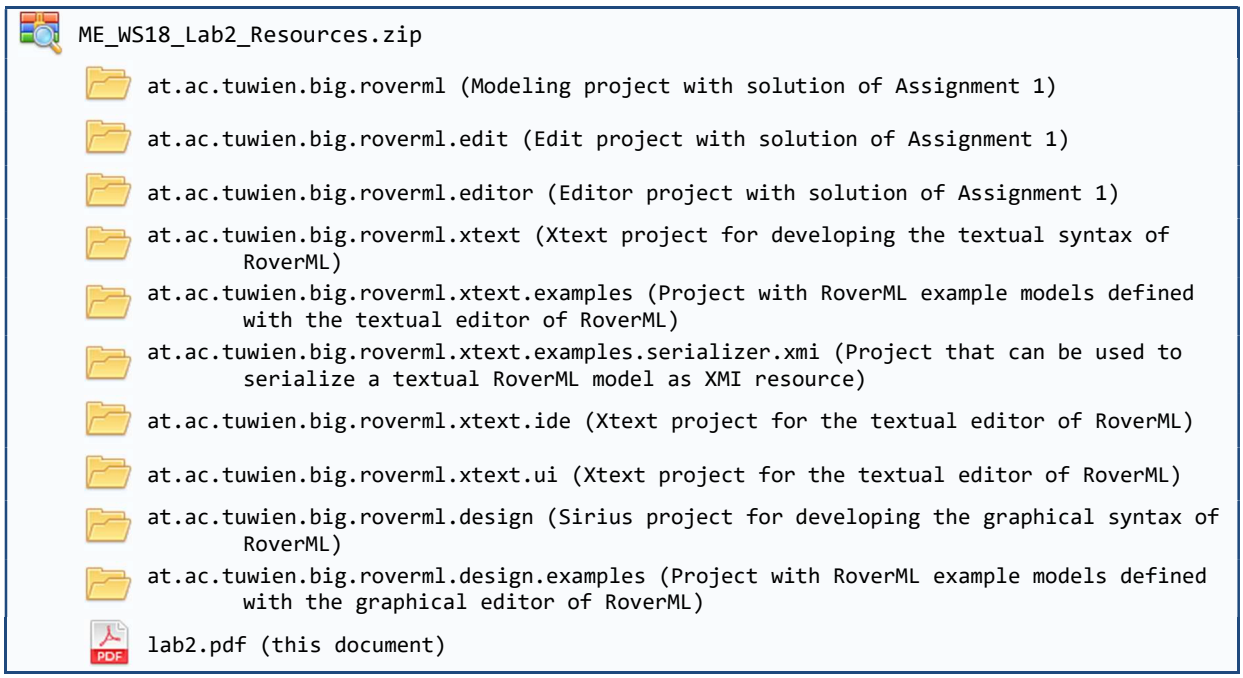
Model Engineering Lab 188.923 IT/ME VU, WS 2018/19	**Assignment 2**
Deadline: Upload (ZIP) in TUWEL until Sunday, November 18th, 2018, 23:55 Assignment Review: Wednesday, November 21st, 2018	25 Points

Concrete Syntax

The goal of this assignment is to develop the concrete syntax of the *Rover Modeling Language (RoverML)* using Xtext and Sirius. In particular, you will develop a textual concrete syntax and textual editor with Xtext in Part A of this assignment, and a graphical concrete syntax and graphical editor with Sirius in Part B of this assignment.

Assignment Resources



Before starting this assignment, make sure that you have all necessary components installed in your Eclipse. A detailed installation guide can be found in the TUWEL course[1].

It is recommended that you read the complete assignment specification at least once. If there are any parts of the assignment specification or the provided resources that are ambiguous to you, don't hesitate to ask in the forum for clarification.

[1] Eclipse Setup Guide: https://tuwel.tuwien.ac.at/mod/page/view.php?id=388945

Part A: Textual Concrete Syntax

In the first part of this assignment, you have to develop the textual concrete syntax (grammar) for RoverML with Xtext. Additionally, you have to implement scoping support for the textual editor that is generated from the developed grammar.

The grammar has to be built upon the sample solution of the RoverML metamodel provided in the assignment resources (`at.ac.tuwien.big.roverml/model/roverml.ecore`).

Do <u>not</u> use the metamodel you have developed in Assignment 1, but the sample solution provided in the assignment resources!

A.1 Xtext Grammar for RoverML

Develop an Xtext grammar that covers all modeling concepts provided by RoverML.

The Xtext grammar has to follow the example given in Figure 1. This textual RoverML example models is also available in full length in the examples project: `at.ac.tuwien.big.roverml.examples.xtext/Example1-ozobotEvo.rml`. A corresponding model serialized with XMI is also provided in the file `Example1-ozobotEvo.roverml` located in the same project.

Figure 1: Example RoverML model in textual concrete syntax

```
roverSystem {
   rovers {
      rover ozobotEvo {
         components {
            motor leftMotor,
            motor rightMotor,
            distanceSensor sensor(kind “proximitySensor”),
            light topLight(kind “LED”),
            light frontLight1(kind “LED”),
            light frontLight2(kind “LED”),
            light frontLight3(kind “LED”),
            light frontLight4(kind “LED”),
            light frontLight5(kind “LED”)
         }
      }
   }
   roverPrograms {
      roverProgram squareWalk {
         rover ozobotEvo
         block repeat 4 times {
            commands {
               setTopLightColorGreen: set lights (topLight) to color green,
               moveForward: move 4.0 cm with 1.0 cm_per_s,
               setTopLightColorRed: set lights (topLight) to color red,
               rotateRight: rotate 90.0 degree
            }
            transitions {
               setTopLightColorGreen > moveForward,
               moveForward > setTopLightColorRed,
               setTopLightColorRed > rotateRight
            }
         }
      }
   }
}
```

To implement the Xtext grammar for RoverML, perform the following steps:

1. Setting up your workspace

As mentioned in the beginning, the Xtext grammar has to be based on the sample solution of the RoverML metamodel. We provide this sample solution as well as skeleton projects for the Xtext grammar as importable Eclipse projects.

Import projects

To import the provided project in Eclipse, select *File → Import → General/Existing Projects into Workspace → Select archive file → Browse*. Choose the downloaded archive *ME_WS18_Lab2_Resources.zip* and import the following projects:

1. `at.ac.tuwien.big.roverml`
2. `at.ac.tuwien.big.roverml.edit`
3. `at.ac.tuwien.big.roverml.editor`
4. `at.ac.tuwien.big.roverml.xtext`
5. `at.ac.tuwien.big.roverml.xtext.ide`
6. `at.ac.tuwien.big.roverml.xtext.ui`

Register the metamodel

Should the Xtext grammar file `Roverml.xtext` (project `at.ac.tuwien.big.roverml.xtext`) show errors, then register the RoverML metamodel in your Eclipse instance, such that the Xtext editor with which you develop your grammar knows which metamodel you want to use. You can register your metamodel by clicking right on `at.ac.tuwien.big.roverml/model/roverml.ecore` → *EPackages registration → Register EPackages into repository*. Clean your projects afterwards by selecting *Project → Clean… Clean all projects*.

After performing these steps, you should have six projects in your workspace without any errors (there may be warnings which you can ignore).

2. Developing your Xtext grammar

Define the Xtext grammar in the file `Roverml.xtext` located in the project `at.ac.tuwien.big.roverml.xtext` in the package `src/at.ac.tuwien.big`. Documentation about how to use Xtext can be found in the lecture slides and in the Xtext documentation[2].

Hint: Make use of the `ID` rule defined in the base grammar `org.eclipse.xtext.common.Terminals` to specify the names of different model elements, e.g., the *name* of components.

Hint: When defining cross-references to other model elements, make sure to use the rule `QualifiedName` already defined in the Xtext grammar `Roverml.xtext`, e.g., `referenceToClass=[Class|QualifiedName]`.

3. Testing your editors

Generate Xtext artifacts

After you have finished developing your Xtext grammar in `Roverml.xtext`, you can automatically generate the textual editor. The Xtext grammar project `at.ac.tuwien.big.roverml.xtext` contains a so-called model workflow file (`GenerateRoverml.mwe2`), which orchestrates the generation of the textual editor. By

[2] Xtext documentation: http://www.eclipse.org/Xtext/documentation/

default, the workflow will generate all the necessary classes and stub classes in the grammar project and in the related *.ide*-project and *.ui*-project.

To run the generation, right-click on the workflow file `GenerateRoverml.mwe2` and select *Run As → MWE2 Workflow*. This will start the editor code generation. Check the output in the Console to see if the generation was successful. Please note that after each change in your grammar, you must re-run the workflow to update the generated editor code.

If you start the editor code generator for the first time, the following message may appear in the console:

```
*ATTENTION*
It is recommended to use the ANTLR 3 parser generator (BSD licence -
http://www.antlr.org/license.html). Do you agree to download it (size 1MB) from
'http://download.itemis.com/antlr-generator-3.2.0.jar'? (type 'y' or 'n' and hit enter)
```

Type *y* and download the jar file. It will be automatically integrated into your project.

Start a new Eclipse instance with your plugins

For starting the generated editor, we have already provided a launch configuration located in the project `at.ac.tuwien.big.roverml.xtext`, which is called "`ME Lab 2 Runtime Eclipse.launch`". Run it by right-clicking on it and selecting *Run As → ME Lab 2 Runtime Eclipse.* You can have a look at the configuration by right-clicking on that file and selecting *Run As → Run Configurations…*.

Start modeling

In the newly started Eclipse instance, you can create a new empty project (*File → New → Project → General/Project*) and create a new file (*File → New → File*) with the extension *.rml* in this project. If you are asked to add the Xtext nature to the project ('Do you want to convert 'projectname' to an Xtext project?') hit *Yes.* Right-click on the created **.rml* file and select *Open With → Roverml Editor*. Now you can start modeling a RoverML model to test the generated textual RoverML editor.

Hint: By pressing Ctrl+Space you activate content assist/auto completion, which provides you a list of keywords or elements that can be input at the next position.

Hint: When you have cross-references to other elements in your grammar, you can check which element is referenced in a model by using the linking feature. For this just hold Ctrl and click with the mouse on the cross-reference.

Hint: Note that after each change in your grammar, you must re-run the workflow to update the generated editor code. Furthermore, you will also have to restart the Eclipse instance that you use for testing the generated editor. We also advise you to always clean your example project (*Project → Clean… → Clean all projects*) and re-open your example model (`*.rml`).

Check examples

In the newly started Eclipse instance, import the example models included in the project `at.ac.tuwien.big.roverml.xtext.examples` provided in the assignment resources. Import the project in the workspace (see "Setting up your workspace") and make sure that your Xtext editor for RoverML can process the examples without problems, i.e. shows no errors or warnings when you open the models.

Check the following examples:

- `Example1-ozobotEvo.rml`
- `Example2-PolarSysRover.rml`

Also make sure that values for attributes and references are actually assigned to the model elements. For checking this, serialize your textual `*.rml` model as XMI resource `*.roverml`. To achieve this, you can use the following project provided in the assignment resource: `at.ac.tuwien.big.roverml.xtext.examples.serializer.xmi`
Import this project into the Eclipse instance that you use for testing your editors and run the serializer by right-clicking on `XMISerializer.java` found in the "src" folder and select *Run As → Java Application*. This will produce the following two RoverML models in the folder `model-gen` (you may have to refresh your workspace for seeing the models; for this, right-klick on the folder `model-gen` and select *Refresh*):

- `Example1-ozobotEvo.roverml` (corresponds to `Example1-ozobotEvo.rml`)
- `Example2-PolarSysRover.roverml` (corresponds to `Example2-PolarSysRover.rml`)

Investigate the elements of these models in the tree editor of RoverML (right-click on the `*.roverml` model file and select *Open With → RoverML Model Editor*) and make sure that their attribute values and references are correctly set via the *Properties* view (if the *Properties* view is not shown, right click on any model element and select *Show Properties View*).

A.2 Scoping Support for RoverML

In the grammar that you have developed, you should have defined some cross-references, e.g., a command refers to a component and a transition refers to a source and a target command. When testing your editor, you will notice that whenever there is a cross-reference, the content assist (CTRL + Space) will provide all elements of the respective type that the editor can find. Which elements the editor provides is defined in the scoping[3] of the respective reference. By default, the editor searches through the whole class-path of the project and you will notice that if you have many models in your project, a lot of elements will show up as possible reference.

To restrict this behavior, Xtext provides a stub where developers can define their own scope. The stub for the RoverML language can be found in the grammar project `at.ac.tuwien.big.roverml.xtext` in the package `src/at.ac.tuwien.big.scoping` (`RovermlScopeProvider.java`). Your task is to implement the following scoping behavior in addition to the already existing ones:

1) Scoping for the referenced lights of the SetLightColor command

It is possible to define multiple rovers (and multiple programs) in a rover environment. A rover program references exactly one rover. If the SetLightColor command is used, the suggested lights should only be of the rover referenced in the program defining the command.

Example: The command "setTopLightColorGreen" of the rover program "squareWalk" defined in the example model `Example1-ozobotEvo.rml` in line 29 may only refer to the lights defined for the rover "ozobotEvo" ("topLight", "frontLight1", etc.) because the rover program refers to this rover. It is not allowed to refer to the light "mainLight" of the rover "ozobotBit".

2) Scoping for the target of a transition

[3] http://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#scoping

The suggested commands (source and target) for transitions should only be the ones defined in the same rover program and the same block as the transitions.

Example: In the model `Example2-PolarSysRover.rml`, a transition in the rover program "testFrontLight" can only reference commands defined in this program. It is not allowed to reference commands from the rover program "travelBetweenObstacles" above.

Hint: Xtext uses a so-called polymorphic dispatcher to handle declarative scoping. This dispatcher uses the Reflection API to search for methods with a specific signature in the `ScopeProvider` class. How you can use this is explained in the lecture slides and in the Java documentation of the class `AbstractDeclarativeScopeProvider`[4], which is the super class of your scoping stub `RovermlScopeProvider.java`.

After you have implemented the scoping as defined above, start again a new instance of your Eclipse (use the provided launch configuration *"ME Lab2 Runtime Eclipse.launch"*) and test your scoping with the content assist (Ctrl+Space) and linking feature (Ctrl+Left Click).

Check examples

In the newly started Eclipse instance you can check the remaining models provided by the examples project `at.ac.tuwien.big.roverml.xtext.examples` and see whether your scoping covers these cases. However, it should be noted that these tests do not cover all possible cases and additional testing of the implemented scoping from your side is required.

scoping-test1.rml (folder *scoping*)	This is a variation of the example "Example1-ozobotEvo.rml", in which the wrong light component is set for the command "setTopLightColorGreen" (see line 29). In particular, the light "mainLight" is wrongly set as the component targeted by the command instead of the correct light "topLight". Your RoverML editor should show an error marker for this line. When you use the content assist in line 29 after the opening brace, only the lights defined as components of the rover "ozobotEvo" should be offered. After choosing any of them, the errors should disappear.
scoping-test2.rml (folder *scoping*)	This is a variation of the example "Example2-PolarSysRover.rml", in which wrong commands are defined in the lines 50 and 51. There are now two rover programs, and each one defines its own commands. Your RoverML editor should show an error marker for these lines (there may be an additional error because of an OCL constraint). In line 50 the command "moveForward" from the rover program "travelBetweenObstacles" is used as target of a transition. On line 51 this command is used again as the source of a transition. If you use the content assist, only commands from the current program should be offered. Choose the "setFrontLightColorYellow" command for both lines. The errors should disappear.

[4] http://download.eclipse.org/modeling/tmf/xtext/javadoc/2.3/org/eclipse/xtext/scoping/impl/AbstractDeclarativeScopeProvider.html

Part B: Graphical Concrete Syntax

In the second part of this assignment, you have to develop a graphical concrete syntax and graphical editor for RoverML with *Sirius*. Like the textual concrete syntax, also the graphical concrete syntax has to be built upon the sample solution of the RoverML metamodel provided in the assignment resources.

Sirius Diagram Editor for RoverML

Develop a graphical concrete syntax and graphical (diagram) editor that allows to display and edit RoverML model. For this, develop *diagram mappings* and *element creation tools* as described in the following.

Mappings
For displaying RoverML models on diagrams, develop diagram node mappings and diagram edge mappings for the following RoverML concepts:

- Rover and RoverProgram
- Block and Repeatblock
- Commands: Wait, Terminate, Repeat, Move, Rotate, SetLightColor
- Components: Light, Motor, Compass, GPS, DistanceSensor
- Transition and Triggered Transitions: DistanceSensorTrigger, CompassTrigger and GPSTrigger
- Quantities: Angle, Time, Length, Velocity

The diagram editor has to be able to display these elements following the *graphical concrete syntax* that is illustrated in Figure 2 and Figure 3. Figure 2 shows the same example model as the one defined textually in Figure 1. The example model is available in the project `at.ac.tuwien.big.roverml.design.examples` in the model file `Example1-ozobotEvo.roverml`. Figure 3 shows the model from file `Example2-curiosity.roverml`.

The images for the mappings to be defined for the concepts are provided in `at.ac.tuwien.big.roverml.design/images`.

Creation Tools
For editing RoverML models, develop element creation tools for the following RoverML concepts (see also the tool palette shown in Figure 2):

- Rover, RoverProgram and Block
- Commands: Wait, Terminate, Repeat, Move, Rotate, SetLightColor
- Components: Light, Motor, Compass, GPS, DistanceSensor
- Transition and Triggered Transitions: DistanceSensorTrigger, CompassTrigger and GPSTrigger
- Quantities: Angle, Time, Length, Velocity

The icons for the element creation tools are provided in `at.ac.tuwien.big.roverml.design/icons`.

Figure 2: Example RoverML model shown in the graphical RoverML editor

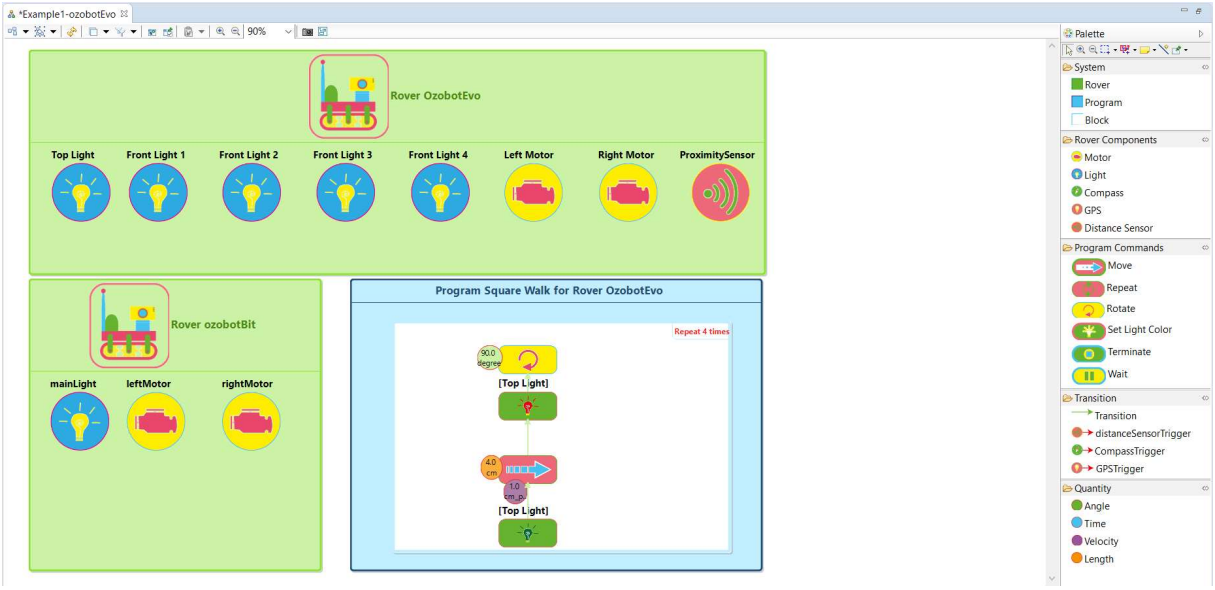
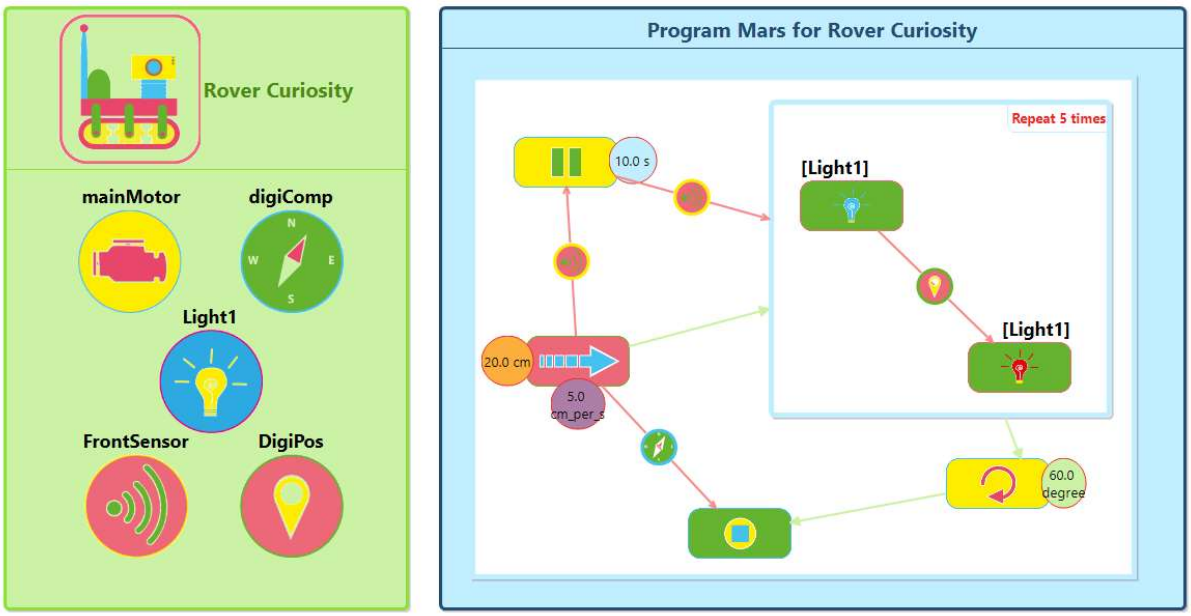


Figure 3: Another example RoverML model



To implement the graphical concrete syntax and graphical (diagram) editor for RoverML, perform the following steps:

1. Setting up your workspace

Follow the instruction of Part A to setup your workspace and launch a new Eclipse instance. In summary, the following steps have to be performed for this:

- Your workspace has to contain the following projects (imported from the assignment resources):
    1. `at.ac.tuwien.big.roverml`
    2. `at.ac.tuwien.big.roverml.edit`
    3. `at.ac.tuwien.big.roverml.editor`
    4. `at.ac.tuwien.big.roverml.xtext`
    5. `at.ac.tuwien.big.roverml.xtext.ide`
    6. `at.ac.tuwien.big.roverml.xtext.ui`

- Launch a new Eclipse instance using the launch configuration "`ME Lab 2 Runtime Eclipse.launch`" located in the project `at.ac.tuwien.big.roverml.xtext`. In the remainder of this document, the newly launched Eclipse instance is called "Runtime Eclipse Instance".

Next, you have to import the following projects into the Runtime Eclipse Instance:

1. `at.ac.tuwien.big.roverml.design`
   You will use this project, to develop the graphical concrete syntax and graphical editor of RoverML.
2. `at.ac.tuwien.big.roverml.design.examples`
   You will use this project to test your graphical editor.

Lastly, switch into the *Sirius* perspective by selecting *Window → Perspective → Open Perspective → Other… → Sirius*. This perspective provides the *Model Explorer* and *Interpreter* views, which are useful for developing Sirius viewpoint specification models.

2. Developing your graphical editor

Define the graphical concrete syntax of RoverML with Sirius by extending the viewpoint specification model `roverml.odesign`, which is located in the project `at.ac.tuwien.big.roverml.design` in the folder `description`. Documentation about how to use Sirius can be found in the lecture slides and in the Sirius documentation[5].

3. Testing your graphical editor

To test your graphical editor, open the diagram "Example1-ozobotEvo" defined in the file `representations.aird` located in the project `at.ac.tuwien.big.roverml.design.examples` (you need to be in the *Sirius* perspective to be able to unfold the content of `representations.aird` in the *Model Explorer*). This diagram is mapped to the RoverML model defined in the file `Example1-ozobotEvo.roverml`. Thus, the diagram will be automatically updated with representations of model elements for which you have correctly defined node/edge mappings. Furthermore, the tool palette will show the element creation tools that you have correctly defined. Updates on your Sirius view point specification model `roverml.odesign` will be automatically reflected in the opened diagram.

For testing your element creation tools, you can use the diagram "Creation Tool Test RoverML" also defined in `representations.aird`. This diagram is mapped to the RoverML model defined in `CreationToolTestModel.roverml`. If you add new elements to the diagram canvas by using the tool palette, this RoverML model has to be correctly updated. For instance, if you create a new Light with the tool palette, a new Light element must be added to the model.

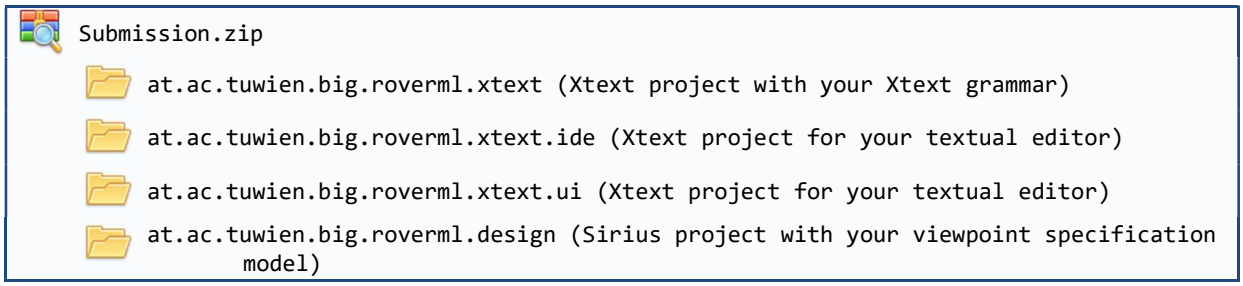
Hint: You can create additional RoverML models using the RoverML tree editor (*File → New → Other… → Example EMF Model Creation Wizards / RoverML Model*). To create a diagram for a new RoverML model, expand the model in the *Model Explorer*, right-click on the System element, select *New Representation → new roverml*, enter a diagram name, and hit *OK*. Your graphical editor will be automatically opened for the example model.

[5] Sirius documentation: http://www.eclipse.org/sirius/doc/
Sirius tutorials: https://eclipse.org/sirius/getstarted.html

Submission & Assignment Review

Upload the following components in TUWEL:

You have to upload one archive file, which contains the following projects:



For exporting these projects, select *File* → *Export* → *General/Archive File* and select the projects.

Make sure to include all resources that are necessary to start your editors!

Thus, if you make changes on any other project that was provided with the assignment resources or implement additional projects, make sure to include them in your submission.

At the assignment review you will have to present your solution, in particular, your Xtext grammar, the implemented scoping support, as well as your Sirius viewpoint specification model. You also have to show that you understand the theoretical concepts underlying the assignment.

All group members have to be present at the assignment review. The registration for the assignment review can be done in TUWEL. The assignment review consists of two parts:

- Submission and **group evaluation:** 20 out of 25 points can be reached.
- **Individual evaluation:** Every group member is interviewed and evaluated separately. The remaining 5 points can be reached. If a group member does not succeed in the individual evaluation, the points reached in the group evaluation are also revoked for this student, which results in a negative grade for the entire course.