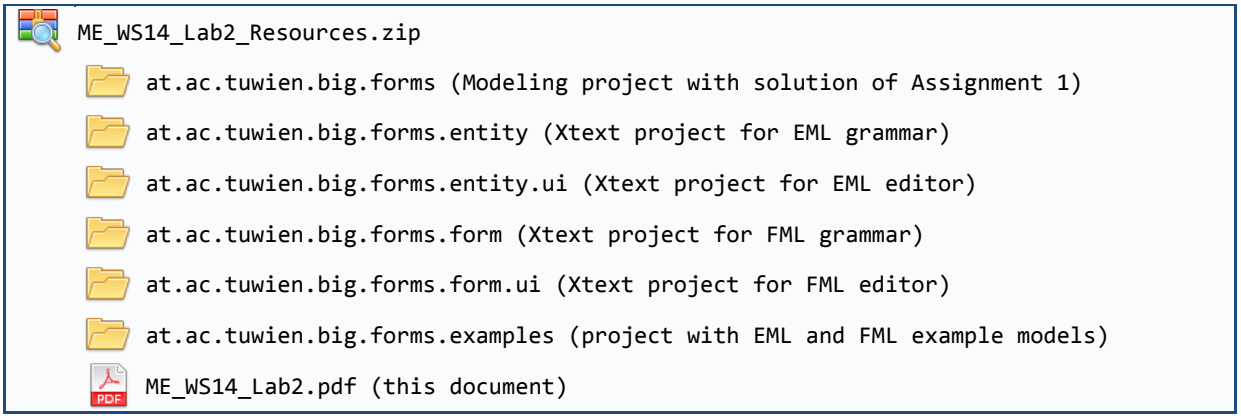
Model Engineering Lab 188.923 IT/ME VU, WS 2014/15	**Assignment 2**
Deadline: Upload (ZIP) in TUWEL until Monday, November 24th, 2014, 23:55 Assignment Review: Wednesday, November 26th, 2014	25 Points

Textual concrete syntax

The goal of this assignment is to develop two textual languages, the *Entity Modeling Language (EML)* and the *Forms Modeling Language* (*FML*), using Xtext[1]. Furthermore you have to implement scoping support for the editors that are generated for these languages. Therefore, in Part A, you have to develop two **Xtext grammars** for EML and FML by applying a meta-model-first approach. In Part B, you implement **scoping support** for the generated EML and FML editors.

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.

Part A: Xtext grammars

In the first part of this assignment, you have to develop two textual concrete syntaxes (grammars) for modeling entities and forms. Both of these grammars must be built upon the Forms meta-model (see Assignment 1), for which we have provided an example solution in the assignment resources (at.ac.tuwien.big.forms/model/forms.ecore).

Do not use the meta-model you have developed in Assignment 1!

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

[1] www.eclipse.org/Xtext/, can be installed using Help → Install Modeling Components

Entity Modeling Language

The Entity Modeling Language (EML) covers the part of the meta-model that is concerned with the definition of entity models. A user must be able to specify entities and enumerations as well as their properties and references using a textual notation. Therefore, you have to develop a grammar for such a notation using Xtext. In this assignment, the grammar you have to develop should follow the example model that is depicted in Figure 1, which is also available in full length in the examples project (`at.ac.tuwien.big.forms.examples/test1.entity`).

Figure 1: Excerpt of an Entity model example in textual concrete syntax EML

```
/**
 * 188.923: Model Engineering Lab, Assignment 2
 * Entity Model
 * - Attributes with an asterisk ('*') are mandatory.
 */
entity Publication {
   id title,
   attribute title* : String,
   attribute keywords : String,
   attribute abstract* : Text,
   attribute doi : String,
   attribute year : Year,
   attribute fromPage : Integer,
   attribute untilPage : Integer,
   attribute publicationType* : None : PublicationType,
   attribute assignedFaculty* : None : Faculty,
   attribute field* : None : ResearchField,
   reference authors : Person[1..-1] opposite-of Person.publications,
   reference proceedings : Proceedings[0..1],
   reference journal : Journal[0..1],
   reference book : Book[0..1]
}
                                                ...
entity PublicationVenue {
   id title,
   attribute title* : String
}

entity Proceedings extends PublicationVenue {
   id title, // title of super-entity: PublicationVenue.title
   attribute publisher* : String,
   attribute volume : Integer,
   attribute isbn : String,
   reference event : Event[0..1],
   reference editors : Person[1..-1]
}
                                                ...
enum PublicationType {
   JA = 'Journal Article',
   BC = 'Book Chapter',
   CP = 'Conference Paper',
   WP = 'Workshop Paper'
}
```

Forms Modeling Language

The Forms Modeling Language (FML) covers the part of the meta-model that is concerned with the definition of form models. A user must be able to specify forms, their containing pages and the respective page elements and conditions using a textual notation. Therefore, you have to develop a grammar for such a notation using Xtext. In this assignment, the grammar you have to develop should follow the example model that is depicted in Figure 2, which is also available in full length in the examples project (`at.ac.tuwien.big.forms.examples/test1.form`).

Figure 2: Excerpt of a Form model example in textual concrete syntax FML

```
/**
 * 188.923: Model Engineering Lab, Assignment 2
 * Forms Model
 * - Syntax for PageElements:
 *   <PageElementType> <PageElement.elementID> "<PageElement.label>"...
 */
 welcome form PublicationForm "Publication" handles Publication {
   page "PublicationPage" {
      text-field PublicationTitleField "Title" handles title,
      text-field PublicationKeywordField "Keywords" format "^([a-zA-Z]+,)+[a-zA-Z]+$"
                                                  handles keywords,
      text-area PublicationAbstractArea "Abstract" handles abstract,
      text-field PublicationYearField "Year" handles year,
      text-field PublicationFromPageField "From page" handles fromPage,
      text-field PublicationUntilPageField "Until page" handles untilPage,
      selection-field PublicationTypeSelection "Type" handles publicationType,
      selection-field PublicationFacultySelection "Assigned faculty" handles assignedFaculty,
      selection-field PublicationResearchSelection "Research field" handles field
   }

   page "Authors" {
      table AuthorTable "Authors" handles authors {
         column AuthorFirstNameColumn "First name" handles firstname,
         column AuthorLastNameColumn "Last name" handles lastname,
         column AuthorEMailNameColumn "E-Mail" handles email
      } edits PersonForm
   }

   page "Proceedings" {
      list ProceedingsList "Proceedings" handles proceedings edits ProceedingsForm
   } with composite-condition ProceedingsCondition:
         (attribute-condition ShowIfCP: publicationType == "Conference Paper" ? Show or
          attribute-condition ShowIfWP: publicationType == "Workshop Paper" ? Show) ? Show

   page "Journal" {
      list JournalList "Journal" handles journal edits JournalForm
   } with attribute-condition ShowIfJA: publicationType == "Journal Article" ? Show

   page "Book" {
      list BookList "Book" handles book edits BookForm
   } with attribute-condition ShowIfBC: publicationType == "Book Chapter" ? Show
}

form PersonForm "Person" handles Person {
   page "Person Details" {
      text-field FirstNameField "First name" handles firstname,
      text-field LastNameField "Last name" handles lastname,
      text-field EMailField "E-Mail" handles email,
      selection-field ExternalPersonSelection "Faculty-external person" handles external,


      selection-field EmployedFacultySelection "Employed faculty" handles faculty
         with attribute-condition HideIfExternal: external == "true" ? Hide
   }
}
                                          ...
form EventForm "Event" handles Event {
   page "Event Details" {
      text-field EventTitleField "Title" handles title,
      text-field EventCityField "City" handles city,
      text-field EventCountryField "Country" handles country,
      date-selection-field EventFromSelection "From date" handles fromDate,
      date-selection-field EventUntilSelection "Until date" handles untilDate,
      time-selection-field EventAdmissionSelection "Admission from" handles admissionFrom
   }
}
```

To implement these two languages, please perform the following steps.

1. Setting up your workspace

As mentioned in the beginning, both languages are based on the meta-model that needed to be defined in Assignment 1. We provide one solution to this assignment as well as the two skeleton projects for the grammars as importable Eclipse projects.

Import projects

To import these project in Eclipse select *File → Import → General/Existing Projects into Workspace → Select archive file → Browse*. Choose the downloaded archive *ME_WS14_Lab2_Resources.zip* and import the projects called `at.ac.tuwien.big.forms`, `at.ac.tuwien.big.forms.entity`, `at.ac.tuwien.big.forms.entity.ui`, `at.ac.tuwien.big.forms.form`, and `at.ac.tuwien.big.forms.form.ui`.

After you have successfully imported the five projects, make sure that the first one has the "Modeling" nature applied or apply it if necessary (Right click on project → *Configure → Add Modeling Project Nature*). The other four projects serve as base for your grammar definitions and editors and should have the Xtext Nature applied.

Register form meta-model

In the last step before you start implementing your grammars, you should register the forms meta-model in your Eclipse instance, so that the Xtext editor with which you develop your grammar knows which model elements you want to use. You can register your meta-model by clicking right on at.ac.tuwien.big.forms/model/forms.ecore → *EPackages registration → Register EPackages into repository*.

After performing these steps, you should have five projects in your workspace without any errors and some warnings about missing plugin.xml files, which will be generated later.

2. Developing your Xtext grammars

Define the grammar for EML in the `Entity.xtext` file located in the `at.ac.tuwien.big.forms.entity` project and the grammar for FML in the `Form.xtext` file located in the `at.ac.tuwien.big.forms.form` project.
Documentation about how to use Xtext can be found in the Xtext documentation[2] or in the lecture slides.

Hint: Make use of the `ID` rule defined in the base grammar `org.eclipse.xtext.common.Terminals` to specify the ids or names of different model elements, e.g., the name of an entity or the elementId of a page element.

Hint: When providing cross-references to other model elements, make sure to use the `QualifiedName` as the parsed syntax, e.g., `referenceToEntity=[Entity|QualifiedName]`. This will avoid confusion if multiple elements with the same name or id can be referenced.

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

3. Starting your editors

Generate Xtext artifacts

After you have finished developing your Xtext grammars (`Entity.xtext` and `Form.xtext`) you can automatically generate the textual editors. Each grammar project is accompanied by a so called model workflow file (`GenerateEntity.mwe2` and `GenerateForm.mwe2`), which orchestrates the generation of the editor. By default, the workflow will generate all the necessary classes and stub classes for extension in the grammar project and in the related ui-project.

Note: In the workflow file you can specify whether the generator should create Xtend[3] or Java stub classes for extension (flag `generateXtendStub`). In Part B of this assignment you have to implement Scoping by implementing one of these stubs. If you are not familiar with the syntax of Xtend, you can leave the setting as it is (we have changed the default setting). Otherwise you can change the flag to `true` and generate Xtend stubs.

To run the generation, select the respective workflow file 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 have to 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:

```
*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)
```

Please 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 editors, we have already provided a launch configuration located in the `at.ac.tuwien.big.forms.form.ui` project called `EclipseIDE_AllPlugins.launch`. You can have a look at the configuration by right-clicking on that file and select *Run As → Run Configurations…*. Basically what this launch configuration states is that a new Eclipse instance should be started that contains all target plugins and the plugins that you have developed in your workspace (the grammars and the editors). Furthermore, we added some arguments for the virtual machine (VM arguments) to provide additional memory as otherwise the editors might run into a memory problem. You can run the launch configuration by pressing *Run*.

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 *.entity* or *.form* in this project. Right-click the created **.entity* or **.form* file and select *Open With → Entity Editor/Form Editor*. If you are asked to add the Xtext nature to the project hit *Yes.* Now you can start modeling your entities or forms and test the opened editors.

[3] Xtend: http://www.eclipse.org/xtend/

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 actually referenced in a model by using the linking feature. For this just hold Ctrl and click with the mouse on the cross-reference.

Check examples

In the newly started Eclipse instance, you can import some example models, which we have included in the `at.ac.tuwien.big.forms.examples` project. Just import the project in the workspace and make sure that the project has the Xtext nature applied (see "Setting up your workspace").

test1.entity	The complete entity model file as partially depicted in Figure 1. This file should show no error or warning markers.
test1.form	The complete form model file as partially depicted in Figure 2. This file should show no error or warning markers.

Part B: Development of scoping support

In the grammars you have developed, you should have defined some cross-references, to other model elements, e.g., an attribute page element refers to a specific attribute of an entity. When testing your editor, you will notice that whenever there is a cross-reference, the content assist will provide all elements of the respective type that the editor can "find". Which elements the editor provides is defined in the scoping[4] 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 the developers can define their own scope. This stub can be found in the grammar projects in the package scoping (`EntityScopeProvider` and `FormScopeProvider`). Your task is to implement the following scoping behavior for the two languages, EML and FML.

EML Scoping

The id of an entity must be an attribute that is either defined by the entity itself or one of its super types (remember, the inheritance hierarchy can have arbitrary many levels).

Example: The id of the entity Proceedings comes from the super-type PublicationVenue.

FML Scoping

An attribute page element has to reference an attribute of the entity the containing form references.

Example: The text-field PublicationTitleField is only allowed to handle the attributes of the entity Publication (title, keywords, …, fields).

[4] Xtext Scoping: http://www.eclipse.org/Xtext/documentation.html#scoping

A relationship page element has to reference a relationship of the entity the containing form references.

Example: The table AuthorTable in page "Authors" is only allowed to handle relationships of the entity Publication (authors, proceedings, journal, book).

An attribute value condition has to reference an attribute of the entity the containing form references.

Example: The condition ShowIfJA in page "Journal" can only reference attributes of the entity Publication (title, keywords, …, fields).

A selection field is only allowed to reference an attribute of type Boolean or an attribute which has a reference to an enumeration.

Example: The selection-field ExternalPersonSelection in page "Person Details" is only allowed to reference the attributes external and faculty of the entity Person.

A relationship page element has to reference a form of the same form model as editing form.

Example: The table AuthorTable can only reference the editing forms PublicationForm, PersonForm, ProceedingsForm, JournalForm, BookForm, or EventForm.

A column of a table can only reference attributes of the entity of the form the table edits.

Example: The column AuthorFirstNameColumn can only edit the attributes of the entity Person (title, keywords, …, fields), because the table AuthorTable defined the PersonForm as editing form, which references the entity Person.

Hint: By default, 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 under declarative scoping in the Xtext documentation[5] or in the Java documentation of the `Abstract-DeclarativeScopeProvider` class, which is the super class of your scoping stubs.

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

Check examples

In the newly started Eclipse instance you can check the remaining models provided by the examples project 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.

test2.form	In this example, no attribute value conditions have been provided for the composite condition of the page "Proceedings". Thus, your editor should show an error for this composite condition.
test3.form	In this example, the selection field "Faculty-external person" of the page "Person Details" references an invalid attribute, which should be indicated by an error in your editor when opening this file.

[5] Declarative scoping: http://www.eclipse.org/Xtext/documentation.html#_62

test4.form	In this model the wrong editing forms are set for the relationship page elements contained by the pages "Authors" and "Proceedings". The editor should show the violated OCL Constraint error "EditingFormRefersTo-RelationshipTarget" for the table "Authors" and the list "Proceedings".
test5.form	This example has invalid references for attributes of the form "Journal".
test6.form	In this model, the table AuthorTables in page "Authors" refers to the journal relationship and edits the correct JournalForm. However, the columns in the table refer to attributes of the entity Person and not of entity Journal. A "TableColumnRefersToTargetEntityFeature" constraint violation should be shown in you editor.

Submission & Assignment Review

Upload the following components in TUWEL:

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

Entity Model Projects	Form Model Projects
at.ac.tuwien.big.forms.entity at.ac.tuwien.big.forms.entity.ui	at.ac.tuwien.big.forms.form at.ac.tuwien.big.forms.form.ui

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

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

Therefore, make sure to also include the files that have been automatically generated by the workflow file. If you are unsure, test your exported projects by importing them into a new workspace and start a new Eclipse instance from there to test your editors again.

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. 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.