Model Engineering Lab 188.923 IT/ME VU, WS 2011/12	**Assignment 2**
Deadline: Upload in TUWEL until Monday, December 05, 2011, 23:55 Assignment Review: Wednesday, December 07, 2011	25 Points

Overview

In the course of this lab, you will develop a family of object-oriented modeling languages that allows its users to specify the structure, the behavior and the execution of a software system. In the first lab, you developed the *abstract syntax* of the modeling language, called SOOML, to represent such a software system with EMF. In Lab 2, your goal is to build a *concrete textual syntax* using Xtext for SOOML. In Lab 3, you will develop a model-to-model transformation, which translates SOOML models to the so-called *Simple Object-oriented Programming Language* (*SOOPL*). Finally, in Lab 4, you will develop a model-to-code transformation that generates executable Java code from SOOPL models.

Textual Concrete Syntax

The goal of this assignment is to develop the textual concrete syntax for the *Simple Object-Oriented Modeling Language* (*SOOML*) using Xtext. Therefore, in Part A, you have to develop the **Xtext grammar** and generate a SOOML editor. Your task in Part B is to specify **five OCL constraints** for ensuring the validity of SOOML models.

Part A: Development of the Metamodel for SOOML

The textual concrete syntax to be developed in the course of this assignment is specified in terms of an example. In Figure 1, we depict an example SOOML file, which exactly represents the SOOML model depicted in Figure 2 in the textual concrete syntax that you have to develop using Xtext. This SOOML file is also provided in the assignment resources [1] (*ME_WS11_Lab2_Resources/ME_WS11_Lab2_Test/test.sooml*) available in TUWEL. In the assignment resources, we also provide the sample solution of Lab 1, i.e., the SOOML metamodel project (*ME_WS11_Lab2_Resources/ME_WS11_Lab1*). You must use the metamodel from this project and not the metamodel you built in Lab 1 yourself.

First of all, import the two Eclipse projects from the assignment resources provided in TUWEL into your Eclipse workspace. Therefore, download the archive *ME_WS11_Lab2_Resources.zip* from TUWEL and, in Eclipse, select *File → Import → General/Existing Projects into Workspace → Select archive file → Browse*. Select the downloaded archive *ME_WS11_Lab2_Resources.zip*, click next, and import both projects called *ME_WS11_Lab1* and *ME_WS11_Lab2_Test*.

To create the Xtext projects and test it, proceed as follows.

- Create the Xtext projects for the existing SOOML metamodel by selecting *File → New → Other → Xtext/Xtext Project From Existing Ecore Models*. In the Xtext wizard click *Add* and select *sooml.genmodel*. Select *Package – sooml* as entry rule in the drop-down box below and click *Next*. Type *at.ac.tuwien.big.me.sw11.SOOMLDsl* in the project name input box and as language name and change the extensions box from *mydsl* to *sooml* and hit F*inish*. Now, three projects should have been automatically generated.
- Xtext automatically generated a default Xtext file for you, which you have to modify in order to meet the requirements of this assignment. More precisely, open

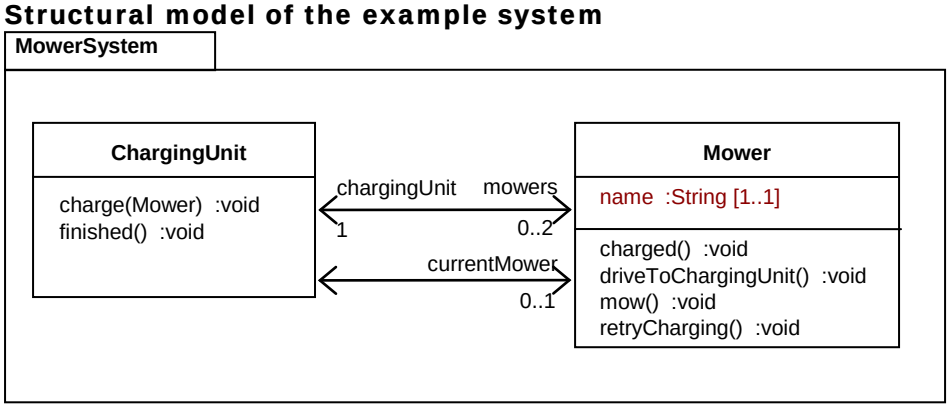
the Xtext file called *SOOMLDsl.xtext*, which is located in the project called *at.ac.tuwien.big.me.sw11.SOOMLDs* in the folder *src/at.ac.tuwien.big.me.ws11* and start to adapt it such that the textual syntax exemplified in Figure 1 is specified in the Xtext grammar. Consult the Xtext documentation [3] and the lecture slides about Xtext, which are available in TUWEL, to achieve this task.

- To test your current Xtext specification in *SOOMLDsl.xtext*, you first have to generate the editor from your *SOOMLDsl.xtext* file and try it. Therefore, right-click the file called *GenerateSOOMLDsl.mwe2*, which is located next to the *SOOMLDsl.xtext* and select *Run As → MWE2 Workflow*. This will start the editor code generation. In the first generation run, you will have to accept to download the ANTLR 3 parser generator by type *y* in the console view. Please note that after each change you made to *SOOMLDsl.xtext*, you have to run the workflow file again to update the generated editor code!
- For starting the generated editor, open the *META-INF/MANIFEST.MF* and hit the play button at the upper right in the opened editor. This should start another Eclipse instance, in which you can create a new empty project and create a new file *my_test.sooml* in this project. Right-click this file and select *Open With → SOOMLDsl Editor*. Now test the opened editor and also use the auto completion (Ctrl+Space) to check whether your editor works as expected and whether the textual syntax corresponds to the one specified in Figure 1. Therefore, you can also paste the contents of *test.sooml* from the assignment resources (in the project named *ME_WS11_Lab2_Resources* you already imported) into your *my_test.sooml* file and check whether there are any error markers in your editor.
- Once, you believe that you correctly finished the Xtext grammar and generated a correctly functioning editor that indicates no errors for contents of *test.sooml* from the assignment resources, extract the model-based representation of your sooml file as described below and compare it with the model in *test.xmi* from the assignment resources (*ME_WS11_Lab2_Resources/test.xmi*). This provided model (*test.xmi)* exactly represents the intended model-based representation that should result from the specified *test.sooml* file. To extract the model from your *my_test.sooml*, copy this file to the project ME_WS11_Lab2_Resources, right-click the JUnit test class at.ac.tuwien.big.me.ws11.lab2.test.ExtractModelTest and select *Run As → JUnit Plug-In Test*. This will extract the file *my_test.xmi* from the file *my_test.sooml*. You have to refresh the project *ME_WS11_Lab2_Resources* afterwards (select it and hit F5) so that the generated *my_test.xmi* will appear in the package explorer in Eclipse. Now, you can compare the contents of *my_test.xmi* with the contents of *test.xmi*. To open the XMI files in the tree-based editor, you first have to register the SOOML metamodel. Therefore, right-click *ME_WS11_Lab1/model/sooml.ecore* and select *Epackages registration → Register epackages into repostitory*. You can also use a model-based comparison tool called EMF Compare to compare these two models: select both XMI files, right-click and select *Compare With → Each other*.

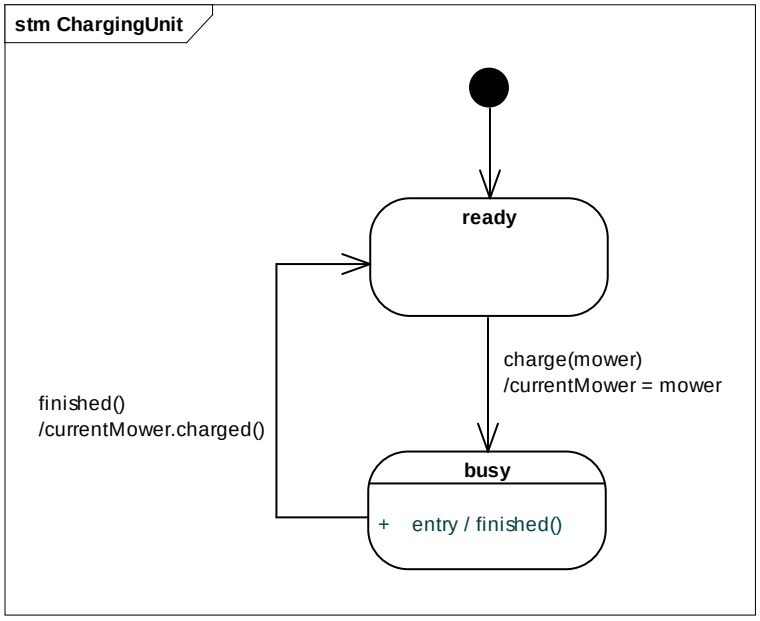
```
package MowerSystem {
  class chargingUnit {
    ref Mower mowers ( 0 .. 2 ) oppositeOf chargingUnit
    ref Mower currentMower ( 0 .. 1)
    op finished ()
    op charge (Complex Class Mower mower )
    stateMachine {
       state ready {
          charge => busy {
            self -> currentMower = mower;
          }
    }
       state busy onEntryCall finished{
          finished => ready {
            self -> currentMower -> charged;
          }
       }
       -> ready // this indicates the initial state
    }
  }
  class Mower {
    att String name ( 1 .. 1 )
    ref ChargingUnit chargingUnit ( 1 .. 1 ) oppositeOf mowers
    op driveToChargingUnit ()
    op charged ()
    op mow ()
    op retryCharging ()
    stateMachine {
       state charged onEntryCall mow {
          mow => lowBattery
       }
       state lowBattery onEntryCall  driveToChargingUnit {
          driveToChargingUnit => waiting unless self -> chargingUnit inState busy
          driveToChargingUnit => charging unless self -> chargingUnit inState ready {
            self -> chargingUnit -> charge ( mower = self ) ;
          }
       }
       state charging {
          charged => charged
       }
       state waiting onEntryCall  retryCharging {
          retryCharging => lowBattery
       }
       -> charged
    }
  }
}
```

Figure 1: Textual syntax of SOOML

Structural model of the example system



Behavioral model of the example system



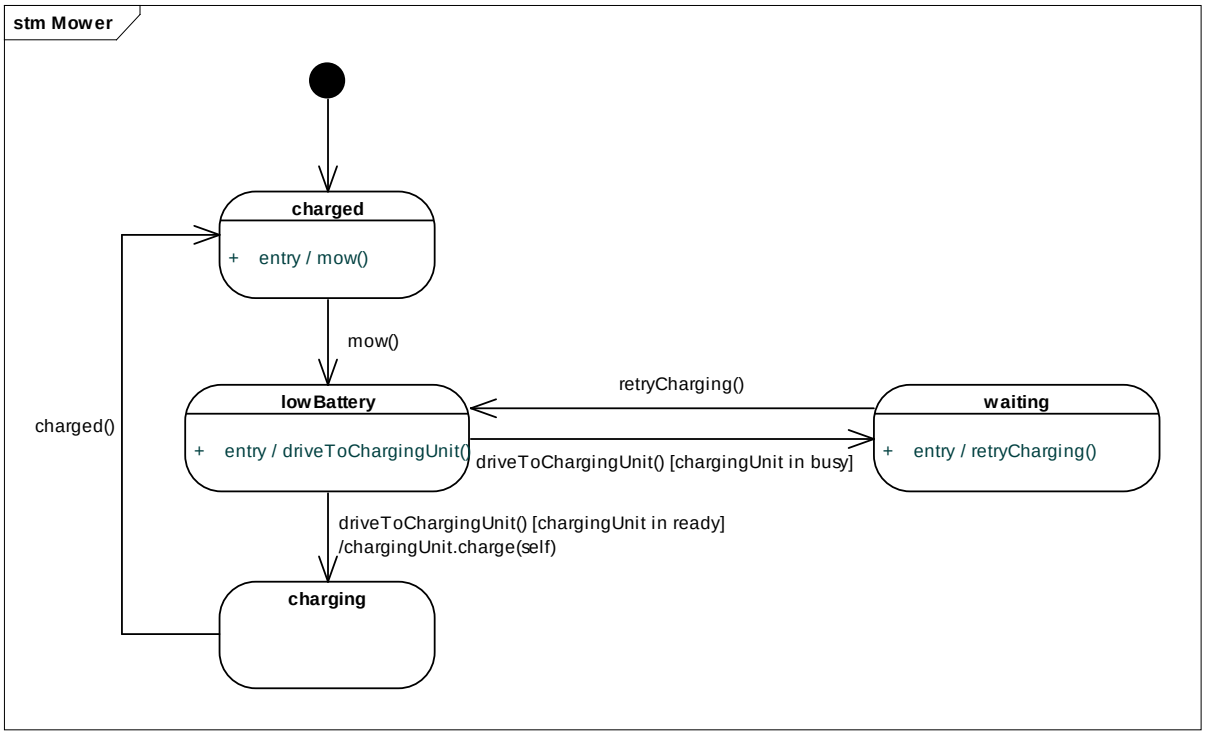


Figure 2: Example SOOML Model

Part B: OCL

To ensure the validity of SOOML models, you have to specify five OCL constraints. In particular, ensure the constraints enumerated in the following in terms of invariants. Invariants must return *true* (if the model is valid) or *false* (if the model is invalid) and must be specified within a context, i.e., a metaclass in the metamodel (e.g., `context Class inv: self.name <> null`).

1. Attributes must not have the data type COMPLEX.
2. The entry operation of a state has to be an operation of the class for which the state is defined.
3. The target state of a transition has to be a state of the same state machine.
4. The two references that define a bidirectional relationship have to reference each other as opposite.
5. The type of a reference must be the class for wich the opposite reference is defined.

Use the *Interactive OCL Console* to test your constraints. Therefore, open the XMI file *ME_WS11_Lab2_Resources/test.xmi* with the Reflective Ecore Editor (remember to register *sooml.ecore* first), open the console view, and select the *Interactive OCL console* using the icon with a windows symbol having a plus symbol (the most right icon in the console view). Now, you can select a context model element in the *Reflective Ecore Editor,* which shows your model*,* and type an OCL constraint in the Interactive OCL console. By hitting enter, the typed constraint is evaluated within the context of the selected model element. It might be best to create SOOML models that purposefully violate the aforementioned constraint and check whether your constraint evaluates to *false* for these models.

Create a file called *ocl.txt*, which contains your OCL constraints, and copy it into the project *at.ac.tuwien.big.me.ws11.SOOMLDsl*.

Submission & Assignment Review

Upload the following components in TUWEL:

One archive file, which contains the three Xtext projects:

- Create a text file called ocl.txt, which contains your OCL constraints from Part B into the project at.ac.tuwien.big.me.ws11.SOOMLDsl.
- Select the three Xtext projects, right-click them, and select *Export → Archive File*. Upload this archive file.

All group members have to be present at the assignment review. 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. Remaining 5 points can be reached [2].

Notes

[1] **Assignment Resources:** For this assignment, we provide two Eclipse projects within one archive file. One project contains the SOOML metamodel you must use. The other project contains test files and a JUnit test for testing your solution in Part A. The archive file can be downloaded from TUWEL.

[2] **Evaluation:** The evaluation of your submission includes the **joint** development of this assignment. If a group member did not participate, (s)he can't reach any points.

[3] **Literature** about Xtext can be found at http://www.eclipse.org/Xtext/documentation.