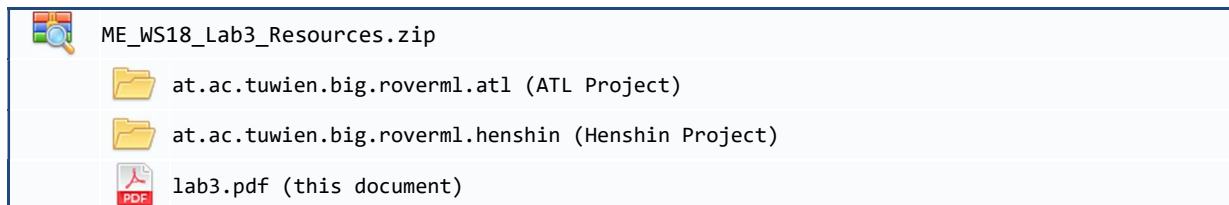


Model Engineering Lab 188.923 IT/ME VU, WS 2018/19	Assignment 3
Deadline: Upload (ZIP) in TUWEL until Sunday, December 16 th , 2018, 23:55 Assignment Review: Wednesday, January 16 th , 2018	25 points

Model-to-Model Transformation

The goal of this assignment is to develop model-to-model transformations for the *Rover Modeling Language* (RoverML) using ATL and Henshin. In Part A of the assignment, you will develop an ATL transformation that translates RoverML models into *Unified Modeling Language* (UML) models. In addition, you will also use UML profiles to extend some elements of UML to allow transferring of some information from the RoverML models. In Part B of this assignment, you will develop a Henshin transformation that applies refactoring on RoverML models.

Assignment Resources



Setting up your workspace

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 course¹.

Import the projects provided in the assignment resources into your workspace. To do so, select *File* → *Import* → *General/Existing Projects into Workspace* → *Select archive file* → *Browse*. Choose the downloaded archive *ME_WS18_Lab3_Resources.zip* and import the following projects:

- at.ac.tuwien.big.roverml.atl
- at.ac.tuwien.big.roverml.henshin

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=523666>

Part A: ATL Transformation

In the first part of this assignment, you have to develop an ATL model transformation to translate RoverML models (source models) into UML models (target models). Rover systems and rovers will be transformed into package diagrams and rover programs will be transformed into activity diagrams. UML² is used to specify, visualize, and document models of software systems, including their structure and design. It can be used for business modeling and modeling of other non-software systems too.

Your task is to implement an ATL model transformation that transforms RoverML models into UML models. The goal of this ATL model transformation is to allow the exchange of information captured in RoverML models with engineering tools that support UML.

Your transformation has to implement the following strategies for mapping RoverML model elements (source models) to UML model elements (target models):

RoverML Concept	UML Concept
RoverSystem	<i>Package</i> . The <i>packaged elements</i> are the system's rovers and programs. Apply the Rover profile to this package.
Rover	<i>Package</i> . The <i>name</i> of the package should be the name of the rover. The <i>packaged elements</i> are the components of the rover.
RoverProgram	<i>Activity</i> . The <i>name</i> of the activity should be the name of the program. The activity <i>owns the node</i> created from the program's block and belongs to the <i>package</i> of the rover the program has referenced in RoverML. The <i>group</i> of the activity is the block of the original program.
Component	<i>Component</i> . The <i>name</i> of the components should be the name of the original components. Apply the Actuator and Sensor stereotypes to the components accordingly. The last sensed value of sensors does not need to be transformed.
Block	<i>Structured Activity Node</i> . Its <i>nodes</i> are the commands of the block and its <i>edges</i> are the transitions of the block. If the block was a repeat block, apply the Repeat stereotype and set its count value.
Transition	<i>Control flow</i> . The control flow's <i>name</i> , <i>source</i> and <i>target</i> are taken from the original transition. Apply the Transition and TriggeredTransition stereotypes accordingly.
Command	<i>Opaque Action</i> . The <i>name</i> is taken from the original command. Each command is mapped as an Opaque Action Node, but a different stereotype for each command is available and should be applied accordingly. In addition, you need to use the stereotypes to add properties to the nodes. The value of the properties are the values of the Quantities contained in each command (e.g. <i>duration</i> for the <i>Wait</i> command). The units of the quantities do not need to be transformed.
Terminate	<i>Activity Final Node</i> . Terminate is transformed differently than the other commands. Set the <i>name</i> of the node to the name of the original command and apply the Terminate stereotype.

² UML website: <http://www.uml.org/>

Additional Requirements

Besides implementing the mapping from RoverML to UML given above, your ATL model transformation has to meet the following requirements:

- Apply the Rover UML profile and its stereotypes accordingly
- Create a helper function for retrieving a stereotype from the profile by its name
- Create an abstract rule for the transformations of commands
- Avoid imperative code (*do* section) as much as possible, i.e. use it only for the application of the profile, applying stereotypes and setting their values

UML Profiles

UML profiles provide a way of extending the UML meta model to adapt it to domain specific needs. A profile defines stereotypes which are applied to the UML meta classes. This can be used to e.g. expand an existing UML element with an additional property.

In this assignment, you will use a UML profile to transfer some additional information to the target model. The profile and its stereotypes are already defined. You only need to apply them during the ATL transformation.

Using profiles in ATL transformations

In addition to the source and target model you need also to specify to use a Profile in the header of the .atl file.

```
create OUT: UML from IN: RML, P: Profile;
```

The profile to use during the transformation is specified in the launch configuration of the transformation.

Applying a profile

To apply a profile, use the *applyProfile()* function of the targeted UML element. The profile to be applied needs to be supplied as parameter.

```
target.applyProfile(profile);
```

If only one profile is applied, it can be retrieved like this:

```
Profile!Profile.allInstances() -> first()
```

Applying a stereotype

To apply a profile, use the *applyStereotype()* function of the targeted UML element. The stereotype to be applied needs to be supplied as parameter.

```
target.applyStereotype(stereotype);
```

A stereotype can be retrieved from a profile like this (where 'name' is the name of the stereotype you want to retrieve):

```
Profile!Stereotype.allInstances() -> any(s | s.name = 'name')
```

Setting values of stereotypes

To set the value of a property use the `setValue()` function of the targeted UML element.

```
target.setValue(appliedStereotype, 'propertyname', value);
```

If only one stereotype is applied, the applied stereotype of a targeted UML element can be retrieved like this:

```
target.getAppliedStereotypes() -> first()
```

Additional information is available from the Javadoc of the UML Java API for Stereotypes (<http://download.eclipse.org/modeling/mdt/uml2/javadoc/3.1.0/org/eclipse/uml2/uml/class-use/Stereotype.html>).

1. Developing the ATL model transformation

Define the ATL model transformation in the file `RoverML2UML.atl` located in the root folder of the project `at.ac.tuwien.big.roverml.atl`. This is the only file you need to change for implementing the ATL model transformation. Some rules are **(partly)** provided to show the usage of Profiles and Stereotypes. Keep in mind that you may still need to complete these rules! The order of the rules is also important, e.g. you may need to insert additional rules above an already existing rule.

Resources of `at.ac.tuwien.big.roverml.atl`:

- **Folder input:** Provides example models which will be processed by your ATL model transformation.
- **Folder metamodel:** Provides the RoverML model as `.ecore` file. A graphical representation is available via the `.aird` file.
- **Folder output:** Models transformed by your ATL model transformation are stored in this folder. Also provides UML models that are expected to be produced as output of your ATL model transformation.
- **Folder profile:** Contains the UML profile you will need to use in the transformation process. The profile contains information about the stereotypes and their properties. You will need to have a look at the profile to determine the names of stereotypes and the names of their properties.
- **Folder runconfigs:** Provides launch configurations for executing your ATL model transformation on the provided example RoverML models.
- The ATL file `RoverML2UML.atl` that you have to use for implementing the requested model transformation is found in the root of the project.

2. Testing the ATL model transformation

For testing your ATL model transformation, we provide the example RoverML models `Example1-ozobotEvo.xmi`, `Example2-curiosity.xmi` and `Example3-polarSysRover.xmi`. When executing your ATL model transformation for these example RoverML models, it should produce UML models corresponding to the provided UML models `Example1-ozobotEvo-expected.uml`, `Example2-curiosity-expected.uml` and `Example3-polarSysRover-expected.uml`.

To execute your ATL model transformation for the example RoverML models, use the provided launch configurations:

- RoverML2UML_Example1.launch
- RoverML2UML_Example2.launch
- RoverML2UML_Example3.launch

These launch configurations will execute the ATL model transformation for the provided example RoverML models and produce the UML models Example1-ozobotEvo.uml, Example2-curiosity.uml and Example3-polarSysRover.uml. To launch a transformation, right-click on RoverML2UML_ExampleX.launch and select *Run As > RoverML2UML_ExampleX*. Transforming example 2 may not work until more than the provided rules are implemented.

Compare the produced UML models (*.uml) with the expected UML models (*-expected.uml). You can compare the models either manually by opening them in the tree-based editor or by using the compare tool. To use the tool, select two models in the workspace, right-click one of them and select *Compare With > Each Other*.

If you want to inspect the provided input models, you have to register the RoverML metamodel once. To do so, right-click on the .ecore file found in the metamodel folder and select *EPackages registration → Register EPackages* into repository. Now you can open any RoverML model by right-clicking on it and selecting *Open With > Sample Reflective Ecore Model Editor*.

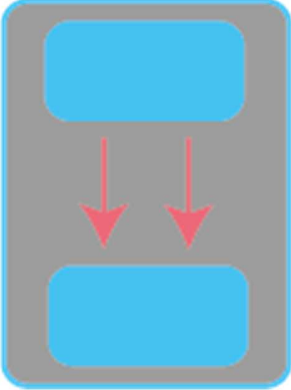
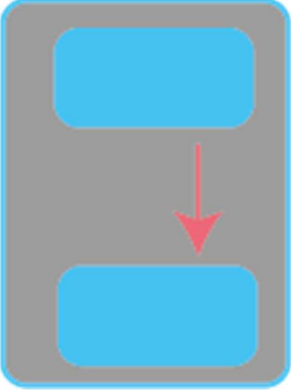
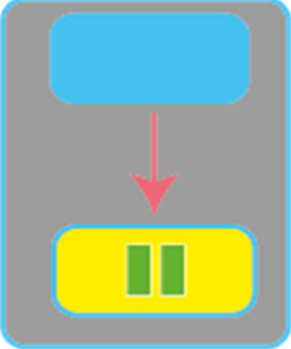
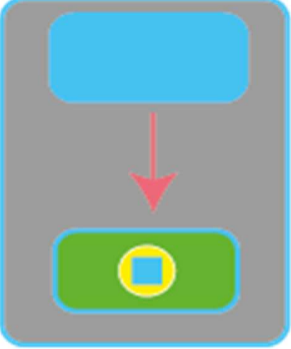
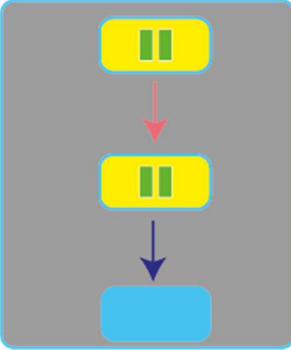
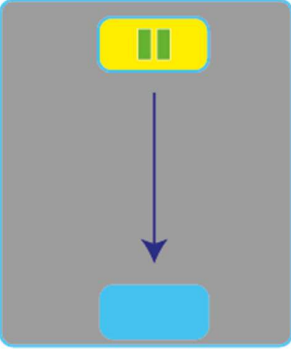
Note that the ordering of elements in the produced UML models is not strict. Thus, your ATL model transformation may produce a UML model where the ordering of elements is different than the ordering in the expected UML model. The correct containment hierarchy is, however, important.

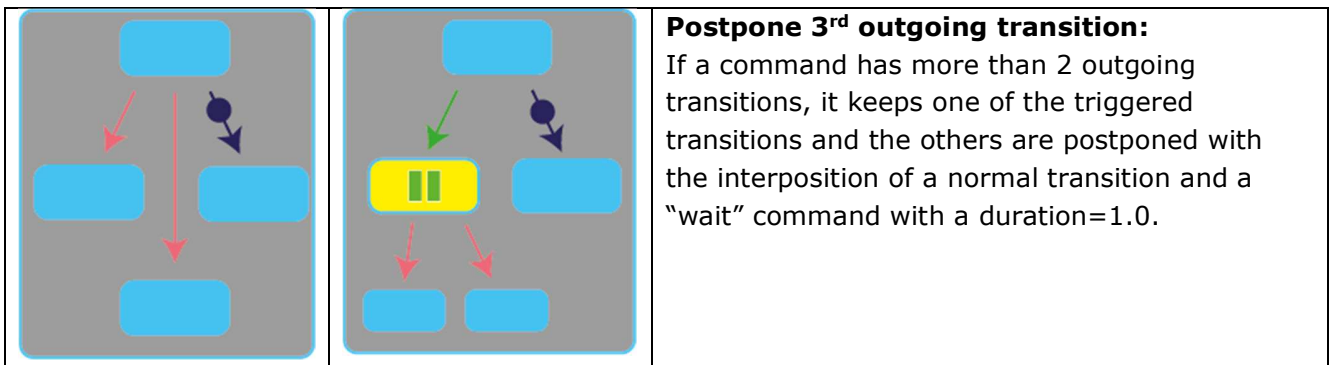
Note that your ATL model transformation should be defined in such a way, that it works properly for all possible RoverML models, i.e., produce for any RoverML model a UML model following the mapping given above. To ensure this, additional testing from your side is required. The UML models should also be valid, i.e. show no errors if you validate them (To validate a model, open it, right-click the root element and select "Validate").

Part B: Henshin Transformation

In the second part of this assignment, you have to refactorize RoverML models using Henshin transformations. This refactorization can be composed of several rules and units, but a single execution of the refactorization should transform the whole model.

The refactorization consists of following operations:

Before	After	Transformation
		Delete duplicates: If two transitions have the same source and target, one of them must be deleted.
		Wait to terminate: If a wait command has no outgoing transitions, it must be transformed into a terminate command.
		Wait Fusion: If a wait command has only one outgoing transition connected to another wait command with exactly one outgoing transition and no further incoming transitions, they have to be merged into one wait command. The duration must be summed up. You can assume that duration is always in seconds.



The refactorization has to consider all possible cases.

Make sure that the RoverML models created by your Henshin transformations are valid, i.e., conform to the RoverML metamodel and fulfill all OCL constraints. To validate a RoverML model, open the model in the Sample Reflective Ecore Model Editor and selecting from the menu *Sample Reflective Editor > Validate*.

1. Developing the Henshin transformation

Define the Henshin transformations in the file `roverml.henshin` located in the project `at.ac.tuwien.big.roverml.henshin` in the folder `henshin`. You can use the file `roverml.henshin_diagram` (located in the same folder) to define the transformation using the graphical Henshin editor. These are the only files you need to change for implementing the Henshin transformation.

HINT: Save your Henshin diagram often. If the graphical editor behaves strangely, do *not* press Undo/Ctrl-Z, but close the diagram window and re-open it and try to create the elements in a different way.

Resources of `at.ac.tuwien.big.roverml.henshin`

- **Folder `henshin`:** Provides the Henshin files `roverml.henshin` and `roverml.henshin_diagram` that you have to use for implementing the requested Henshin transformation.
- **Folder `metamodels`:** Provides the metamodel of RoverML (`roverml.ecore`).
- **Folder `models`:** Provides example RoverML models that should be processed by your Henshin transformation as input, and refactored versions of these models that are expected to be produced as output of your Henshin transformation.

2. Testing the Henshin transformation

For testing your Henshin transformation, we provide the example RoverML models `Example1.xmi` and `Example2.xmi` (folder `models`). When executing your Henshin model transformation for these example RoverML models, it should produce RoverML models corresponding to the provided RoverML models `Example1_expected.xmi` and `Example2_expected.xmi`. Don't forget to register the metamodel from the folder `metamodels`.

To execute your Henshin model transformation for the example RoverML models, right-click on the rule/unit you want to test in the `roverml.henshin` file and select *Apply Transformation* and select the example you want to transform in *Input Model*; check the box *Open Compare* to automatically compare the transformation. You may need to refresh the `models` folder to see the produced models. To do so, right-click on the `models` folder and select *Refresh* (or press F5).

Compare the produced models (`*_transformed.xml`) with the expected models (`*_expected.xml`). You can compare the models either manually by opening them in the tree-based editor or automatically using EMF Compare. To use EMF compare, select two models in the workspace, right-click one of them and select *Compare With > Each Other*.

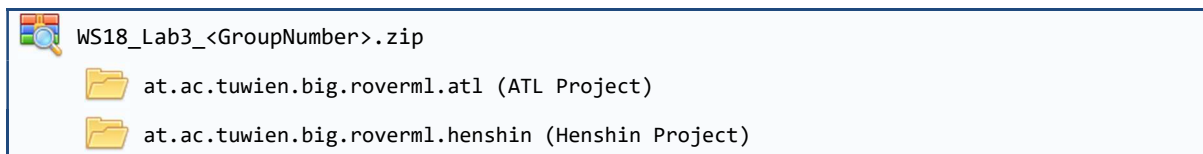
Additional Information

- Literature to solve this assignment is provided in the TUWEL course.
- You can find additional information, examples, and use cases for ATL at <https://www.eclipse.org/atl/documentation/>. All relevant ATL concepts for this assignment can be found in the ATL user guide.
- You can find additional information, examples and use cases for Henshin at <https://www.eclipse.org/henshin/>. A good beginner tutorial can be found here: <https://www.eclipse.org/henshin/examples.php?example=bank>.

Submission & Assignment Review

Upload the following components in TUWEL:

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



For exporting these projects, select *File* → *Export* → *General/Archive File* and select the projects. Make sure that all folders and files contained by the projects are selected.

Assignment Review:

At the assignment review, you will have to present your ATL transformation and your Henshin transformation. 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, resulting in a negative grade for the entire course.