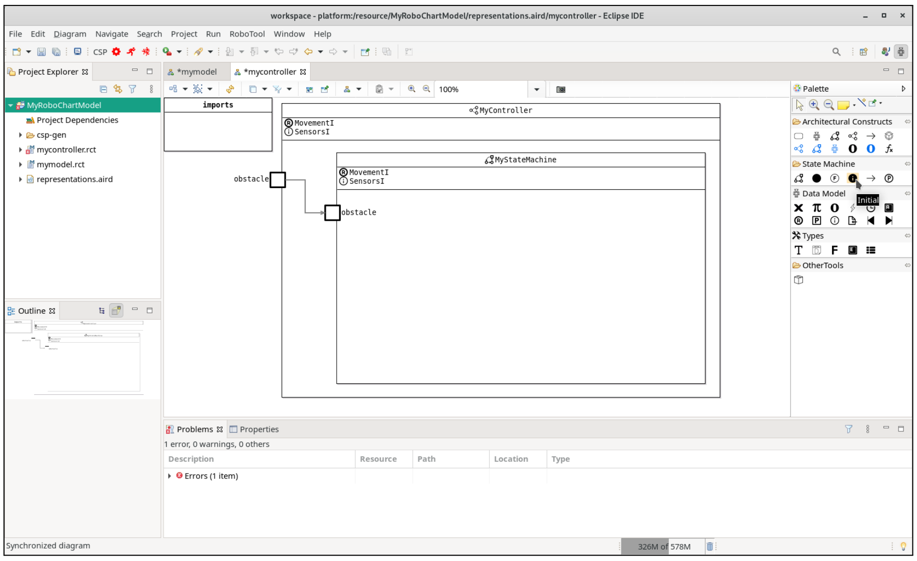
4. Start the connection by first clicking on the event of the controller.



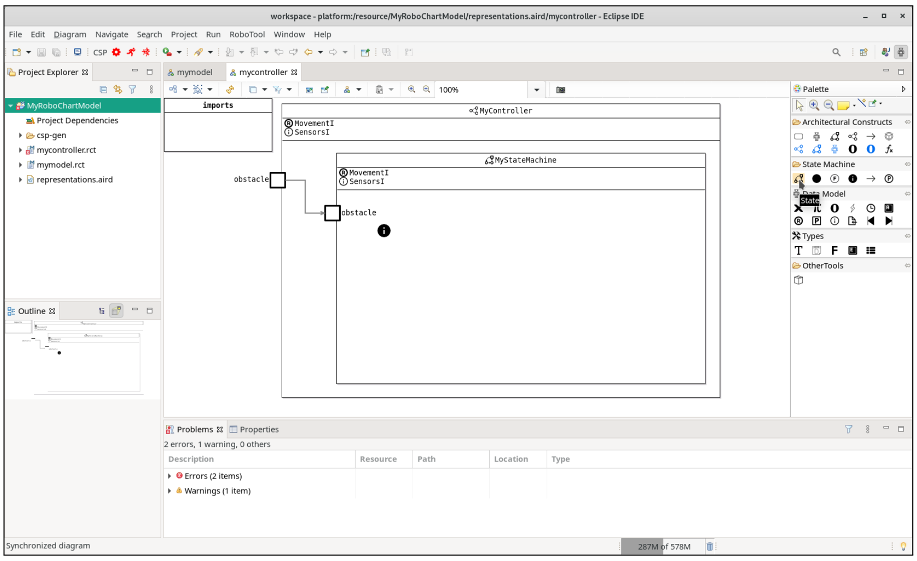
5. Complete the connection by clicking on the event of the state machine.



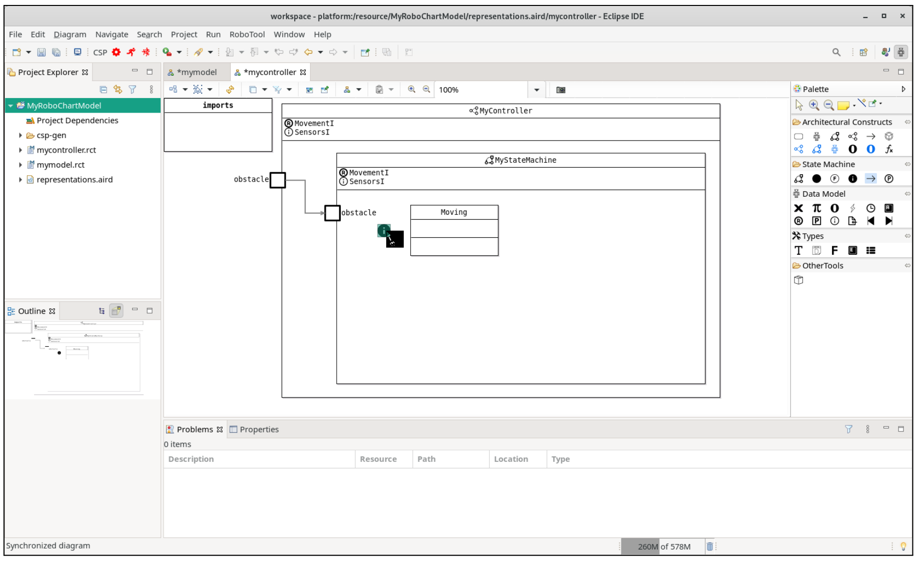
6. Next, add an initial junction to the state machine using the initial junction tool 🛈 in the State Machine section of the palette. Click inside the state machine to position the initial junction.



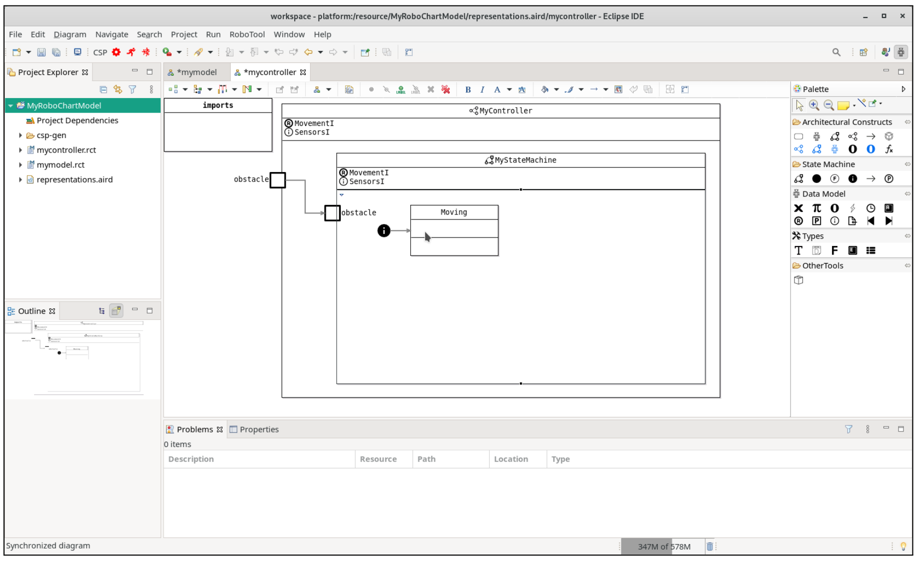
7. Select the state tool ▭ in the State Machine section of the palette, and click inside the state machine to position state. Press F2 to rename the state.



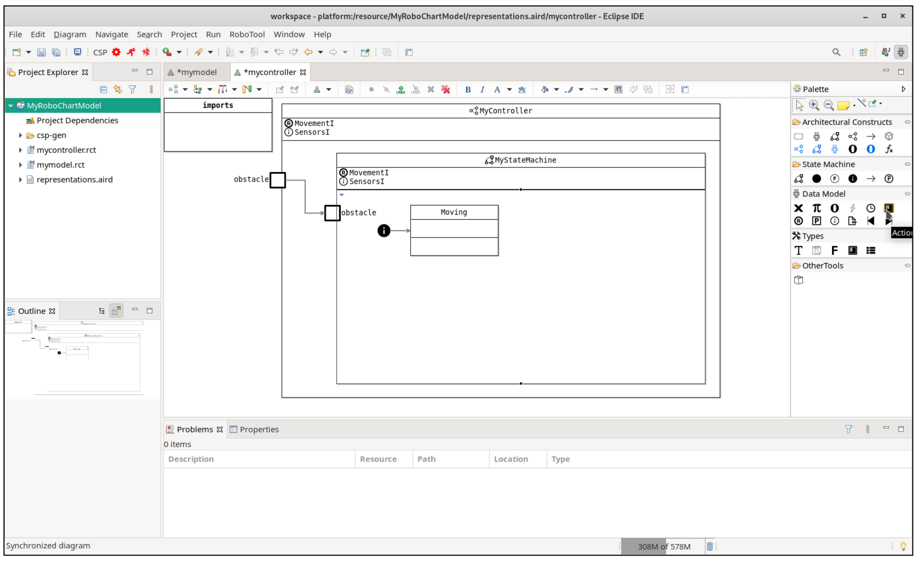
8. Use the transition tool → in the State Machine section of the palette, and click on the initial junction to start connecting the transition.



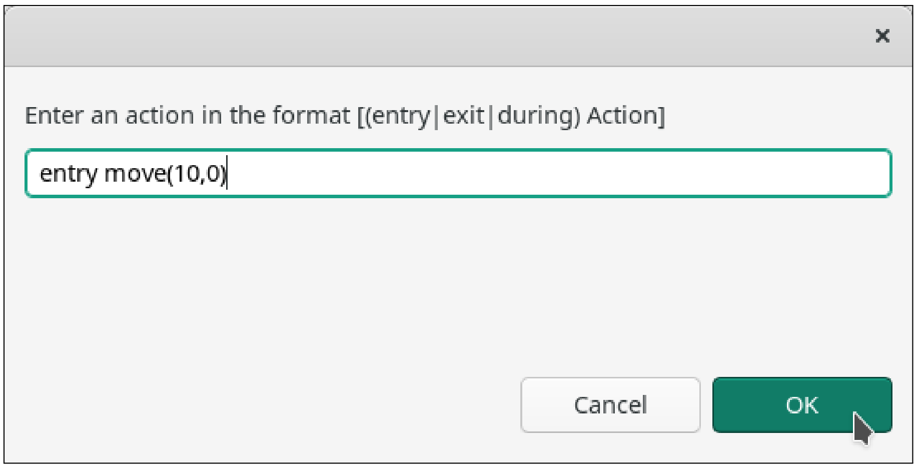
9. Next, click on the state to complete the transition.



10. Add an entry action to the state `Moving` using the action tool 🅰 in the State Machine section of the palette.



11. RoboTool will open a dialog window to input the action. A syntax hint is provided on the title of the dialog window, and syntax errors are shown below the text field. Add the entry action `entry move(10,0)` and click `OK`.

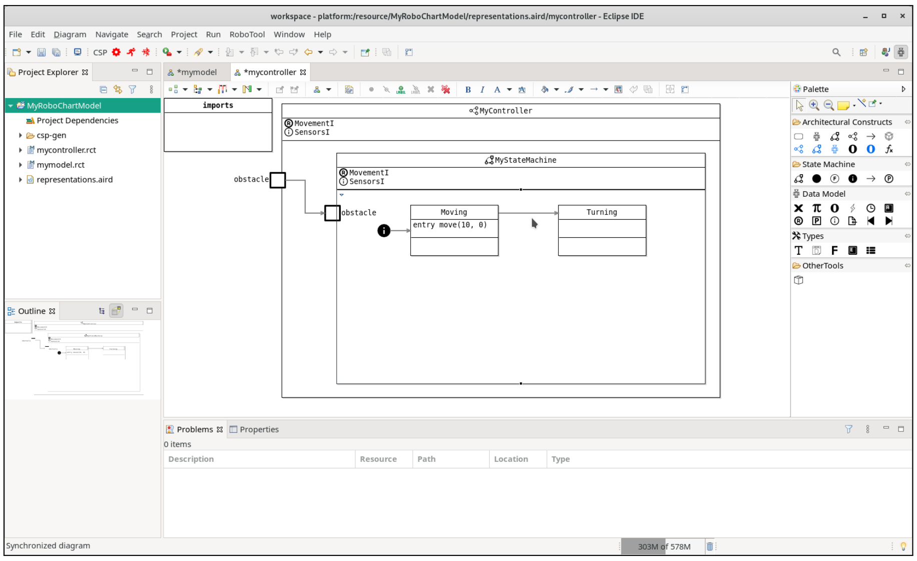


Enter an action in the format [(entry|exit|during) Action]

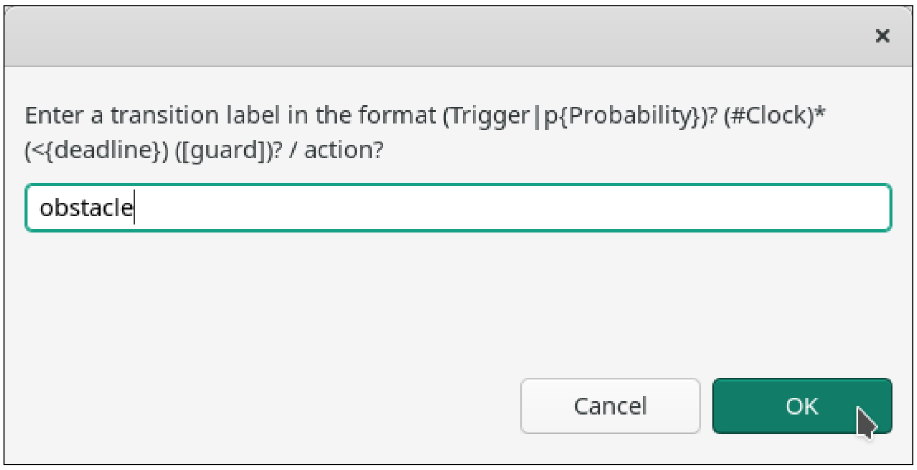
entry move(10,0)

Cancel OK

12. Add a second state `Turning` and a transition from `Moving` to `Turning`. Double click the transition to open a dialog windows to edit the transition label.



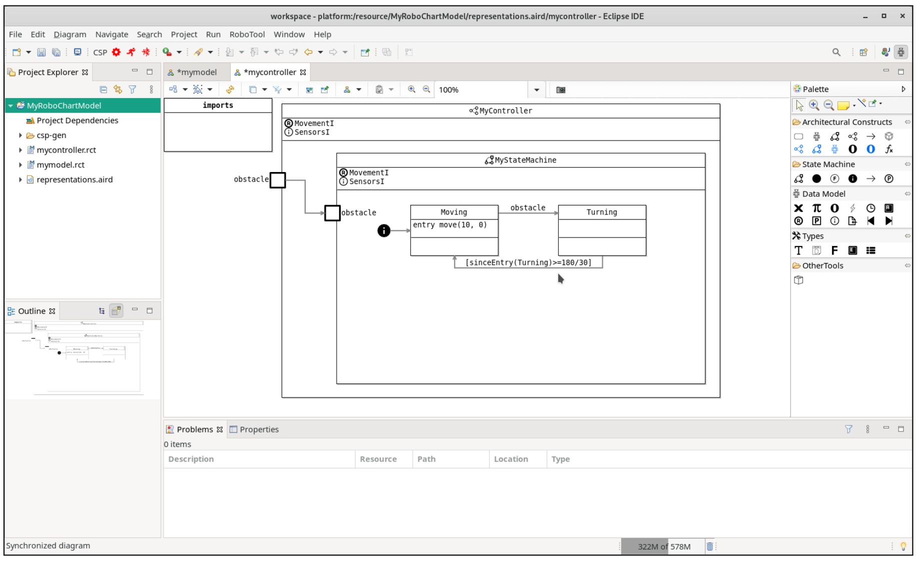
13. The dialog box provides a description of the syntax of labels and also any syntax errors. Define the trigger of the transition as the event `obstacle` and click `OK`.



Enter a transition label in the format (Trigger|p{Probability})? (#Clock)* (<{deadline}) ([guard])? / action?

obstacle

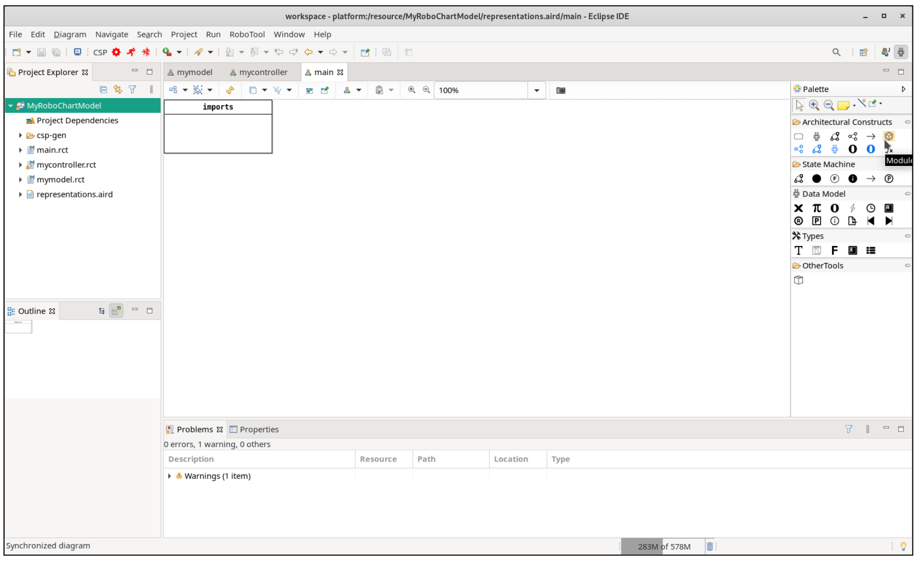
14. Finally, add a transition from `Turning` to `Moving`. Add a label to the transition specifying its condition as `[sinceEntry(Turning)>=180/30]`, which enables the transition as soon as 6 time units have elapsed since entering the state `Turning`.



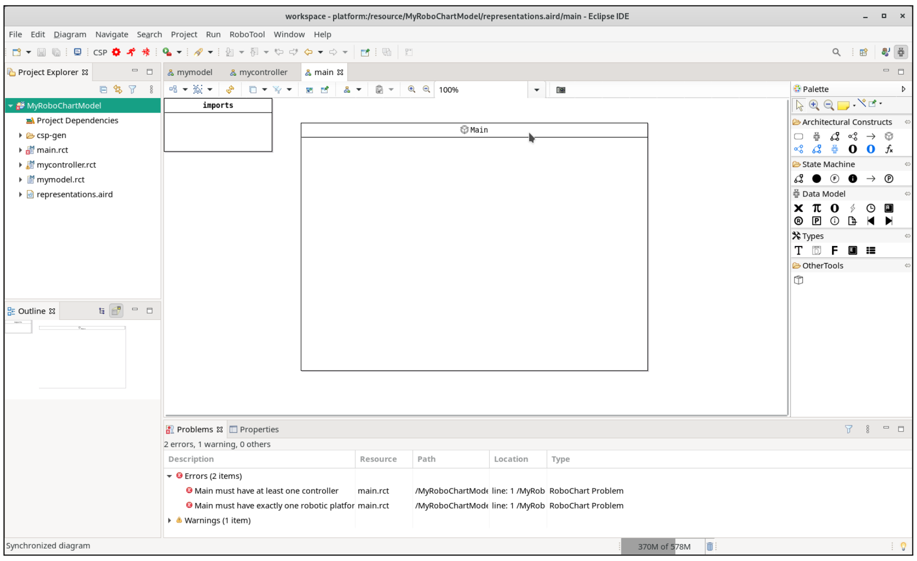
2.5 Creating Modules

Modules are the top level constructs of RoboChart, and they associate one or more controllers with exactly one robotic platform. Below are the steps to create a module.

1. Select the module tool in the Architectural Constructs section of the palette.



2. Click on a new created diagram, and press F2 to rename the module[2].

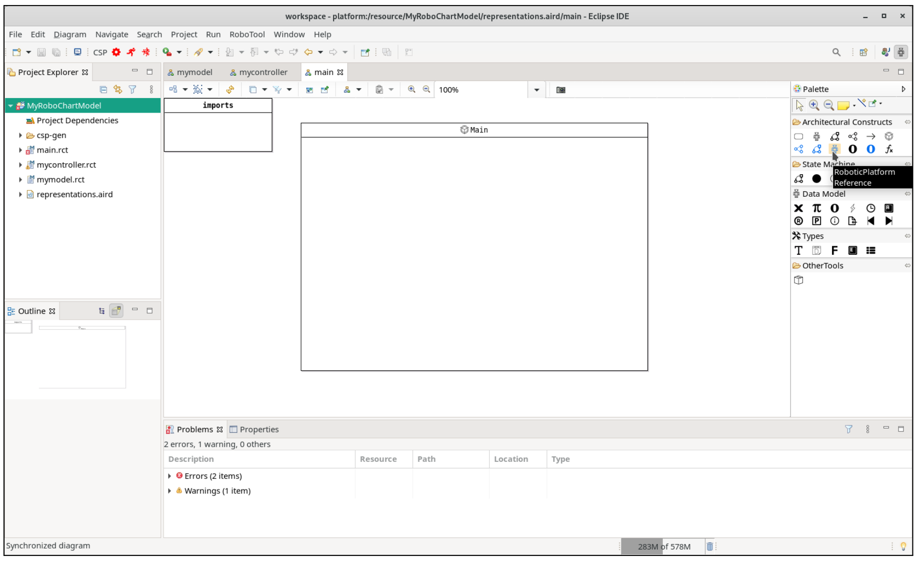


Problems | Properties

2 errors, 1 warning, 0 others

Description	Resource	Path	Location	Type
Errors (2 items)				
Main must have at least one controller	main.rct	/MyRoboChartMode	line: 1 /MyRob	RoboChart Problem
Main must have exactly one robotic platfor	main.rct	/MyRoboChartMode	line: 1 /MyRob	RoboChart Problem
Warnings (1 item)				

3. Select the robotic platform reference tool 🤖 (blue icon) in the Architectural Constructs section of the palette, and click on the module.

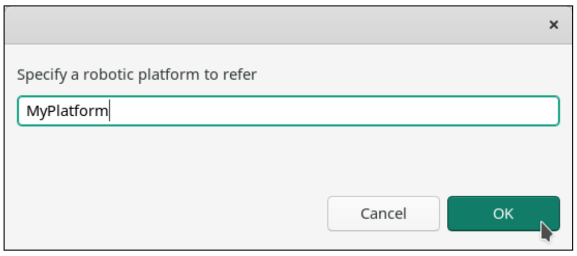


Problems | Properties

2 errors, 1 warning, 0 others

Description	Resource	Path	Location	Type
Errors (2 items)				
Warnings (1 item)				

4. RoboTool will open a dialog window to input the name of a robotic platform. Input the name of the platform created in Section 2.2 and click `OK`.

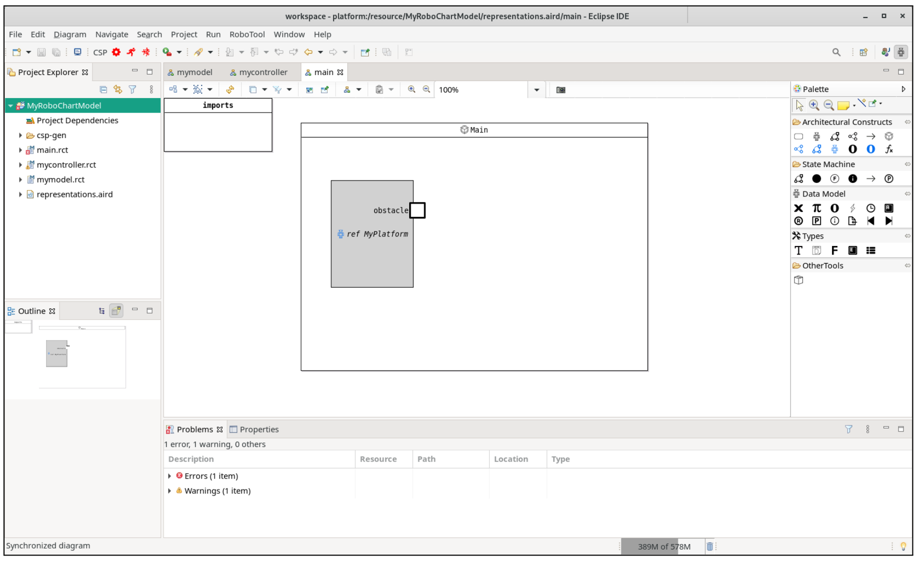


Specify a robotic platform to refer

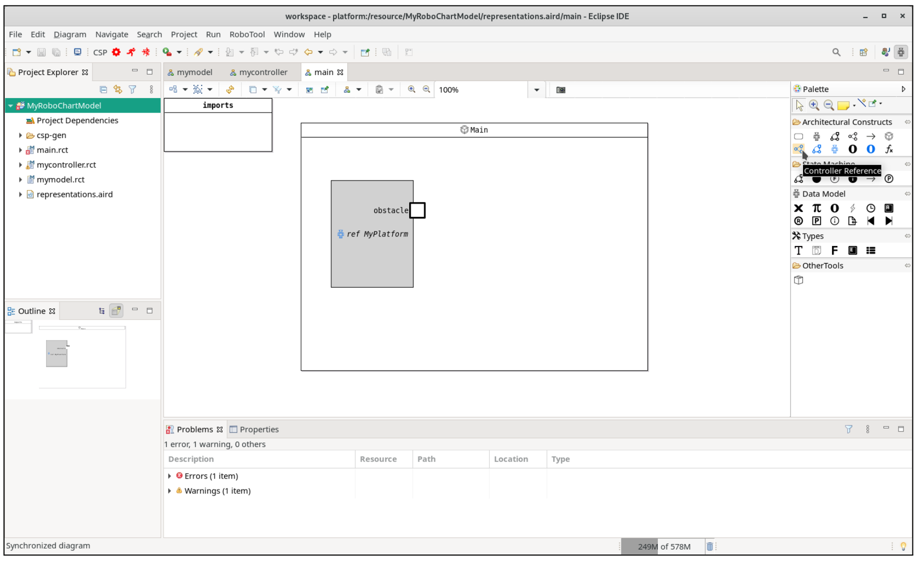
MyPlatform

[2]The errors shown in the problems tab at the bottom of the window occur because a module must have one robotic platform and at least one controller. These errors are resolved in the remaining steps.

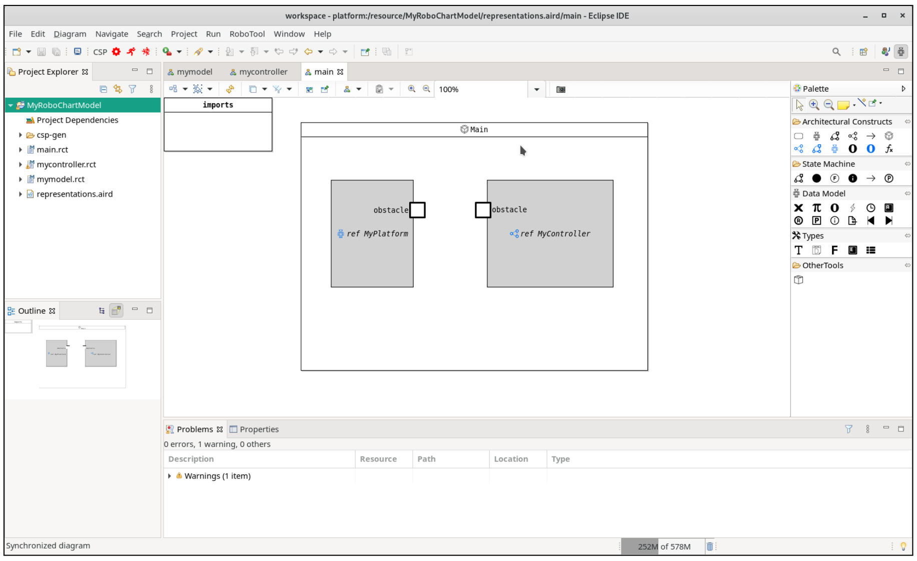
5. The resulting module shows the reference with a gray background and blue icon.



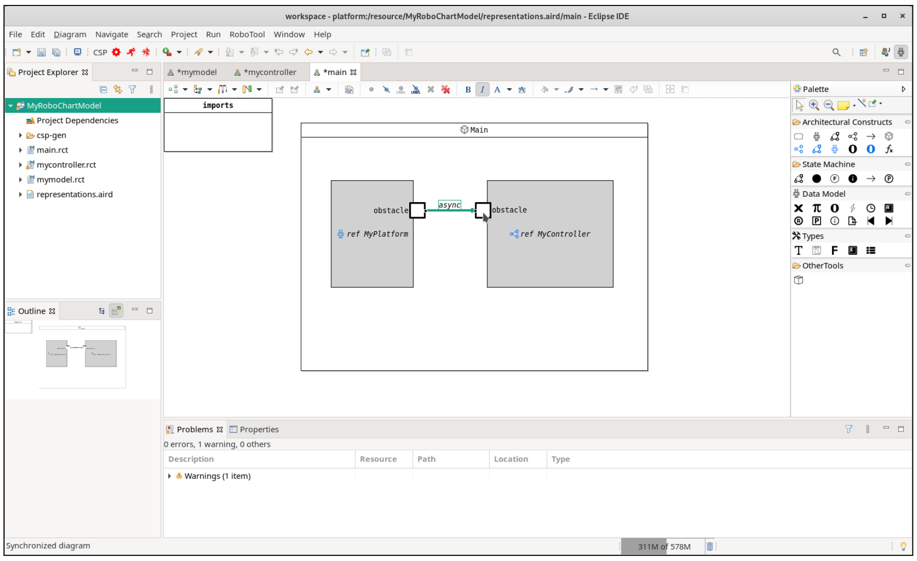
6. Select the controller reference tool in the Architectural Constructs section of the palette.



7. Click on the module, input the name of the controller created in Section 2.3 in the dialog box, and click `OK`.



8. Finally, use the connection tool → in the Architectural Constructs section of the palette to connect the obstacle event of the robotic platform to the obstacle event of the controller.



2.6 Exercises

Exercise 2.1 The alpha algorithm is a simple aggregation algorithm in swarm robotics [2].

A simulation of this algorithm can be found in `www.youtube.com/watch?v=8A7454VVhys`.

The basic algorithm is very simple:

- The default behaviour of a robot is forward motion.
- While moving each robot periodically sends an "Are you there?" message. It will receive "Yes, I am here" messages only from those robots that are in range, namely its neighbours.
- If the number of a robot's neighbours should fall below the threshold α then it assumes it is moving out of the swarm and will execute a 180° turn.

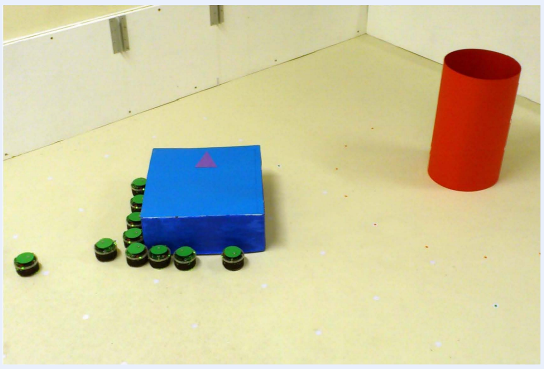
- When the number of neighbours rises above α (when the swarm is regained) the robot then executes a random turn. This is to avoid the swarm simply collapsing in on itself.

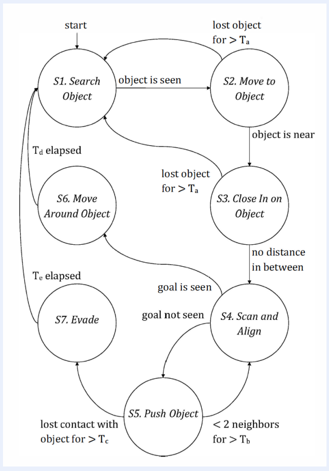
Complete the following tasks to model a robot that implements the alpha algorithm:

a) Define the robotic platform. *Suggestion: the fewer requirements on the platform, the more actual platforms can be used to implement the system.*
b) Define the module: controller(s) and connections to platform. *Suggestion: create separate controllers for separate requirements.*
c) Define a state machine to model movement control.
d) Define a state machine to model neighbour detection.

■

Exercise 2.2 In this exercise, you will model the control software of a simple robot that by itself cannot push an object, but when combined in a swarm can achieve the desired goal of transporting a larger object.


A group of robots transporting an object (blue box) towards a goal (red cylinder) [4].

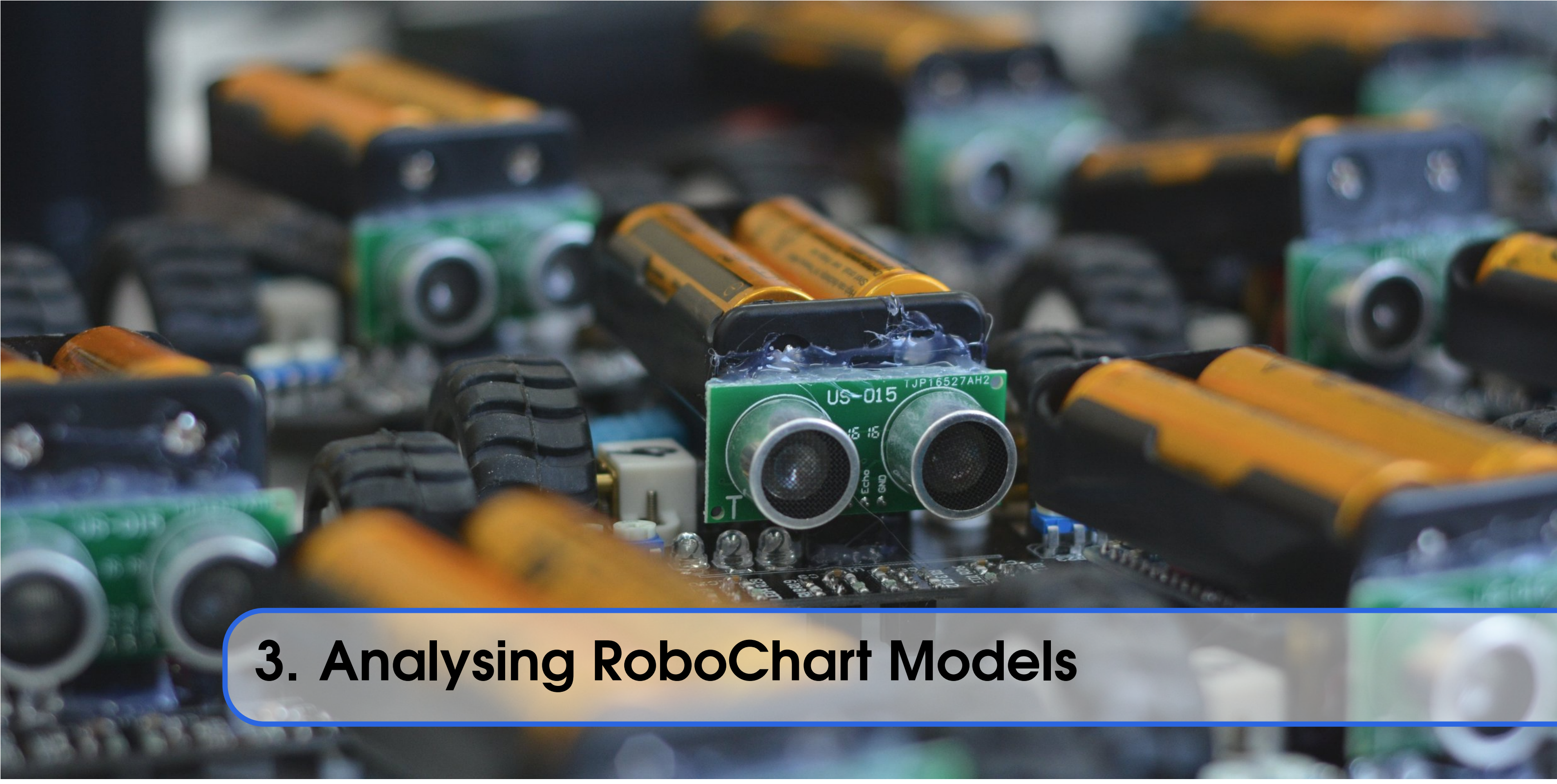


An informal state machine account of the solution [4].

Construct a RoboChart model that implements the transporter robot:

a) Define the robotic platform based on the events, variables, clocks and operations used in the state machine.
b) Rewrite the state machine above in RoboChart.
c) Define the module: controller(s) and connections to platform.

■



3. Analysing RoboChart Models

The semantics of RoboChart is formalised in CSP, and RoboTool generates the semantics of well-formed model in the `csp-gen` folder. You can use the refinement checker FDR[7] to analyse your RoboChart models.

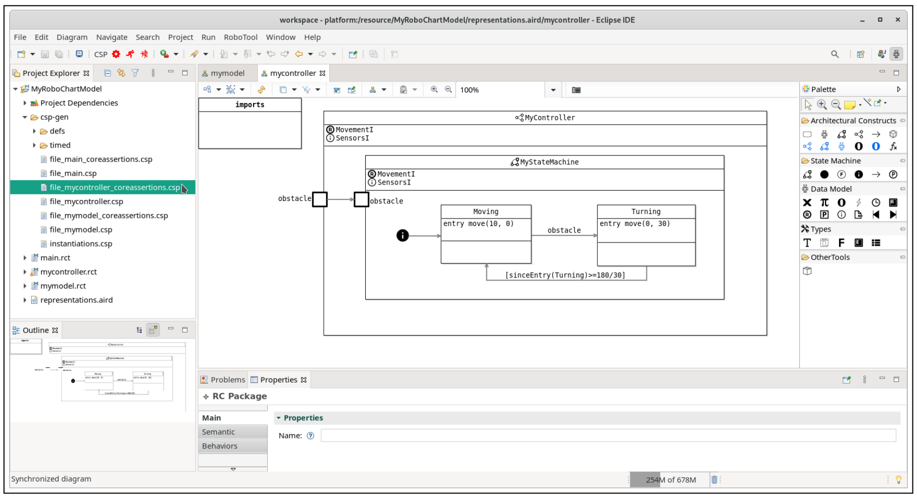
Section 3.1 illustrates the use of automatically generated assertions to check standard properties such as deadlock freedom and determinism, and Section 3.2 illustrates the use of a simple assertions DSL to specify custom properties.

3.1 Checking core assertions

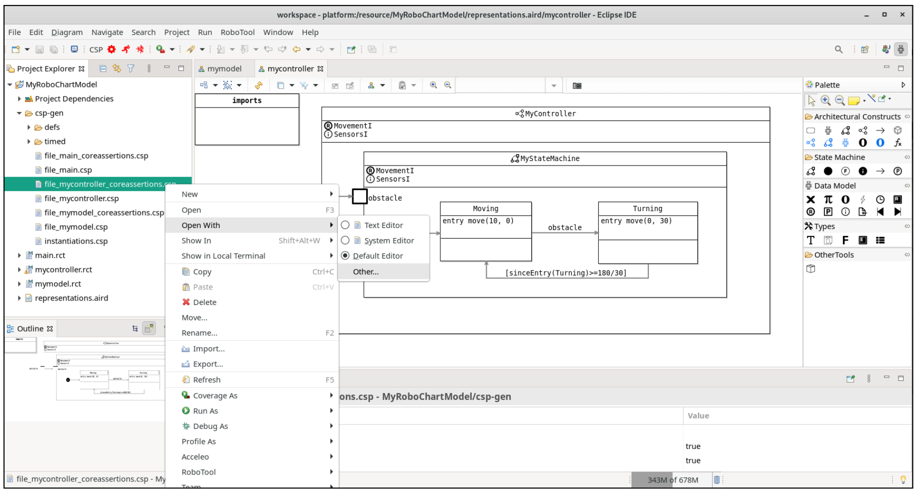
Along with the CSP semantics of a model, RoboTool automatically generates assertions to check standard properties such as deadlock freedom and determinism. These properties are specified in a file with the suffix `_coreassertions.csp`, and can be checked by FDR.[1]

[1]We recommend cleaning the project at least once using the menu item `Project > Clean...`. This is to avoid potentially outdated generated components.

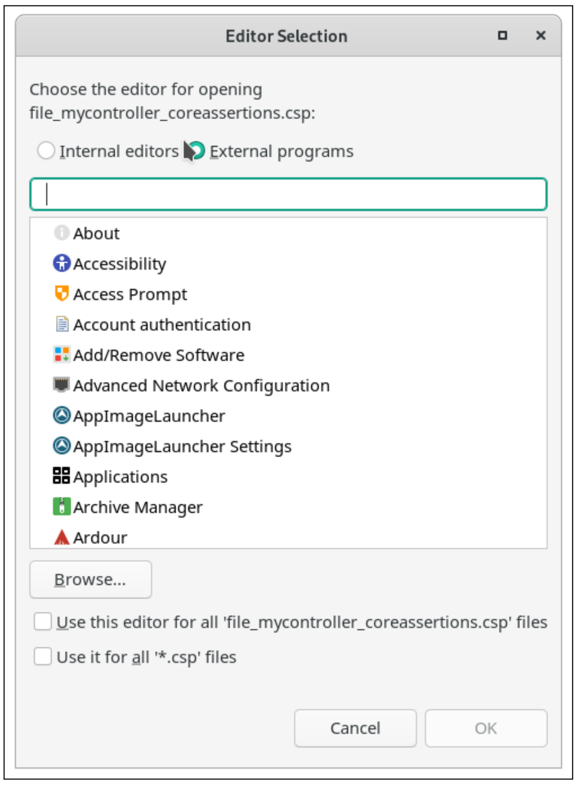
1. The core assertions for the controller created in the previous chapter are contained in the file `file_mycontroller_coreassertions.csp` in the `csp-gen` folder.



2. (Optional) In order to open the file in FDR directly from eclipse, select FDR as the default editor. Right-click the file, and select `[Open With > Other...]`.



3. In the Editor Selection dialog, select `External programs`.



Editor Selection

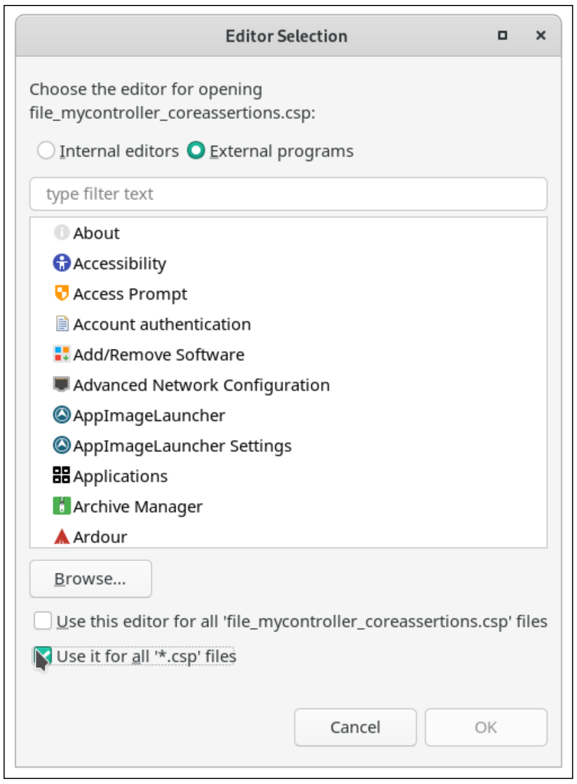
Choose the editor for opening
file_mycontroller_coreassertions.csp:

Internal editors [✓ External programs]

About
Accessibility
Access Prompt
Account authentication
Add/Remove Software
Advanced Network Configuration
AppImageLauncher
AppImageLauncher Settings
Applications
Archive Manager
Ardour

Browse...
Use this editor for all 'file_mycontroller_coreassertions.csp' files
Use it for all '*.csp' files
Cancel
OK

4. Check both “Use this editor for all FILENAME files” and “Use it for all ‘*.csp’ files”.



Editor Selection

Choose the editor for opening
file_mycontroller_coreassertions.csp:

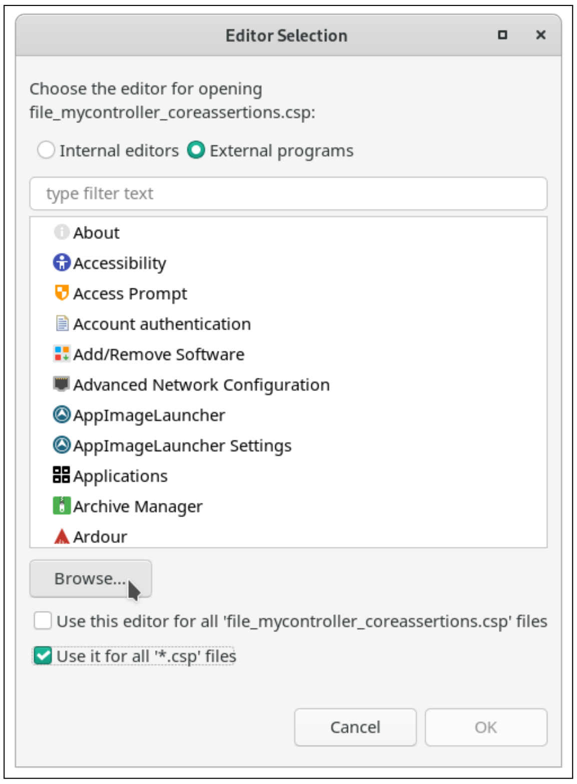
Internal editors [✓ External programs]

type filter text

About
Accessibility
Access Prompt
Account authentication
Add/Remove Software
Advanced Network Configuration
AppImageLauncher
AppImageLauncher Settings
Applications
Archive Manager
Ardour

Browse...
Use this editor for all 'file_mycontroller_coreassertions.csp' files
✓ Use it for all '*.csp' files
Cancel
OK

5. Click `Browse...` to select FDR as the editor.



Editor Selection

Choose the editor for opening file_mycontroller_coreassertions.csp:

Internal editors [✓ External programs]

type filter text

- About
- Accessibility
- Access Prompt
- Account authentication
- Add/Remove Software
- Advanced Network Configuration
- AppImageLauncher
- AppImageLauncher Settings
- Applications
- Archive Manager
- Ardour

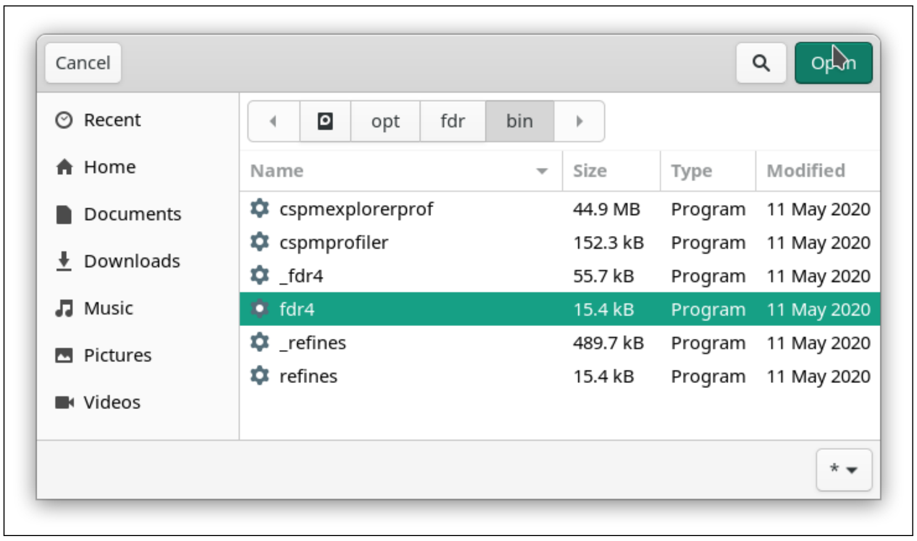
Browse...

☐ Use this editor for all 'file_mycontroller_coreassertions.csp' files

✓ Use it for all '*.csp' files

Cancel OK

6. Find the FDR4 executable, and click `OK`.



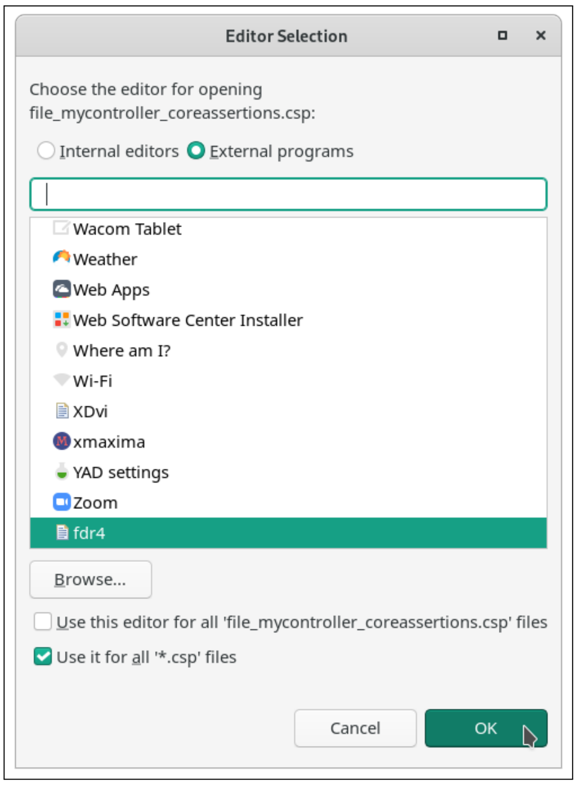
Cancel Open

Recent, Home, Documents, Downloads, Music, Pictures, Videos

opt / fdr / bin

Name	Size	Type	Modified
cspmexplorerprof	44.9 MB	Program	11 May 2020
cspmprofiler	152.3 kB	Program	11 May 2020
_fdr4	55.7 kB	Program	11 May 2020
fdr4	15.4 kB	Program	11 May 2020
_refines	489.7 kB	Program	11 May 2020
refines	15.4 kB	Program	11 May 2020

7. Make sure FDR4 is selected in the Editor Selection dialog, and click `OK`.



Editor Selection

Choose the editor for opening file_mycontroller_coreassertions.csp:

Internal editors [✓ External programs]

- Wacom Tablet
- Weather
- Web Apps
- Web Software Center Installer
- Where am I?
- Wi-Fi
- XDvi
- xmaxima
- YAD settings
- Zoom
- [✓ fdr4]

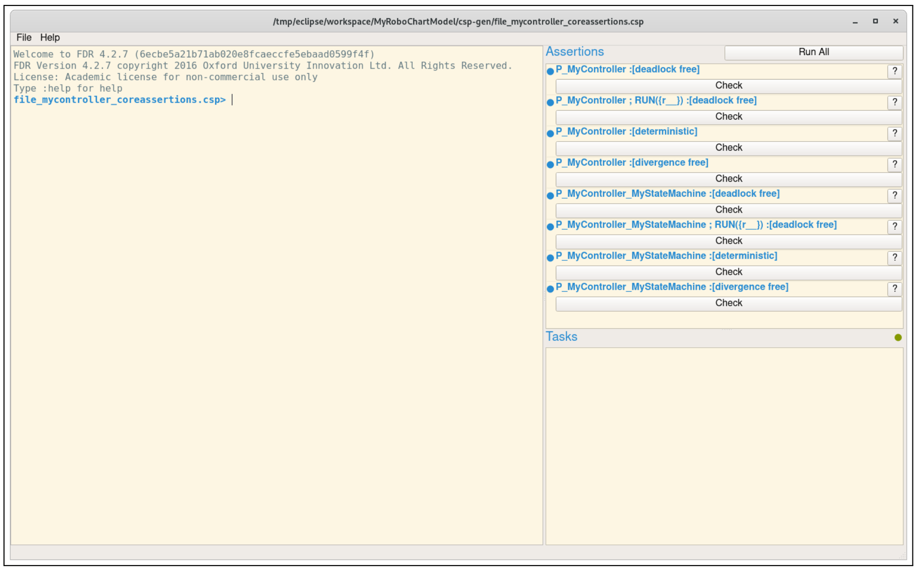
Browse...

☐ Use this editor for all 'file_mycontroller_coreassertions.csp' files

☑ Use it for all '*.csp' files

Cancel OK

8. The last step opens the FDR4 windows with all assertions loaded and displayed on the right-hand side panel.



/tmp/eclipse/workspace/MyRoboChartModel/csp-gen/file_mycontroller_coreassertions.csp

File Help

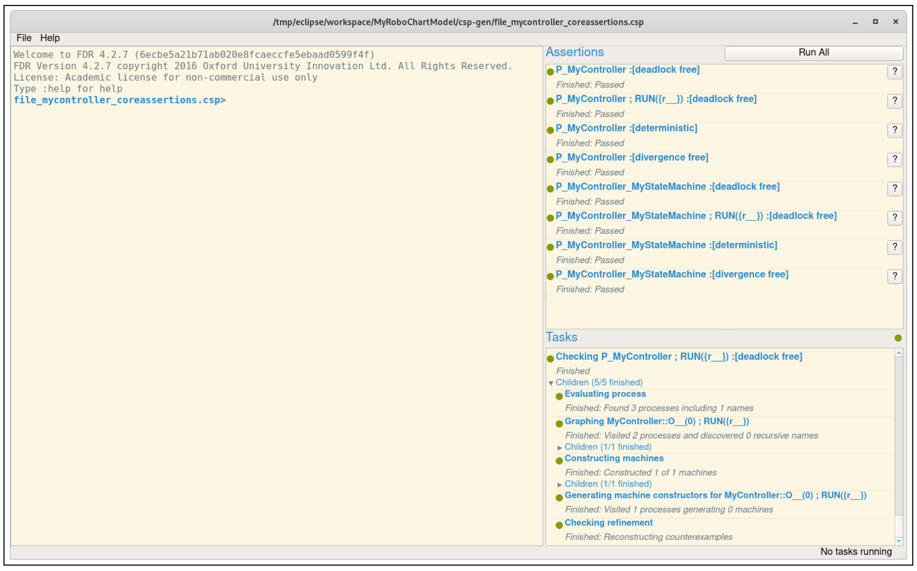
```
Welcome to FDR 4.2.7 (6ecbe5a21b71ab020e8fcaeccfe5ebaad0599f4f)
FDR Version 4.2.7 copyright 2016 Oxford University Innovation Ltd. All Rights Reserved.
License: Academic license for non-commercial use only
Type :help for help
file_mycontroller_coreassertions.csp> |
```

Assertions — Run All

- P_MyController :[deadlock free] — Check
- P_MyController ; RUN({r__}) :[deadlock free] — Check
- P_MyController :[deterministic] — Check
- P_MyController :[divergence free] — Check
- P_MyController_MyStateMachine :[deadlock free] — Check
- P_MyController_MyStateMachine ; RUN({r__}) :[deadlock free] — Check
- P_MyController_MyStateMachine :[deterministic] — Check
- P_MyController_MyStateMachine :[divergence free] — Check

Tasks

9. Click the `Run All` button at the top-right corner, and wait for the checks to finish. Alternatively, click each `Check` button to run each assertion separately.

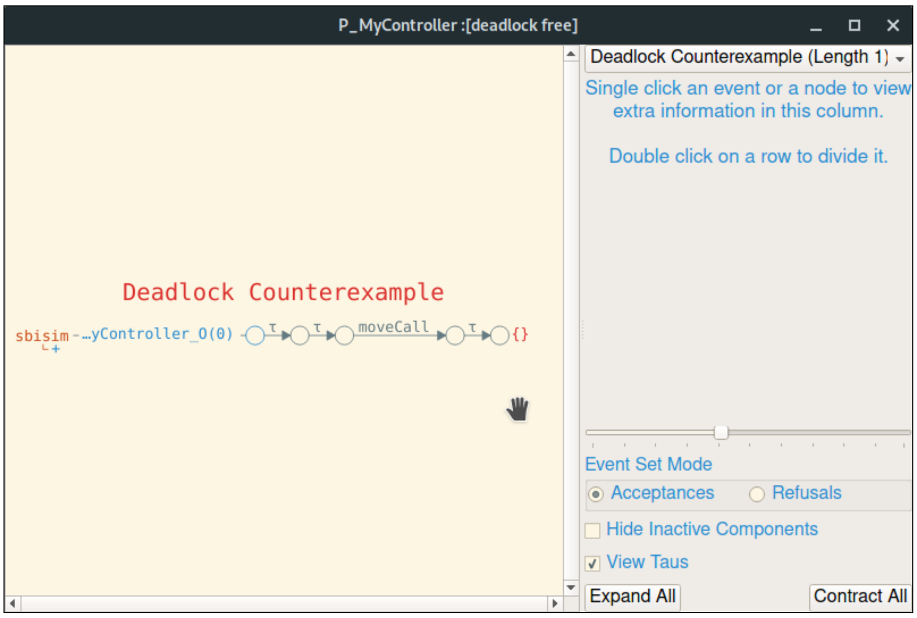

/tmp/eclipse/workspace/MyRoboChartModel/csp-gen/file_mycontroller_coreassertions.csp
File
Help
Assertions
Run All
Tasks

```
Welcome to FDR 4.2.7 (6ecbe5a21b71ab020e8fcaeccfe5ebaad0599f4f)
FDR Version 4.2.7 copyright 2016 Oxford University Innovation Ltd. All Rights Reserved.
License: Academic license for non-commercial use only
Type :help for help
file_mycontroller_coreassertions.csp>
```

```
● P_MyController :[deadlock free]
  Finished: Passed
● P_MyController ; RUN({r__}) :[deadlock free]
  Finished: Passed
● P_MyController :[deterministic]
  Finished: Passed
● P_MyController :[divergence free]
  Finished: Passed
● P_MyController_MyStateMachine :[deadlock free]
  Finished: Passed
● P_MyController_MyStateMachine ; RUN({r__}) :[deadlock free]
  Finished: Passed
● P_MyController_MyStateMachine :[deterministic]
  Finished: Passed
● P_MyController_MyStateMachine :[divergence free]
  Finished: Passed
```

```
● Checking P_MyController ; RUN({r__}) :[deadlock free]
  Finished
▾ Children (5/5 finished)
  ● Evaluating process
    Finished: Found 3 processes including 1 names
  ● Graphing MyController::O__(0) ; RUN({r__})
    Finished: Visited 2 processes and discovered 0 recursive names
  ▸ Children (1/1 finished)
  ● Constructing machines
    Finished: Constructed 1 of 1 machines
  ▸ Children (1/1 finished)
  ● Generating machine constructors for MyController::O__(0) ; RUN({r__})
    Finished: Visited 1 processes generating 0 machines
  ● Checking refinement
    Finished: Reconstructing counterexamples
No tasks running
```

10. If any of the (positive) assertions fail, a counter example is produced. It can be viewed by clicking the `Debug` button of the assertion.



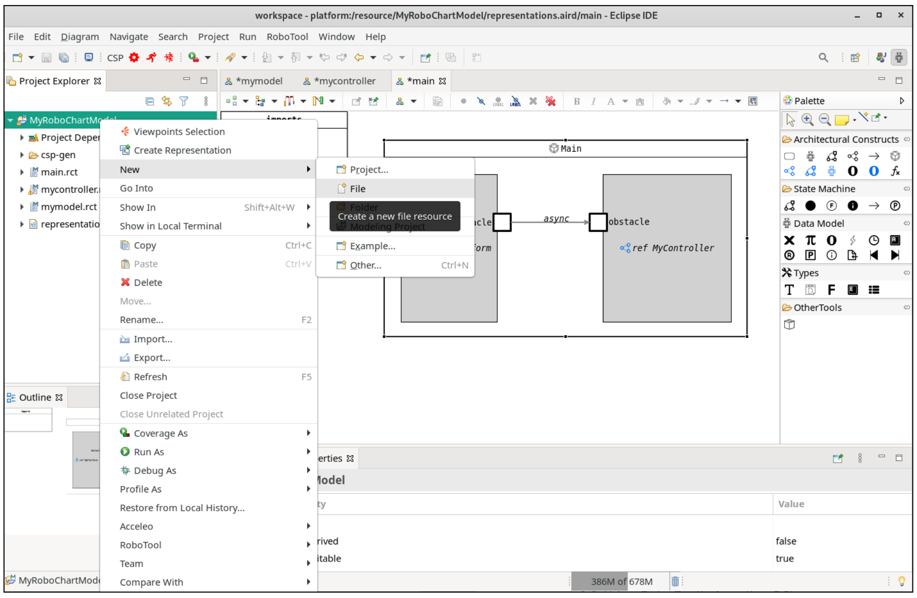
Single click an event or a node to view extra information in this column.

Double click on a row to divide it.

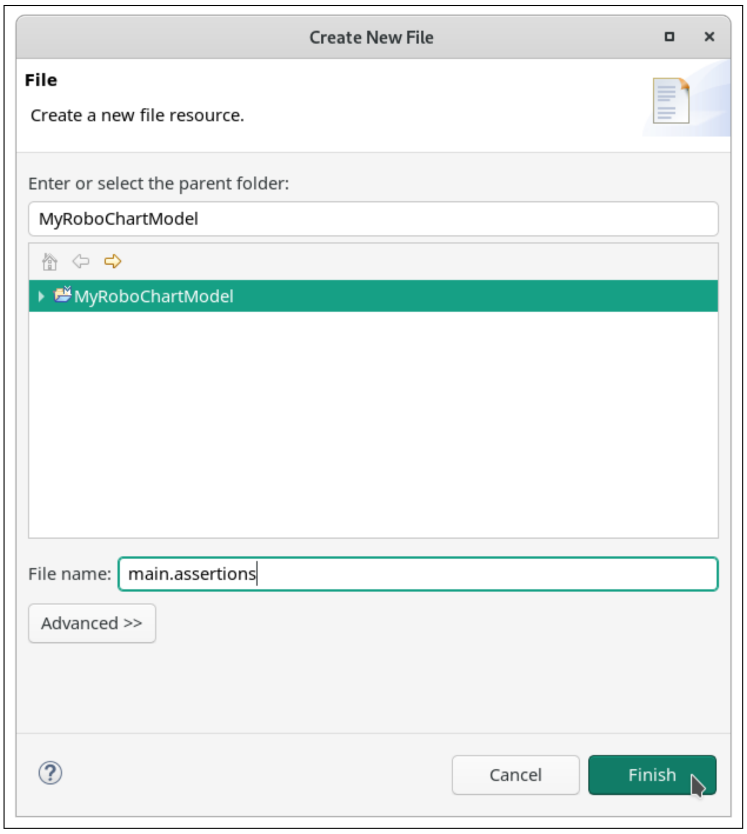
3.2 Using RoboTool's Assertion DSL

RoboTool also provides a simple text editor for an assertion DSL, which includes syntax highlighting, auto-completion, and error feedback. The DSL helps you write simple assertions such as deadlock freedom and refinement without requiring knowledge of the naming conventions of our semantics. More complex properties can be specified in CSP within special environments, but this requires an understanding of the structure and naming conventions of the RoboChart semantics.

1. Create a new file by right-clicking the project, and selecting `[New > File]`.



2. Name the file with the `.assertions` extension, and click `OK`.



File

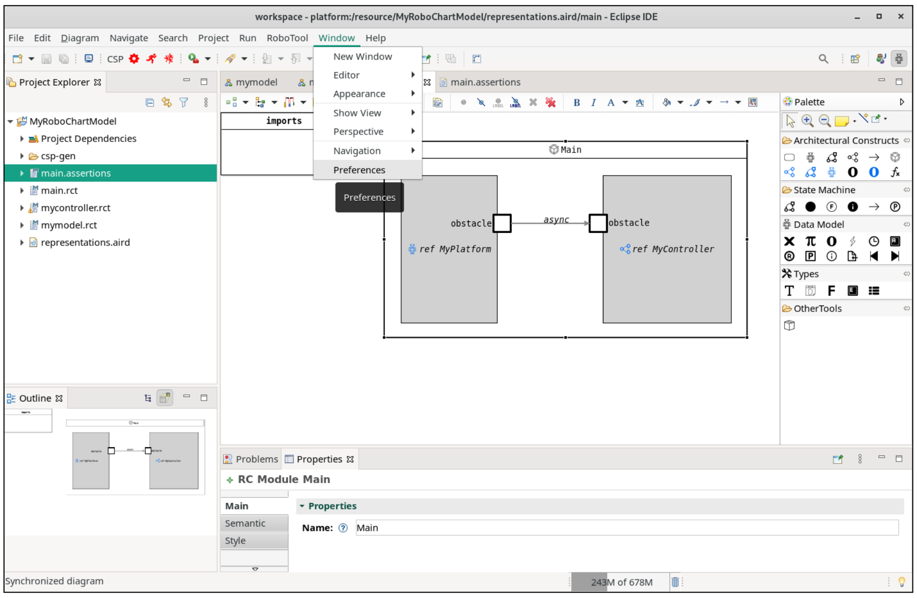
Create a new file resource.

Enter or select the parent folder: MyRoboChartModel

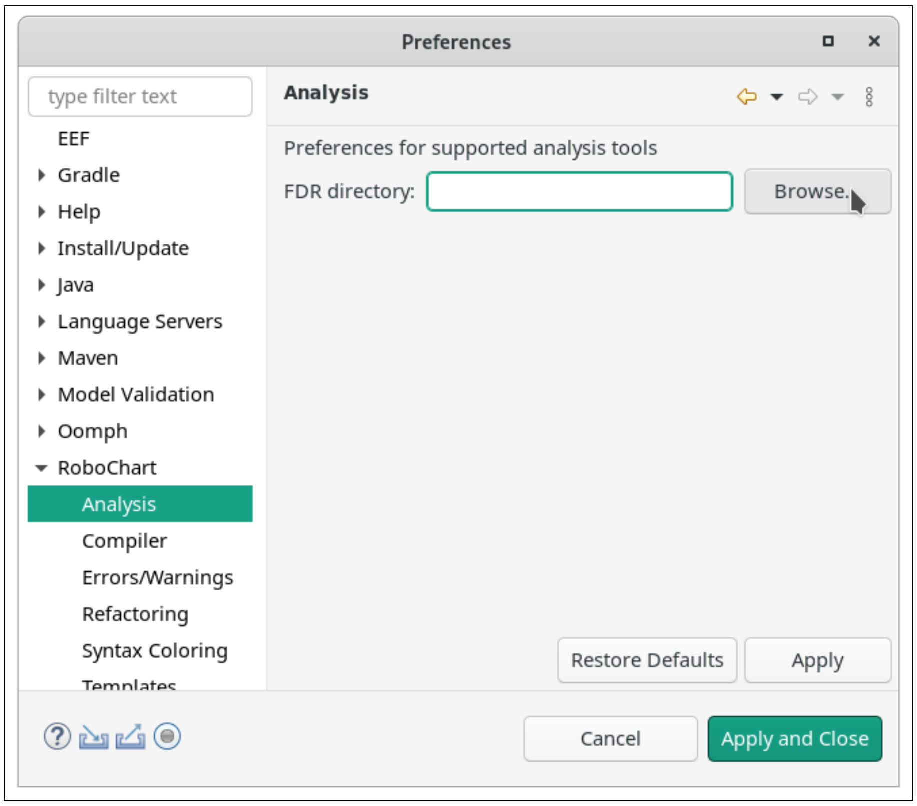
MyRoboChartModel

File name: main.assertions

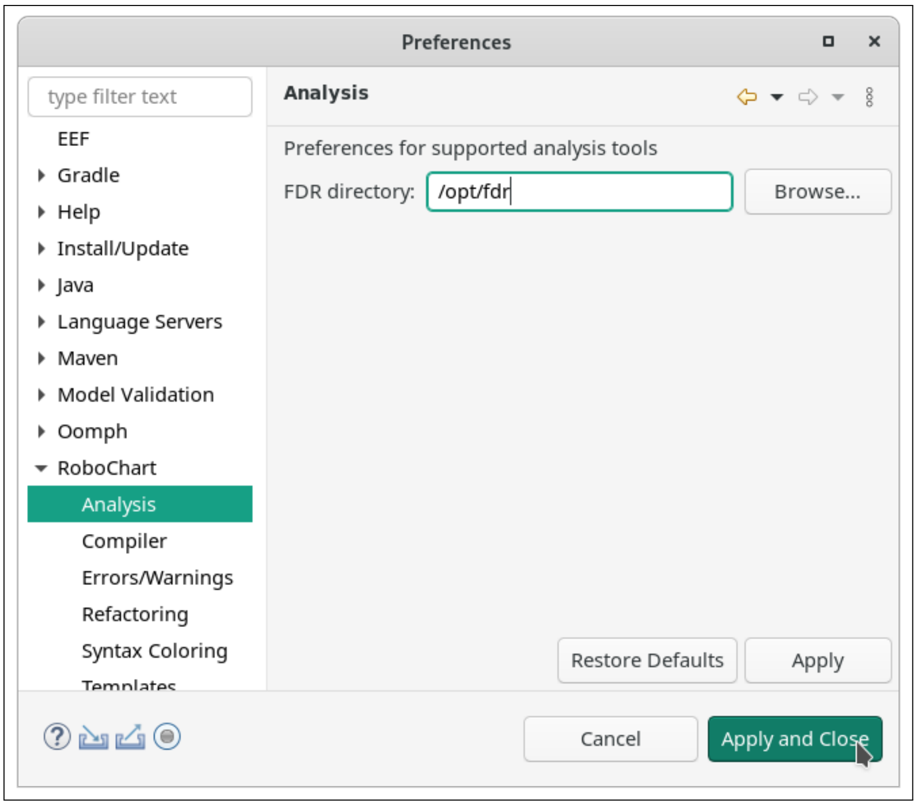
3. (Optional) If RoboTool has not yet been configure to find the FDR executable, select the menu item `[Window > Preferences]`.



4. (Optional) Select the `RoboChart > Analysis` item, and click `Browse...` to select the path to the installation directory of FDR.



5. (Optional) Click `Apply and Close` to apply the configuration.

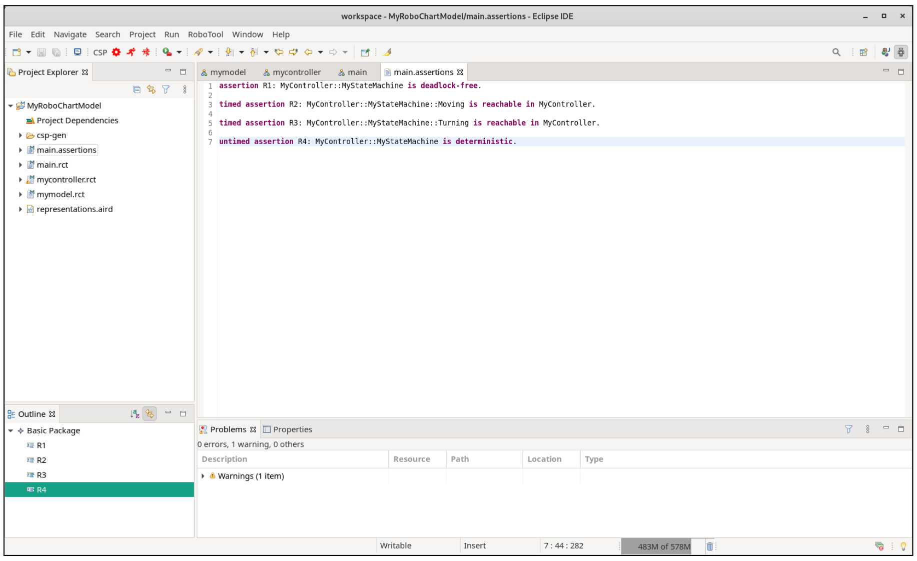


Analysis

Preferences for supported analysis tools

FDR directory: /opt/fdr

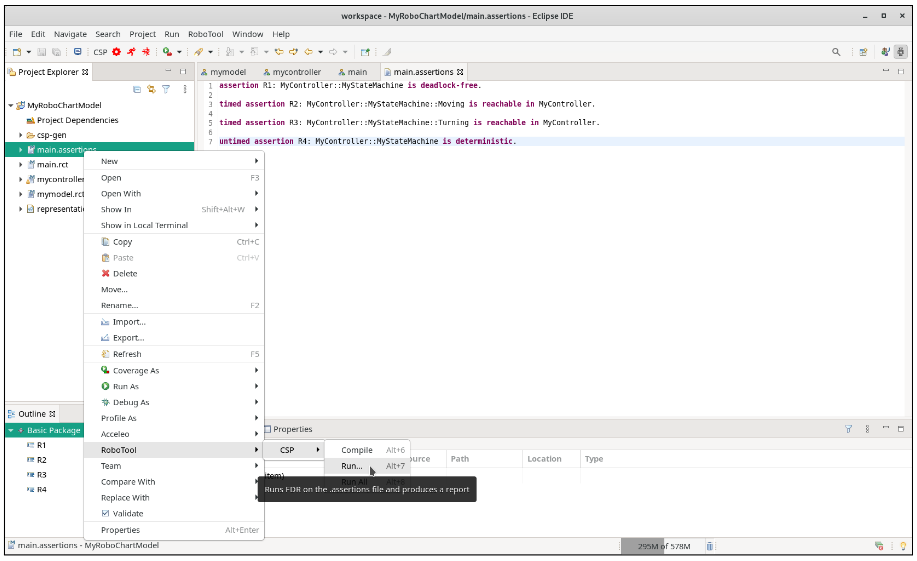
6. In the `.assertions` file, right your custom assertions. Notice that it may be necessary to use the qualified name of RoboChart elements, such as, `MyController::MyStateMachine`.



```
assertion R1: MyController::MyStateMachine is deadlock-free.

timed assertion R2: MyController::MyStateMachine::Moving is reachable in MyController.

timed assertion R3: MyController::MyStateMachine::Turning is reachable in MyController.

untimed assertion R4: MyController::MyStateMachine is deterministic.
```

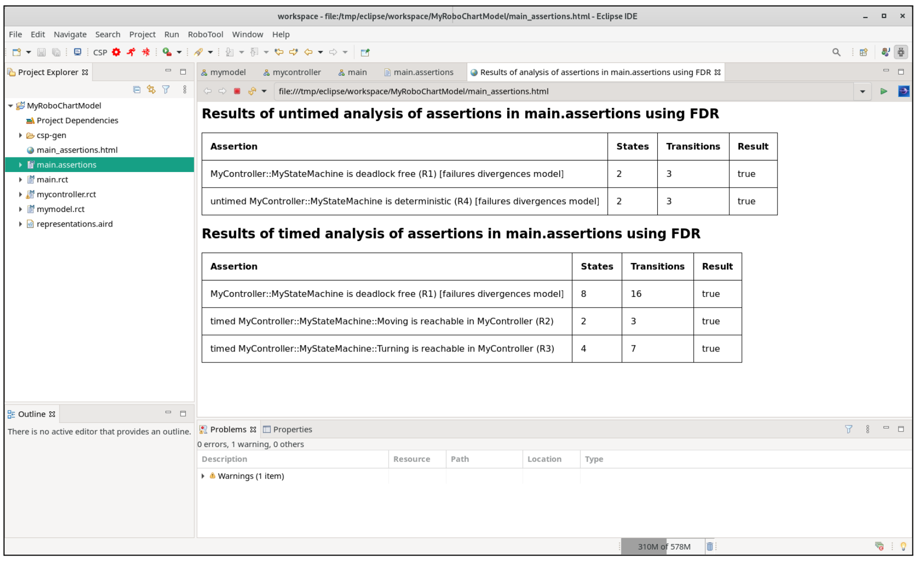
0 errors, 1 warning, 0 others

7. In order to verify the assertions, right-click the `.assertions` file, and select the `[RoboTool > CSP > Run...]` item.



```
assertion R1: MyController::MyStateMachine is deadlock-free.

timed assertion R2: MyController::MyStateMachine::Moving is reachable in MyController.

timed assertion R3: MyController::MyStateMachine::Turning is reachable in MyController.

untimed assertion R4: MyController::MyStateMachine is deterministic.
```

8. Provided there are no errors in the assertions or models, FDR checks the assertions in the background, and RoboTool summarises the result in the form of a report, which is automatically opened upon completion of the checks.



Results of untimed analysis of assertions in main.assertions using FDR

Assertion	States	Transitions	Result
MyController::MyStateMachine is deadlock free (R1) [failures divergences model]	2	3	true
untimed MyController::MyStateMachine is deterministic (R4) [failures divergences model]	2	3	true

Results of timed analysis of assertions in main.assertions using FDR

Assertion	States	Transitions	Result
MyController::MyStateMachine is deadlock free (R1) [failures divergences model]	8	16	true
timed MyController::MyStateMachine::Moving is reachable in MyController (R2)	2	3	true
timed MyController::MyStateMachine::Turning is reachable in MyController (R3)	4	7	true

0 errors, 1 warning, 0 others

workspace - file:/tmp/eclipse/workspace/MyRoboChartModel/main_assertions.html - Eclipse IDE
File Edit Navigate Search Project Run RoboTool Window Help
Project Explorer
MyRoboChartModel
Project Dependencies
csp-gen
main_assertions.html
main.assertions
main.rct
mycontroller.rct
mymodel.rct
representations.aird
mymodel
mycontroller
main
main.assertions
Results of analysis of assertions in main.assertions using FDR
file:///tmp/eclipse/workspace/MyRoboChartModel/main_assertions.html
Outline
There is no active editor that provides an outline.
Problems
Properties
Description
Resource
Path
Location
Type
Warnings (1 item)
310M of 578M

3.3 Exercises

Exercise 3.1 Check the core assertions for the model created in exercise 2.1.

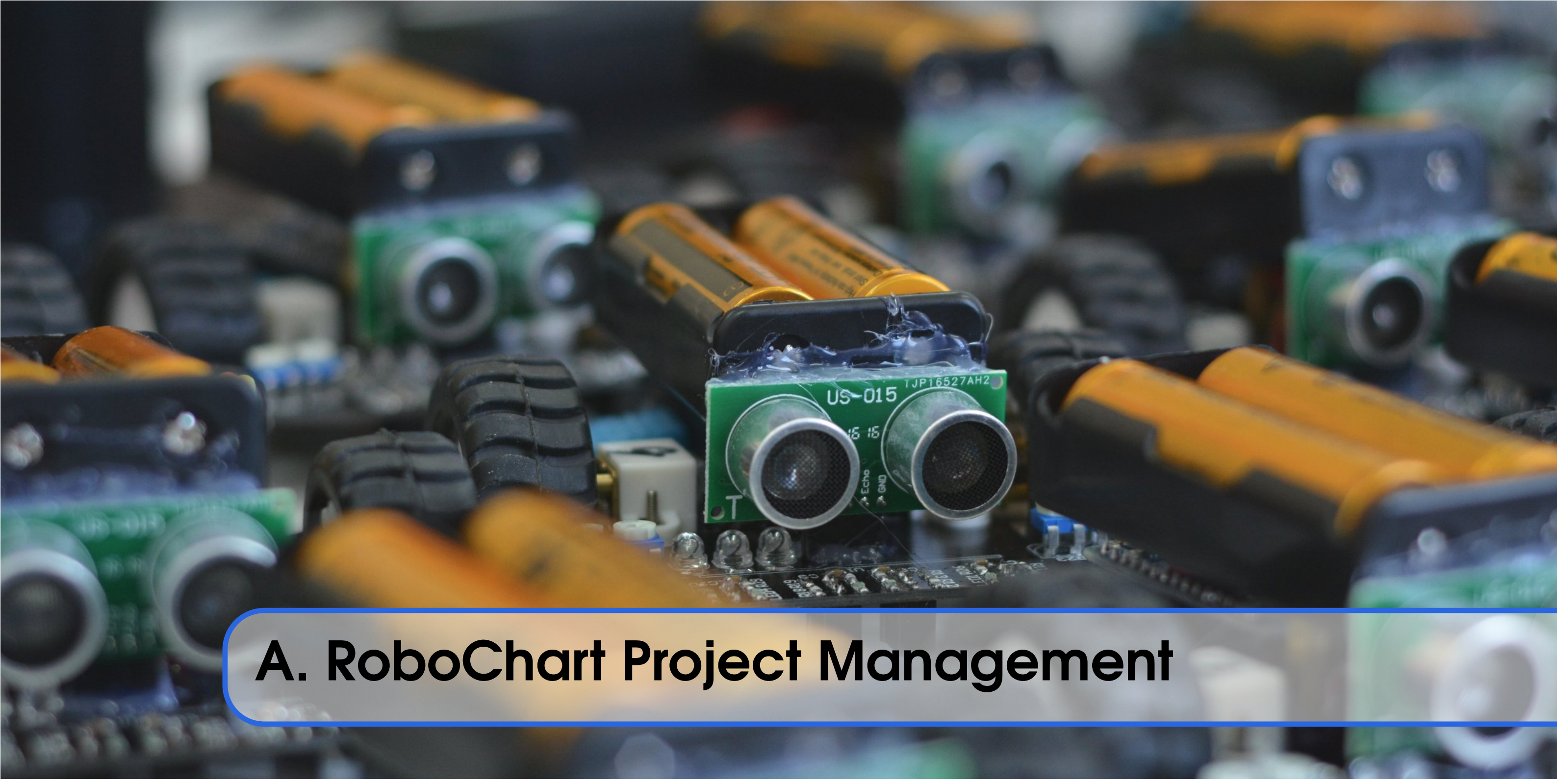
Exercise 3.2 Write custom assertions for the model created in exercise 2.1, and check them using RoboTool and FDR.

Exercise 3.3 Check the core assertions for the model created in exercise 2.2.

Exercise 3.4 Write custom assertions for the model created in exercise 2.2, and check them using RoboTool and FDR.

II

Appendices

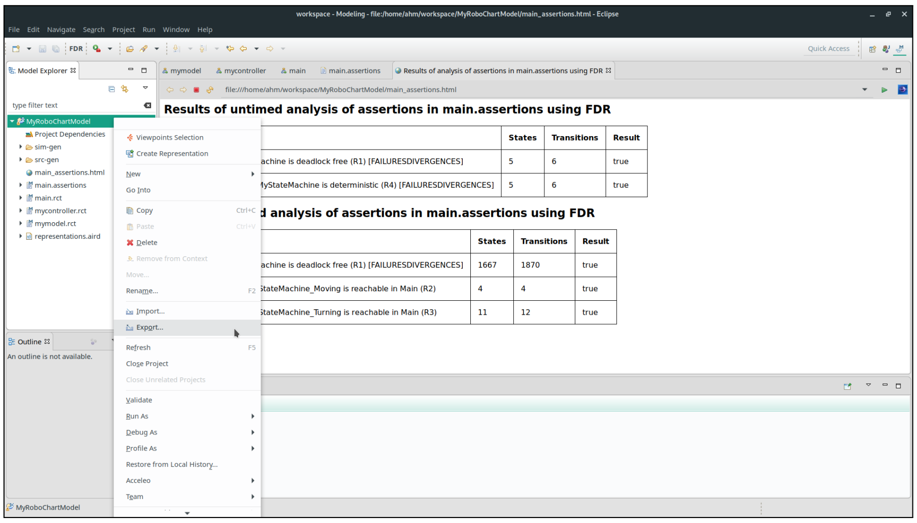


A. RoboChart Project Management

This appendix describes common tasks in managing RoboChart projects.

A.1 Exporting a RoboChart project

1. Right click on the project and select `Export...` item.



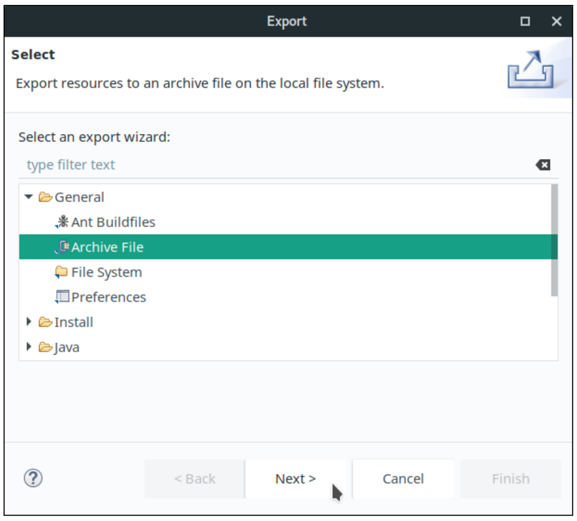
Results of untimed analysis of assertions in main.assertions using FDR

	States	Transitions	Result
achine is deadlock free (R1) [FAILURESDIVERGENCES]	5	6	true
lyStateMachine is deterministic (R4) [FAILURESDIVERGENCES]	5	6	true

d analysis of assertions in main.assertions using FDR

	States	Transitions	Result
achine is deadlock free (R1) [FAILURESDIVERGENCES]	1667	1870	true
StateMachine_Moving is reachable in Main (R2)	4	4	true
StateMachine_Turning is reachable in Main (R3)	11	12	true

2. Select the wizards [General > Archive File], and click Next.


Export

Select

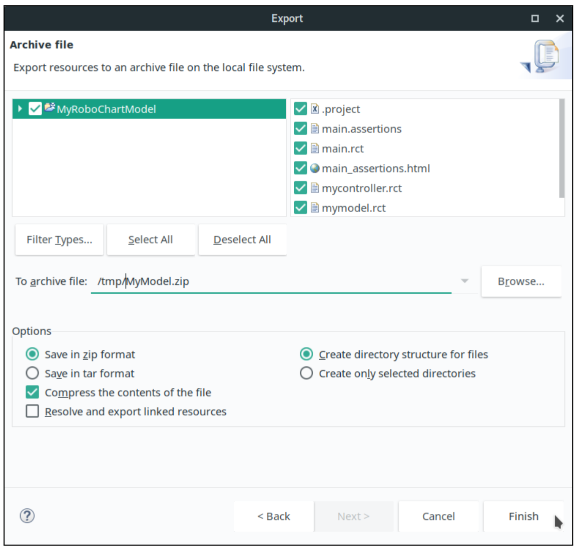
Export resources to an archive file on the local file system.

Select an export wizard:

type filter text

- General
  - Ant Buildfiles
  - Archive File
  - File System
  - Preferences
- Install
- Java

3. Select the path to the archive file, and click Finish.


Export

Archive file

Export resources to an archive file on the local file system.

[✓] MyRoboChartModel

[✓] .project
[✓] main.assertions
[✓] main.rct
[✓] main_assertions.html
[✓] mycontroller.rct
[✓] mymodel.rct

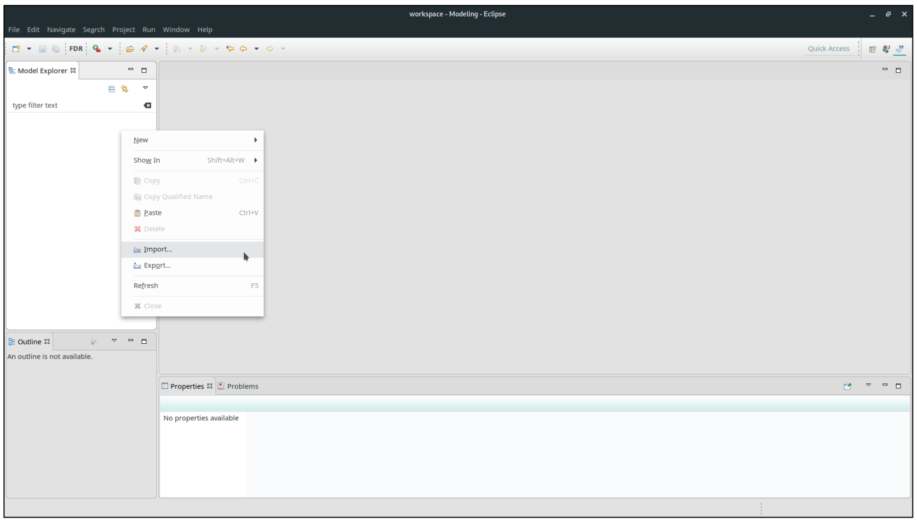
To archive file: /tmp/MyModel.zip

Options

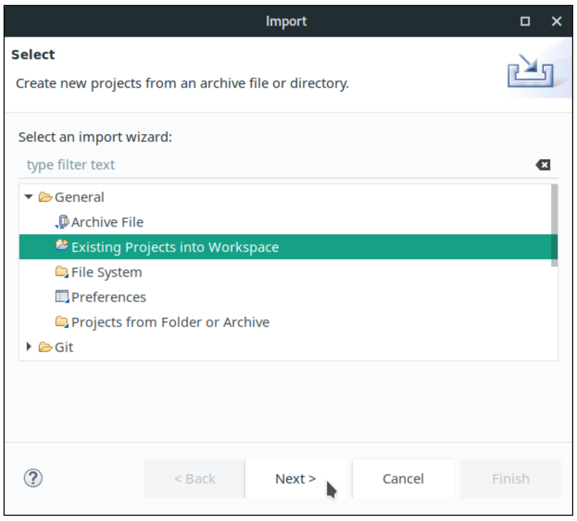
- (●) Save in zip format
- ( ) Save in tar format
- [✓] Compress the contents of the file
- [ ] Resolve and export linked resources
- (●) Create directory structure for files
- ( ) Create only selected directories

A.2 Importing a RoboChart project

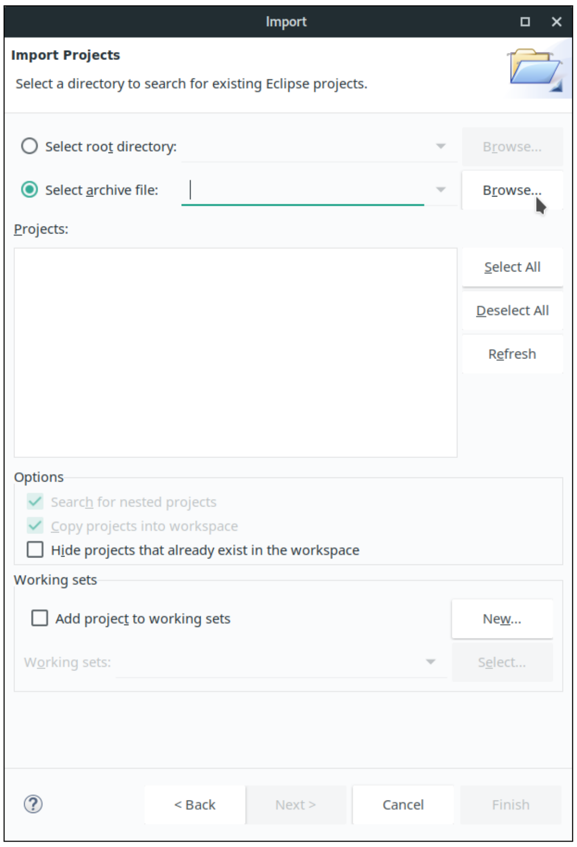
1. Right click on the `Model Explorer`, and select `Import...` item.



2. Select wizards `General > Existing Project into Workspace`, and click `Next`.



3. Check `Select archive file`, and click `Browse` to select the archive.



Import

Import Projects

Select a directory to search for existing Eclipse projects.

○ Select root directory:

● Select archive file:

Projects:

Options

✓ Search for nested projects

✓ Copy projects into workspace

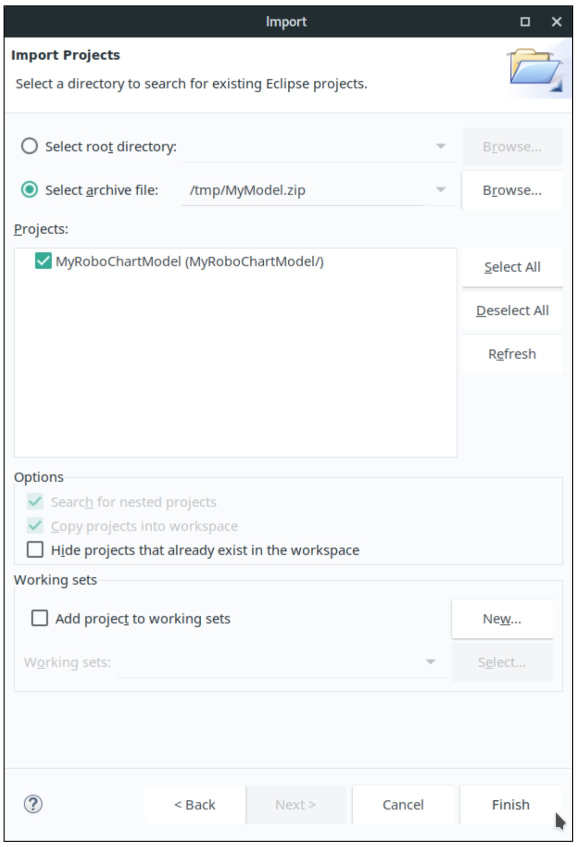
☐ Hide projects that already exist in the workspace

Working sets

☐ Add project to working sets

Working sets:

4. Select the project to import, ad click `Finish`.



Import

Import Projects

Select a directory to search for existing Eclipse projects.

○ Select root directory:

● Select archive file: /tmp/MyModel.zip

Projects:

✓ MyRoboChartModel (MyRoboChartModel/)

Options

✓ Search for nested projects

✓ Copy projects into workspace

☐ Hide projects that already exist in the workspace

Working sets

☐ Add project to working sets

Working sets:

A.3 Deleting a RoboChart package

Since a RoboChart package is associated with an `rct` file and a diagram (in the aird file), we need to delete both the `rct` file and the diagram[1].

1. In order to delete a RoboChart package, that is, the rct file and the associated diagrams:
   (a) Select the file, press `Delete` (or select delete on the context menu) and click `OK`;
   (b) Click the left arrows on the file *representation.aird*, the subitem *RoboChart* and the subitem RCPackage. The names of the diagrams associated with the deleted file will be in a lighter font. Select them, press `delete` (or right-click and click `Delete`) and confirm the deletion by clicking `OK`.

A.4 Editing a RoboChart diagram

1. Use the palette to the right to select objects to add to the diagram. For most objects (except connection) you can either drag and drop the object to the diagram, or select and click (if you keep `Ctrl` pressed, it is possible to click multiple times on the diagram to create multiple instances of the object). After creating a new component (e.g., type), make sure to save the diagram (press `Ctrl+S` or click `File > Save`) to guarantee the newly created element can be used.

   R The palette may not show all tools depending on the size of the screen. It is divided into category boxes that group similar tools. In order to explore the available tools you can click click a category box to close it, and click once more to fully expand it.

   It is also possible to use the arrows at the bottom and top of the category box (or the scroll wheel on your mouse) to scroll through the available tools.

   If you are familiar with the icons, it is possible to reduce the space taken by the tools by removing the text and showing only the icons. To do this, right click the palette and select `Layout > Icons Only`

2. To use connections, select the appropriate connection tool (e.g., transition), click the source node and then the target node[2];
3. To change labels (e.g., interface names, transition labels) there are a few options:
   (a) Select the item and start typing. This option will delete the original text; to avoid this use one of the next two options; or
   (b) Press `F2` and the text will become editable; or
   (c) Click once over the text and it will become editable.

   R Label editing has some limitations that are worth mentioning:
   (a) Not all labels editors are implemented yet (if you think label editing for a specific feature is missing, create an issue in the bitbucket issue tracker);
   (b) If you type a label with syntax errors, the edit will fail and the previous text will show (potentially empty text).

4. Most textual elements (variable declarations, operations, actions etc) are input through a pop-up window that provides a hint of the syntax and parses the input before creating the element. For example, when creating a new action in a state:
   (a) a pop-up windows will appear requesting the text of the action in the format `[(entry|exit|during) Action]`, that is, one of the keyword **entry**, **exit** or **during** followed by an action[3]

[1]We must delete the diagram associated with the RoboChart package, not the whole `aird` file.
[2]While you can use Ctrl+Z to undo changes to the diagram, it can create inconsistencies, so use it carefully.
[3]See the language reference manual for the concrete syntax of actions and other elements.

   i. if you only type `entry`, for instance, the box will show `Cannot parse` below the text box and keep the button `OK` disabled
   ii. if, on the other hand, you type `entry skip`, the tool parses the action correctly and enables the `OK` button. If you click `OK` the action will be added to the state.
5. To save a diagram, press `Ctrl+S` or click `File > Save`.

A.5 Deleting element from diagrams

To delete an object from the diagram, select it and press `delete`; avoid using the option `Right-click > Edit > Delete from Model` as it can make the diagram inconsistent.

Be careful when deleting elements:

1. In general, make sure they are not used elsewhere. For example, when deleting an event used as the trigger of a transition, the transition will point to a non-existent event; make sure to remove the reference first;
2. We have implemented controlled deletions for Interfaces and Types, but not for other elements[4].
   (a) When deleting an interface, the tool will delete all required and provided references and try to replace call to the operations in the interface by other operations of the same name in scope; if this is not possible, operation calls are converted into `skip`
   (b) When deleting a type, the tool will look for uses of that type, if none are found, the type is deleted. Otherwise, a list of types is offered to the user to replace the deleted type.

A.6 Exporting diagrams as figures

It is possible to export the diagrams as figure[5].

To export all the diagrams at the same time:

1. Click the arrow to the left of the file `representations.aird` to expand it.
2. Expand the items `RoboChart` and `Package`.
3. Use `ctrl+left click` to select multiple diagrams
4. Right click the selection and click `Export representations as images`
5. A popup window will appear allowing you to select the target directory and the image format; after selecting the directory and the format, click `OK`.

To export a specific diagram, follow these steps:

1. Open the diagram (if it not already open)
2. If any diagram elements are selected, deselect them by pressing `Esc` or clicking in the background of the diagram
3. The toolbar at the top of the diagram contains a camera icon 📷 ; click it
4. A popup window will appear allowing you to select the target file and the image format; after selecting the file and the format, click `OK`.

If you right click `representation.aird`, an option `Export representations as images` is available. While this option works and generates images for all diagrams, it may have a side effect of closing all open diagrams, and making the project tree inconsistent. In this case, collapse the project by clicking the arrow to the left of the project name, and expand it again. This should refresh the project and offer to reopen the diagrams.

[4] If and when I implement more cases, I'll add them to this manual.
[5] The available formats are JPG, PNG, SVG, BMP and GIF.

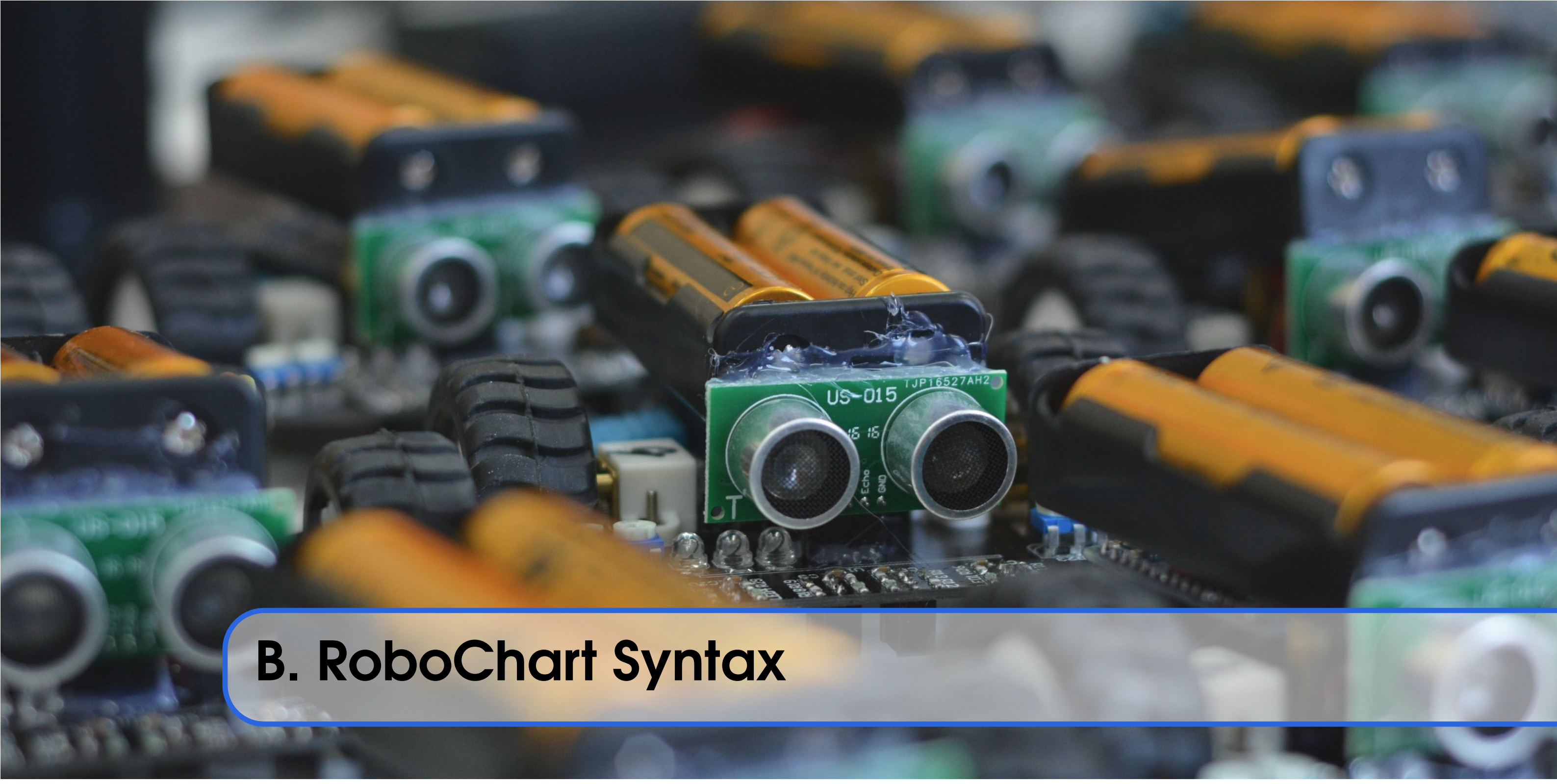
A.7 Adapting the generated semantics

RoboTool automatically generate the semantics of each construct of a model, provided there are no warnings concerning that construct. Additionally, it generates a `instantiations.csp` file and `_assertions.csp` files for each construct and rct file.

These files use a special annotation to indicate which parts of the file can be re-generated. They have annotations of the form -- `generate ID`, where `ID` is any identifier, followed by excerpts of CSP. The specification under the comment will be regenerated every time the model is saved. If you do not want the specification under the comment to be re-generated, add a `not` after the identifier (-- `generate ID not`). This will cause the generator to ignore that part of the specification.

The instantiations file provides default values for constants, functions and bounds, and the assertions file provide default assertion checks where relevant. Use these files and the annotations above to add your own assertions and values without losing them on re-generation.

If you delete these files, they will be regenerated but the changes will be lost.



B. RoboChart Syntax

B.1 Transition labels

The syntax of transition labels is described below.

Transition Label

```
Label ::= Trigger ('<{'Expression'}')? ('['Expression']')? ('/'Statement)?
```

The first expression after the trigger is the end deadline, the second expression is the transition condition and the statement is the transition action. The syntax of triggers is as follows.

Trigger

```
Trigger ::= (Input|Output|Sync|Simple)? ClockReset*
Input ::= Event '?' Variable
Output ::= Event '!' Expression
Sync ::= Event '.' Expression
Simple ::= Event
```

The last component of a trigger consists of zero or more `ClockResets` (e.g., `#T`, where `T` is a clock), which are executed when the transition is taken.

Trigger Type	Meaning
Input Trigger (I)	Receives any value from the event and stores it on the variable.
Output Trigger (O)	Sends the value of the expression through the event.
Sync Trigger (Sync)	Synchronises on the event with the value of the expression.
Simple Trigger (S)	Synchronises on the event.

The table above summarises the meaning of the different types of triggers.

B.2 Types

RoboChart has type system similar to that of Z. The syntax of types is described below.

Type

```
Type  ::=  ’(’ Type ’)’                        – parenthesised type
        |  N                                   – type reference
        |  ’Set’ ’(’ Type ’)’                  – set type
        |  ’Seq’ ’(’ Type ’)’                  – sequence type
        |  Type ’*’ Type                       – product type
        |  Type ’->’ Type                      – function type
        |  Type ’<->’ Type                     – relation type
```

The table below summarises the interpretation of the type constructors.

Element	Concrete Syntax	Meaning
Type Reference	`N`	`N` is the name of a declared type.
Set Type	`Set(T)`	Type of sets of elements of `T`.
Sequence Type	`Seq(T)`	Type of sequences of elements of `T1` to `T2`.
Product Type	`T1 * T2`	Type of pairs whose first element has type `T1` and second element has type `T2`.
Function Type	`T1 -> T2`	Type of functions from `T1` to `T2`.
Relation Type	`T1 <-> T2`	Type of relations between `T1` and `T2`.

B.3 Expressions

The syntax of expressions is summarised below.

Expressions

```
Expr  ::=  (’0’..’9’)+                                  – integer
     |     (’0’..’9’)+.(’0’..’9’)+                      – float
     |     ’"’[^’"’]+’"’                                – string
     |     ’true’ | ’false’                             – boolean
     |     N                                            – reference
     |     ’<’(Expr (’,’ Expr)*)?’>’                    – sequence
     |     ’{’(Expr (’,’ Expr)*)?’}’                    – set
     |     ’{’ N ’:’ Type ’|’ Expr ’@’ Expr ’}’         – set comprehension
     |     ’[’Expr ’,’ Expr ’]’                         – closed interval
     |     ’(’ Expr ’,’ Expr ’)’                        – open interval
     |     N ’::’ N                                     – enumeration constant
     |     ’(|’ (Expr (’,’ Expr)*)? ’|)’                – tuple
     |     Expr ’[’  Expr ’]’                           – array access
     |     Expr ’(’ (Expr (’,’ Expr)*)? ’)’             – function application
     |     Expr ’.’ N                                   – field access
     |     ...                                          – continues on next page
```

Expressions (cont.)

```
Expr  ::=  ...
       |    '-' Expr                                          – negation
       |    Expr '+' Expr                                     – sum
       |    Expr '-' Expr                                     – subtraction
       |    Expr '*' Expr                                     – multiplication
       |    Expr '/' Expr                                     – division
       |    Expr '%' Expr                                     – remainder
       |    Expr '==' Expr                                    – equality
       |    Expr '=' Expr!                                    – difference
       |    Expr '>' Expr                                     – greater
       |    Expr '>=' Expr                                    – greater or equal
       |    Expr '<' Expr                                     – less
       |    Expr '<=' Expr                                    – less or equal
       |    'not' Expr                                        – not
       |    Expr '/\' Expr                                    – and
       |    Expr '\/' Expr                                    – or
       |    Expr '=> Expr                                     – implies
       |    Expr 'iff' Expr                                   – if and only if
       |    'forall' N ':' Type '|' Expr '@' Expr             – universal quantification
       |    'exists' N ':' Type '|' Expr '@' Expr             – existential quantification
       |    'exists1' N ':' Type '|' Expr '@' Expr            – uniqueness quantification
       |    Expr '^' Expr                                     – concatenation
       |    'if' Expr 'then' Expr 'else' Expr 'end'           – conditional
       |    'let' N '==' Expr '@' Expr                        – local definition
       |    'the' N ':' Type '|' Expr '@' Expr                – definite description
       |    'lambda' N ':' Type '|' Expr '@' Expr             – lambda expression
       |    'since' '(' N ')'                                 – clock expression
       |    'sinceEntry' '(' N ')'                            – state clock expression
```

The tables below summarise and explain expressions used to construct values, arithmetic expressions, comparison expressions, logic expressions, and advanced expressions.

Basic Expressions	**Concrete Syntax**	**Comment**
Integer	`(0..9)+`	
Float	`(0..9)+.(0..9)+`	
String	`"..."`	Quoted values
Boolean	`true | false`	
Reference	`N`	`N` is the name of a variable or constant.
Sequence	`<e1,e2,...>`	Sequence with values `ei`.
Set	`{e1,e2,...}`	Set with values `ei`.
Set Comprehension	`{x:T | P @ e}`	Set containing values `e`, calculated from elements of type `T`, for which the predicate `P` holds.
Interval	`[e1,e2]` or `(e3,e4)`	Closed interval between `e1` and `e2`, and open interval between `e3` and `e4`, or a combination of both.
Enumeration	`E::c`	Constant `c` of enuneration `E`.
Tuple	`(|e1,e2,...|)`	Tuple containing elements `ei`.
Array	`e[i]`	The i-th element of array `e`.
Function application	`f(e1,e2,...)`	Apply function `f` to parameters `ei`.
Selection	`e.n`	The `n` field of record `e`.

Arithmetic Expressions	**Concrete Syntax**	**Comment**
Negation	`-e`	Arithmetical negation of expression `e`.
Sum	`e1 + e2`	Sum of `e1` and `e2`.
Subtraction	`e1 - e2`	Subtraction of `e1` and `e2`.
Multiplication	`e1 * e2`	Multiplication of `e1` by `e2`.
Division	`e1 / e2`	Division of `e1` by `e2`.
Modulo	`e1 % e2`	Remainder of dividing `e1` by `e2`.

Comparison Expressions	**Concrete Syntax**	**Comment**
Equality	`e1 == e2`	True if both expressions are equal.
Different	`e1 != e2`	True if both expressions are different.
Greater than	`e1 > e2`	True if `e1` is greater than `e2`.
Greater than or equal to	`e1 >= e2`	True if `e1` is greater than or equal to `e2`.
Less than	`e1 < e2`	True if `e1` is less than `e2`.
Less than or equal to	`e1 <= e2`	True if `e1` is less than or equal to `e2`.

Logical Expressions	**Concrete Syntax**	**Comment**
Logical not	`not e`	True if and only if `e` is false.
Logical and	`e1 /\ e2`	True if and only if `e1` and `e2` are true.
Logical or	`e1 \/ e2`	True if and only if at least one of the expressions is true.
Logical implies	`e1 => e2`	Equivalent to `not e1 \/ e2`.
Logical iff	`e1 iff e2`	Equivalent to `e1=>e2 /\ e2=>e1`
Universal quantification	`forall x: T | P @ Q`	True if and only if for all elements of `T`, if `P` is true, then `Q` is true.
Existential quantification	`exists x: T | P @ Q`	True if and only if there is an element of `T`, for which `P` is true and `Q` is true.
Uniqueness quantification	`exists1 x: T | P @ Q`	True if and only if there is a unique element of `T`, for which `P` and `Q` are true.

Advanced Expressions	**Concrete Syntax**	**Comment**
Concatenation	`e1^e2`	Concatenate sequences `e1` and `e2`.
Conditional	`if c then e else f end`	If condition `c` is true, `e1` else `e2`.
Local definition	`let n == e @ f`	Define locally `n` and use it to calculate `f`.
Definite description	`the x: T | P @ e`	The value `e` calculated based on the unique `x` for which `P` holds.
Lambda expression	`lambda x: T | P @ e`	The anonymous function that takes values of type `T` for which `P` holds, to values `e` calculated based on `x`.
Clock Expression	`since(C)`	Expression counting elapsed time since the last reset of clock `C`.
State Clock Expression	`sinceEntry(S)`	Expression counting elapsed time since entry of state `S`.

B.4 Actions and statements

The syntax of actions is described below.

Actions

```
Action ::= ('entry' | 'during' | 'exit') Statement
```

The syntax of statements is as follows.

Statements

```
Statement  ::=  'skip'
             |  N '(' (Expr (',' Expr)*)? ')'                 – operation call
             |  'if' Expr                                     – conditional
                'then' Statement
                'else' Statement 'end'
             |  N '=' Expr                                    – assignment
             |  N '!' Expr                                    – output event
             |  N '?' N                                       – input event
             |  N                                             – simple synchronisation
             |  N '.' Expr                                    – synchronisation
             |  Statement ';' Statement                       – sequential composition
             |  Statement '<{ Expr '}'                        – timed statement
             |  'wait' '(' Expr ')'                           – wait statement
             |  'wait' '(' '[' Expr ',' Expr ']' ')'          – nondeterministic wait
             |  '#' N                                         – clock reset
```

Statement	Concrete Syntax	Comment
Skip	`skip`	Statement that terminates immediately.
Call	`o(e1,e2,...)`	Calls operation `o` with parameters `ei`.
Conditional	`if c then S1 else S2 end`	If `c` is true, execute `S1`, otherwise execute `S2`.
Assignment	`x = e`	Assign expression `e` to variable `x`.
Output event	`ev!e`	Output value `e` through channel `ev`.
Input event	`ev?x`	Receive value through channel `ev` and store it in variable `x`.
Synchronisation	`ev.e`	Synchronise on value `e` through event `ev`.
Synchronisation	`ev`	Synchronise on event `ev`.
Sequential composition	`S1;S2`	Execute `S1`, and then `S2`.
Timed Statement	`S<{e}`	Statement `S` is required to terminate within `e` time units.
Wait Statement	`wait(e)`	Waits for `e` units of time.
Nondeterministic Wait	`wait([a,b])`	Waits nondeterminstically for d units of time where `a`$\leq d \leq$`b`.
Clock Reset	`#C`	Resets clock `C`.

C. Assertion DSL Syntax

Assertions

```
Assertion  ::=  ('timed' | 'untimed')? 'assertion' N ':' SPEC
                ('in' 'the' MODEL)?
                ('with' ('constant'| 'constants') CONSTANTS)?
```

Specification

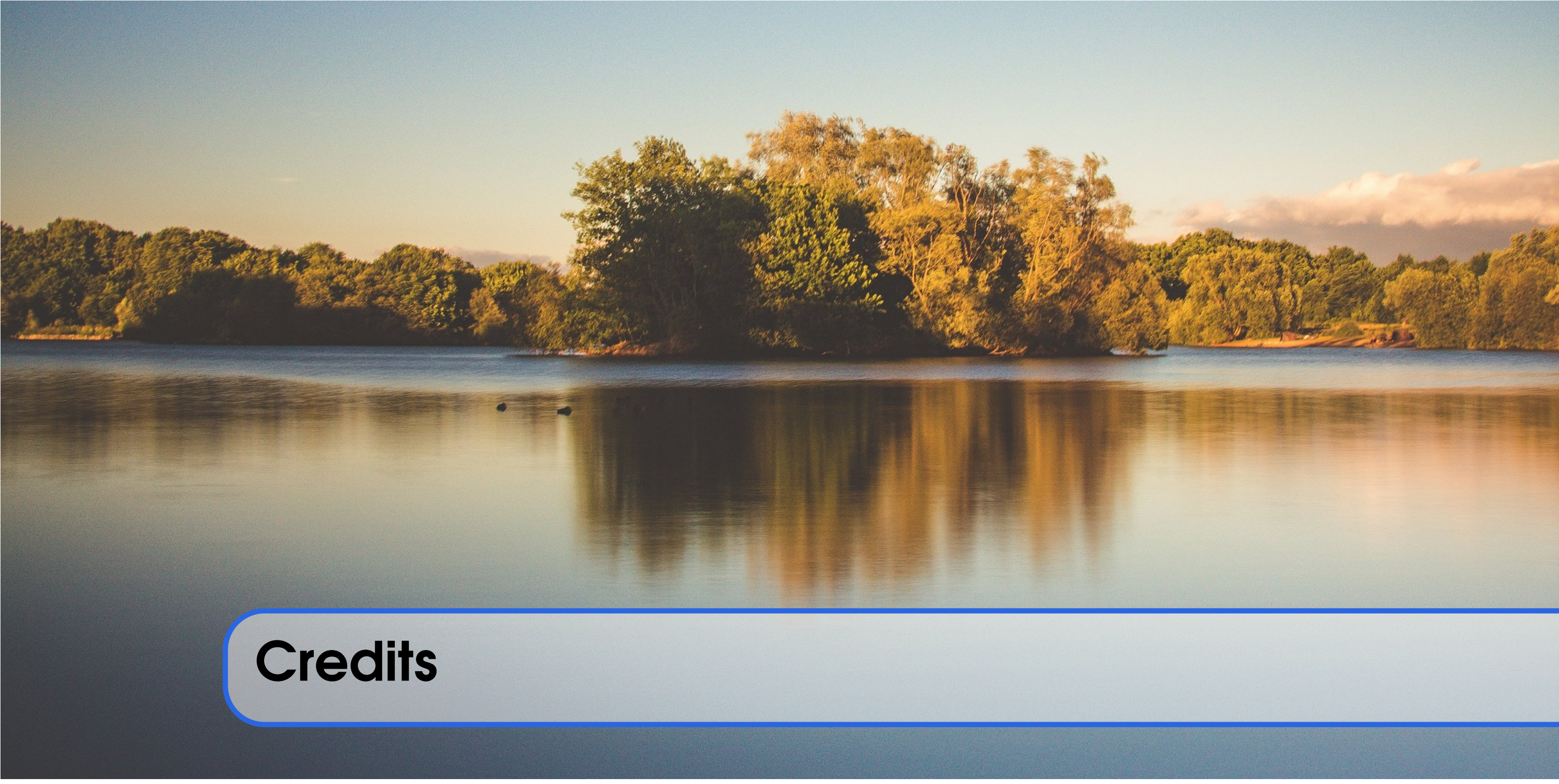
```
SPEC   ::=  N 'is' ('not')? PRED
         |  N ('does' 'not' 'terminate' | 'terminates')
         |  N 'is' ('not')? 'reachable' 'in' N
         |  N REL N
         |  'clock' N 'is' ('not')? 'initialised'
PRED   ::=  'deadlock-free'
         |  'divergence-free'
         |  'deterministic'
         |  'timelock-free'
REL    ::=  'refines'
         |  'equals'
         |  'does' 'not' 'refine'
         |  'is' 'not' 'equal'
```

CSP Models

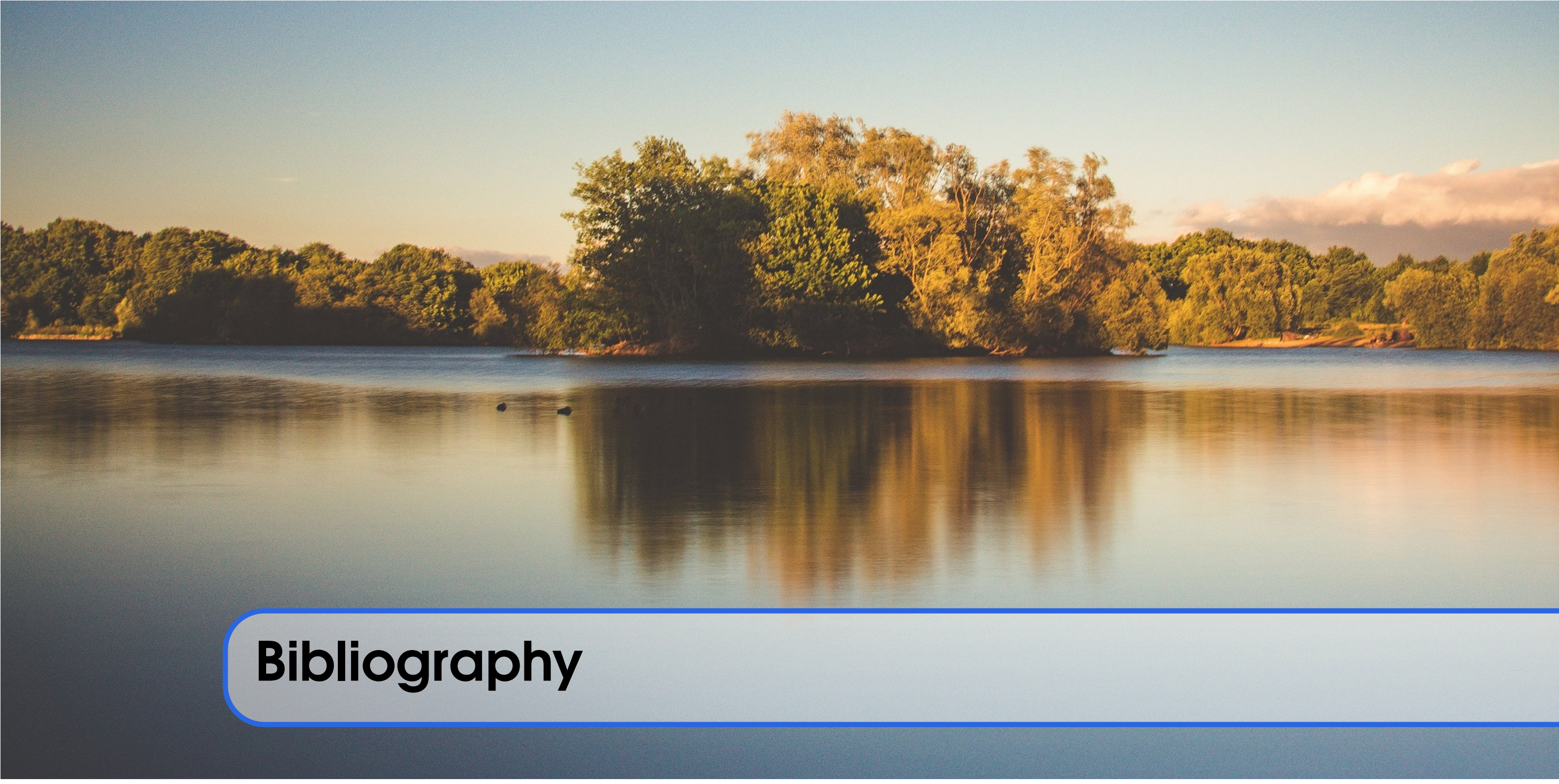
```
MODEL   ::=  'traces' 'model'
          |  'failures' 'model'
          |  'failures' 'divergence' 'model'
```

Constant Definitions

```
CONSTANTS  ::=  (DEF (’,’ DEFs)*)?
DEF        ::=  N (’assigned’ | ’set’ ’to’ | ’with’ ’value’) Expr
```



Credits



Bibliography

[1] A. W. Roscoe. *Understanding Concurrent Systems*. Texts in Computer Science. Springer, 2011.

[2] C. Dixon et al. “Towards temporal verification of swarm robotic systems”. In: *Robot. Auton. Syst.* 60.11 (2012), pages 1429–1441 (cited on page 33).

[3] Ana Cavalcanti et al. “Verified simulation for robotics”. English. In: *Science of Computer Programming* (Jan. 2019). ISSN: 0167-6423. DOI: `10.1016/j.scico.2019.01.004`.

[4] J. Chen, M. Gauci, and R. Groß. “A strategy for transporting tall objects with a swarm of miniature mobile robots”. In: *2013 IEEE International Conference on Robotics and Automation*. May 2013, pages 863–869. DOI: `10.1109/ICRA.2013.6630674` (cited on page 34).

[5] Alvaro Miyazawa et al. “RoboChart: modelling and verification of the functional behaviour of robotic applications”. In: *Software & Systems Modeling* (Jan. 2019). ISSN: 1619-1374. DOI: `10.1007/s10270-018-00710-z`.

[6] S. Schneider. *Concurrent and Real-time Systems: The CSP approach*. Wiley, 2000.

[7] T. Gibson-Robinson et al. “FDR3: A Modern Refinement Checker for CSP”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. 2014, pages 187–201 (cited on page 35).