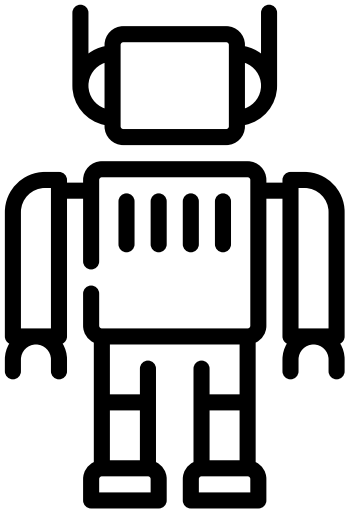

RoboTool
RoboChart Tool Manual

Alvaro Miyazawa

May 31, 2019

This is the manual for the *RoboChart* tool. It describes the requirements, installation process and usage of the tool.

Contents

Chapter 1

Requirements

- Linux – Certain aspects of eclipse seems to work differently in windows, linux and iOS. While our plugins will work on any platform that has an eclipse distribution, we will focus our efforts on fixing bugs that occur on linux.
- Eclipse Neon – `https://www.eclipse.org/`.
- FDR3 (optional) – `https://www.cs.ox.ac.uk/projects/fdr/` Needed if you want to analyse the automatically generated semantics of the RoboChart specifications.

Chapter 2

Installation

1. Download Eclipse Neon (`https://www.eclipse.org/`) and install it. You can choose the *Eclipse IDE for Java Developers* distribution;
2. Select `Help > Install New Software...`;
3. Type `http://www.barom.org.uk/robochart/update` in the `Work with` textbox and press `Enter`;
4. Click and select the box next to *RoboChart Editors*;
5. Click `Next`;
6. Click `Next`;
7. Accept the license and click `Finish`;
8. A security warning will appear; click the `OK` button if you wish to continue with installation;
9. Once the plugins are installed, a popup box will appear requesting Eclipse to be restarted. Click `Yes`.
10. Verify you installation by:
    a) Click the `Help > About Eclipse` menu item;
    b) Click the `Installation Details`;
    c) Look for *RoboChart Feature* in the list of Installed Software;
    d) Click the arrow to the left of *RoboChart Feature*;

e) The *RoboChart Feature* should contain four sub-items: Sirius Core Runtime, Sirius Runtime IDE, Sirius Runtime Support AQL, and Sirius Specifier Environment.

11. You are done. If you have any problems with the installation or usage, check for an associated issue or create a new issue on `https://bitbucket.org/robocalc-tools/robochart/issues`. (This tool project is currently private to the RoboCalc group, so the issue tracker is only accessible to users with access to the project repository.)

2.1 Updating the tool

In order to update the tool, there are a few alternative ways:

1. Click `Help > Check for Updates`
2. If there are updates available, *RoboChart Feature* will be available; select it and click `Next` twice
3. Accept the license agreement, and click `Finish`
4. A security warning will appear; click the `OK` button if you wish to continue with installation;
5. Once the plugins are installed, a popup box will appear requesting Eclipse to be restarted. Click `Yes`.

While convenient, the *Check for Updates* features looks for all updates, not just RoboChart updates, and it can take longer. Additionally, it may lead to accidental updates. If you want to update only the RoboChart tool, follow these steps:

1. Click `Help > About Eclipse`
2. Click the button `Installation Details`
3. In the `Installed Software` tab, find the *RoboChart Feature* and select it;
4. Click the button `Update...`; this will search for updates specific to RoboChart
5. If an update is found, a details page will be shown, click `Next`;

6. Accept the license agreement, and click `Finish`

7. A security warning will appear; click the `OK` button if you wish to continue with installation;

8. Once the plugins are installed, a popup box will appear requesting Eclipse to be restarted. Click `Yes`.

CHAPTER 3

Usage

The RoboChart plugins are based on the Sirius framework. A generic user's manual for Sirius based tools is available in `http://www.eclipse.org/sirius/doc/user/Sirius%20User%20Manual.html`. The next sections describe basic activities specific to our tool.

A RoboChart specification is a collection of named or anonymous packages (RCPackage in the abstract syntax). In the tool, we associate a RoboChart specification with a Modeling project, and a package with a file with the extension `rct`. The `rct` records information about the elements defined in the package, but not how they are represented graphically. Packages recorded as `rct` files can be visualised and edited through a graphical editor. These diagrams are recorded in the `aird` file. For this reason, we first create a `rct` file, and then create a diagram to represent that file.

The next section describes how to create a project (i.e., RoboChart specification), and the following section describes how to create new `rct` files (i.e., RoboChart packages).

3.1 Creating a new project

1. Select Modeling pespective, if it is not yet selected:
   a) Click the `Window > Pespective > Open Pespective > Other ...` menu item;
   b) Select the item `Modeling` from the list; and
   c) Click `OK`. The *Modeling* perspective is divided in four main parts:
      i. *Model Explorer* – top left corner
      ii. *Outline* – bottom left corner
      iii. *Text editor* – top right corner

iv. *Properties, Problems and Console* – bottom right corner

2. Create a new Modeling project

   a) Right click the *Model Explorer* window and select `New > Project`; alternatively, in the menu click `File > New > Project`;

   b) Either type *Modeling* in the wizard text box, or open the *Sirius* folder;

   c) Select the item *Modeling Project* and click `Next`;

   d) Type the project name and click `Finish`

3. In order to browse the contents of the project click the arrow to the left of the project name in the Model Explorer; the new project will contain two initial components:

   a) Project Dependencies – This can be used to add references to external packages, but I have not used it;

   b) representations.aird – This file stores the graphical representation of all the diagrams.

3.2 Create a new RoboChart package

A project can have several packages (`rct` files). Here, we describe how to create a package inside a project.

1. Select the project in which you wish to create a new package;

2. Press `Ctrl+N` and select `General > File`; or right click the project and select `New > File`; or in the menu click `File > New > File`;

3. Type the name of the file with extension *.rct* and press `Finish`;

4. The first time you create a package in a new project, a popup will appear asking to convert the project into an Xtext project. This is necessary, so click `Yes`;

5. The first time you create a package in a new project, right click the project and select *Viewpoint Selection*. A popup window will appear with an item *RoboChart*; tick this item and click `OK`;

6. In order to open a diagram follow the following steps:

   a) Click the arrow to the left of the name of the *rct* file; there should be a subitem *RCPackage* under the file name[1];

      i. If there is no arrow to the left of *RCPackage*, or no subitem of type diagram (marked by the icon ), right click *RCPackage* and click `New Representation > {name}`, where {name} is the name of the file without the extension[2]. Finally, click `OK` (this creates a new diagram associated with the *rct* file);

      ii. If there is already a diagram under *RCPackage*, do nothing.

   b) Finally, double click a diagram (identified by the icon ) to open it.

An alternative, more automated way of creating a RoboChart involves the use of the RoboChart Wizard. Notice that this is a newer feature and more prone to errors, so if it fails, try the steps above.

1. Select the project in which you wish to create a new package;
2. Press `Ctrl+N`, or right click the project and select `New > Other...`; or in the menu click `File > New > Other...`;
3. Type *RoboChart* in the wizard text box, or find the RoboChart folder and open it;
4. Select the item *New RoboChart Package* and click `Next`;
5. Change the file name (it must end in .rct) and click `Finish`;
6. The first time you create a package in a new project, a popup will appear asking to convert the project into an Xtext project. This is necessary, so click `Yes`;
7. The wizard will create the `rct` file, select the appropriate viewpoint, create the diagram and open it automatically.

[1] Sirius does not seem to behave consistently when creating a new file, sometimes it creates the diagram automatically, other times it does not.
[2] If you wish, you can change the name.

3.3 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[3].

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`.

3.4 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.

   Notice that 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[4];

[3] We must delete the diagram associated with the RoboChart package, not the whole `aird` file.
[4] While you can use Ctrl+Z to undo changes to the diagram, it can create inconsistencies, so use it carefully.

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.

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[5]

      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`.

3.5 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.

[5] See the language reference manual for the concrete syntax of actions and other elements.

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[6].

   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.

3.6 Exporting diagrams as figures

It is possible to export the diagrams as figure[7].

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)

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

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.

3.7 Exporting projects

Since the tool is under development, it is advisable to not only frequently save your specifications, but also export them as compressed files. To do that follow these steps:

1. Right-click the project in the Model Explorer and click `Export`

2. Select `General > Archive File` and click `Next`

3. Give a name to the compressed file (or select an existing file using the `Browse` button) and click `Finish`

3.8 Importing projects

To import a RoboChart specification[8] into eclipse:

1. Right-click the Model Explorer and click `Import` (alternatively, click `File > Import`)

2. Select `General > Existing Projects into Workspace` and press `Next`

3. Choose the option `Select archive file` and press the button `Browse`

[8] This should have been exported as described in the previous section

4. Find the compressed file (zip or tar file) that contains the specification and press `OK`

5. The project should be selected in the window below. In this case, click `Finish`. If the project is not selected in the window, it is because a project with the same name already exists in the workspace; either delete the project or rename it before trying to import again.

3.9 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.

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

3.10 Opening FDR directly from Eclipse

The semantics of the RoboChart model is generated automatically in the `src-gen` folder as a collection of `.csp` files. Each file at the root folder `src-gen` is self-contained and can be open in FDR to perform analysis and simulation. The actual definitions of the elements of the model can be found in the files in the folder `defs`.

It is possible to open the `.csp` files directly from the `Model Explorer`.

If you installed FDR on Ubuntu, it is possible that FDR is the default program for `.csp` files. In this case, it may be possible to open the file in FDR by double clicking the file. If, instead of FDR, the file is open in Eclipse, right-click the `.csp` file, select `Open with > System Editor`.

If FDR is not the default program for `.csp` files, you can add it to the list of editors in Eclipse as follows:

1. Right-click a `.csp` file and select `Open with > Other`
2. Click `Browse...`, find the `fdr4` executable and click `OK`
3. Select `External programs`, and check both boxes below the `Browse...` button
4. Click `OK`

Once you follow these steps, every time you double click a `.csp` file in Eclipse, FDR should open. If you want to open the `.csp` file in a text editor, right-click the file, and select `Open with > Text Editor`. To open the file in FDR again, right-click the file, and select `Open with > fdr4`.

Chapter 4

Model Checking with FDR

RoboTool provides facilities for running the CSP model checker – FDR – on user specified assertions directly via the tool. Section 4.1 describes the language used to specify assertions, and Section 4.2 describes the steps to run FDR from inside RoboTool.

4.1 Assertion language

RoboTool provides a textual editor to support the specification of assertions in files with extension `.assertions`.

4.1.1 assertion

An assertion is specified as follows:

assertion = (**untimed** | **timed**)? **assertion** *NAME* : *specification* (**in the** *CSP Model*)?
(**with constant**(**s**)? *constant definitions*)?

specification =
ELEMENT **is** (**not**)? (**deadlock-free** | **divergence-free** | **deterministic** | **timelock-free**) |
ELEMENT (**does not terminate** | **terminates**) |
STATE **is** (**not**)? **reachable in** *ELEMENT* |
ELEMENT (**refines** | **equals** | **does not refine** | **is not equal**) *ELEMENT* |
clock *CLOCK* **is** (**not**)? **initialised**

CSP Model = **failures divergence model** | **traces model** | **failures model**

constant definitions = *definition* (**,** *definition*)* (**, and** *definition*)?

definition = *NAME* (**assigned** | **set to** | **with value**) *Expression*

In the description above, *NAME* is any valid identifier, *ELEMENT* is the name of any state machine, controller, module or collection, *STATE* is any state in a state machine, and *CLOCK* is any clock name. Below is an example of an assertion:

assertion A1: LMachine **is deadlock-free with constants**
SET_SPACE_TIME **set to** 1,
SET_SPACE_BOUND **set to**1,
CHANGE_LANE_TIME **set to** 1,
CHANGE_LANE_BOUND **set to** 1,
JOIN_DISTANCE_TIME **set to** 1,
JOIN_DISTANCE_BOUND **set to** 1,
and maxPL **with value** 3 .

While it is not necessary to specify all required or defined constants in an assertion, the tool will give a warning indicating which constants are missing.

The value of unspecified constants is either a default value for the type of the constant (for example, `0` is the default value of the type `nat`), the value stored in the `instantiations.csp` file, or the initial value specified in the model.

Values specified in a assertion override the usual definition.

While the assertion language supports comparison of constructs (refinement and equality), this may not be enough for the verification of more general properties. In order to address this limitation, we allow the specification of general CSP processes in an `.assertions` file, which can then be used in an assertion as a member of the class *ELEMENT*. The syntax that allows the specification of CSP processes in an `.assertions` file is as follows:

CSP specification = (**untimed** | **timed**) **csp** *PROCESS_NAME* **csp-begin** *CSP-M* **csp-end**

A *CSP specification* declares an *ELEMENT* with name *n* (*PROCESS_NAME*). The content *CSP-M* is a CSP specification in the syntax accepted by FDR. This specification must include one process called *n*.

4.2 Running FDR

In order to run an `.assertions` file in FDR via RoboTool, the user must select the file in the `model explorer` or `project explorer` window, and either:

1. Click the `FDR` button at the tool bar;
2. Right-click the file, and click `RoboTool Analysis > FDR`.

In both cases, FDR will verify the assertions in the background, providing some progress information in the `Progress` window, and upon completion a HTML report is generated and displayed in the tool.

Appendix A

Deviations from RoboChart reference manual

RoboTool is developed and maintained to be consistent with the RoboChart language definition. Nevertheless, certain aspects of the language definition have not yet been implemented, or have been implemented in a slightly different way, due to limitations in technologies. Here we discuss such deviations.

A.1 Well-formedness condition

Two well-formedness conditions have not been implemented in RoboTool as it requires the use of automated or semi-automated theorem provers. They are:

J2 The guards of the transitions out of a junction must form a cover; and

V1 If the initial value of a required variable or constant of a state machine or controller is defined, it must be consistent with the value of any (complementing) variable provided by the contexts (controllers or modules) where the state machine or controller is used.

A.2 Semantics

While the code generation faithfully implements the semantics of RoboChart, it targets the flavour of CSP called CSP-M, which is the input language for FDR. In particular, we make extensive used, in the implementation, of process declarations and FDR compression functions to improve the usability of our semantic models.

Other aspects, such as naming conventions, channel declarations, and so on, which are underspecified in the semantics, are made concrete in the implementation to support the use of FDR with our models. Finally, the concrete syntax of CSP-M varies slightly from the syntax standard CSP, which is used in the reference manual.