

<b>Model Engineering Lab</b> 188.923 IT/ME VU, WS 2014/15	<b>Assignment 4</b>
<b>Deadline:</b> Upload (ZIP) in TUWEL until Monday, January 12 <sup>th</sup> , 2015, 23:55 Assignment Review: Wednesday, January 14 <sup>th</sup> , 2015	<b>25 Points</b>


## Code Generation and Model Analysis





The goal of this assignment is to develop code generators for HTML and Alloy<sup>1</sup> code from models conforming to the *Entity Modeling Language (EML)* and the *Forms Modeling Language (FML)*, by using Xtend.

Therefore, in Part A, you have to develop an **HTML code generator** for FML and EML. In Part B, you have to implement an **Alloy code generator** for EML and define additional constraints to ensure the validity of EML models.

Alloy will be covered by lecture *M10 Model Analysis 1/2* on December 15<sup>th</sup>, 2014.

### Assignment resources

 ME\_WS14\_Lab4\_Resources.zip

-  at.ac.tuwien.big.forms (Modeling project with solution of Assignment 1)
-  at.ac.tuwien.big.forms.htmlgen (Xtext project for HTML generator)
-  at.ac.tuwien.big.forms.alloygen (Xtext project for Alloy generator)
-  ME\_WS14\_Lab4.pdf (this document)

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. Furthermore, for Part B you have to install an additional tool for this assignment called Alloy. How to install Alloy will be explained in Part B of this specification.

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

## Part A: HTML Code Generation

In this part you have to implement a model-to-text transformation to generate valid and interactive HTML documents that represent the content specified in FML models and EML models. With the generated HTML documents, it should be possible to navigate from form to form and enter information into input fields and selection fields of the forms. Furthermore mandatory fields should be enforced in the HTML forms, regular expressions should be verified, and conditions should be realized by making elements of the form visible or invisible interactively based to the selected option.

However, for switching between forms, enforcing mandatory fields, verifying regular expression, and realizing conditions, a JavaScript file is provided in the HTML generator project (at.ac.tuwien.big.forms.htmlgen/assets/form.js).

<sup>1</sup> <http://alloy.mit.edu/alloy/>

The HTML code you have to generate has to follow the example code that is depicted in Figure 1. In the head of the HTML file, the JavaScript libraries, as well as the CSS file have to be specified (lines 7-9) in order to use them in the HTML document. Furthermore, in the head of the HTML file, you also have to generate JavaScript code that registers the HTML elements generated for the forms defined in an FML model (lines 10-18). In the body of the HTML file, you have to generate these HTML elements according to the forms defined in an FML model (lines 20-24).

**Figure 1: Excerpt of the generated HTML Code**

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <title>Form</title>
5     <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
6     <meta http-equiv="content-type" content="text/html; charset=UTF-8"/>
7     <link rel="stylesheet" type="text/css" href="../assets/form.css"/>
8     <script src="../assets/jquery-1.10.2.min.js" type="text/javascript"></script>
9     <script src="../assets/form.js" type="text/javascript"></script>
10    <script type="text/javascript">
11      $(document).ready(
12        function(){
13          ...
14          //register HTML elements here
15          ...
16          form.init();
17        });
18    </script>
19  </head>
20  <body>
21    ...
22    //add HTML elements here
23    ...
24  </body>
25 </html>
```

## Registration of HTML elements

In order to define which HTML form serves as welcome form, which text fields have regular expressions, etc., those elements have to be registered using JavaScript code. Therefore, you have to generate JavaScript code that calls the methods defined in the provided JavaScript file `form.js` as described in the following.

### Welcome form

The welcome form has to be registered with its title.

```
form.addWelcomeForm('<Form.title>');
```

Example: `form.addWelcomeForm('PublicationForm');`

### Regular expression defining text field value format

The regular expression defining the allowed value format of a text field has to be registered with the `elementID` and `format` value of the text field.

```
form.addRegularExpression('<TextField.elementID>', '<TextField.format>');
```

Example: `form.addRegularExpression('01', '^[a-zA-Z -]+$');`

### Relationship page elements

Relationship page elements have to be registered with their container page, elementID, editing form, type ('table' or 'list'), and relationship multiplicities.

```
form.addRelationshipPageElement ('<ContainerPage.title>',  
    '<RelationshipPageElement.elementID>',  
    '<RelationshipPageElement.editingForm.title>',  
    '<type>',  
    '<RelationshipPageElement.relationship.lowerBound>',  
    '<RelationshipPageElement.relationship.upperBound>');
```

Example: `form.addRelationshipPageElement('Authors', '11', 'PersonForm', 'table', '1', '-1');`

### Composite conditions

A composite condition has to be registered with its conditionID, the conditionID of its container composite condition (or null if the composite condition is not contained by another composite condition), and its type.

```
form.addCompositeCondition('<CompositeCondition.conditionID>',  
    '<CompositeCondition.ContainerCompositeCondition.conditionID>',  
    '<CompositeCondition.type>');
```

Example: `form.addCompositeCondition('1', null, 'Or');`

### Attribute value condition

An attribute value condition has to be registered with its conditionID, the conditionID of its container composite condition (or null if the attribute value condition is not contained by a composite condition), the element for which the attribute value condition is defined (the title of the page containing the attribute value condition or alternatively the elementID of the page element containing the attribute value condition), and the value and type of the attribute value condition.

```
form.addAttributeValueCondition('<AttributeValueCondition.conditionID>',  
    '<AttributeValueCondition.ContainerCompositeCondition.conditionID>',  
    ('<AttributeValueCondition.ContainerPage.title>'  
        XOR '<AttributeValueCondition.ContainerPageElement.elementID>'),  
    '<AttributeValueCondition.value>',  
    '<AttributeValueCondition.type>');
```

Example: `form.addAttributeValueCondition('4', null, 'Journal', 'JA', 'Show');`

## HTML elements

For the forms defined in an FML model, you have to generate suitable HTML code as described in the following based on examples. For a better understanding, some parts of the example HTML code are highlighted in **light green**. This highlighted code indicates that it has been generated from the elements of an FML model.

### Form

For a form, a `<div>` element is created with the class "form" and an id corresponding to the name of the form. Furthermore, an `<h1>` and an `<h2>` heading are generated for the title and the description of the form, respectively. These headings are contained in a `<form>` element.

Example
<pre>&lt;div class="form" id="PublicationForm"&gt;   &lt;form action="#" class="register"&gt;     &lt;h1&gt;Publication&lt;/h1&gt;     &lt;h2&gt;Form for scientific publications&lt;/h2&gt;     ...     //add pages here     ...   &lt;/form&gt; &lt;/div&gt;</pre>

### Page

For a page, a `<div>` element is created with the class "page" and an id corresponding to the title of the page. Furthermore, a `<fieldset>` with contained `<h3>` heading defining the page title are created.

Example
<pre>&lt;div class="page" id="PublicationDetails"&gt;   &lt;fieldset class="row1"&gt;     &lt;h3&gt;PublicationDetails&lt;/h3&gt;     ...     //add pageElements here     ...   &lt;/fieldset&gt; &lt;/div&gt;</pre>

### Attribute page element

For attribute page elements, suitable HTML elements are generated, namely `<input>` elements for text fields, date selection fields, and time selection fields; `<textarea>` elements for text areas; and `<selection>` elements for selection fields. The id of these HTML elements correspond to the elementID of the respective attribute page element. Furthermore, for each of these attribute page elements, a `<label>` element is created displaying the label of the attribute page element.

The `<selection>` element created for a selection field has its name set to the name of the attribute referenced by the selection field. If this attribute has an enumeration set as type, for each of the literals of this enumeration, an `<option>` element is created. Thereby, the value and the displayed text of this `<option>` element correspond to the value and the name of the literal, respectively. If the attribute has the type Boolean, two `<option>` elements with the value and displayed text 'Yes' and 'No' are created.

For each column of a table, a <th> element with the column's label is created. These <th> elements are contained by one <tr> element, whose id is set to the elementID of the table with appended '\_header' (e.g., 11\_header).

An attribute page element can be mandatory. This information has to be provided in the generated HTML code. Therefore, the label created for an attribute page element must be marked with a star at the end (e.g., Title<span>\*</span>). Additionally, the generated <input>, <textarea>, or <select> element has to define the class "mandatory". The columns created for a table don't need this mandatory check because they are not used as an input field and only display entered data.

#### Examples

```
//mandatory TextField
<p>
  <label for="01">Title<span>*</span></label>
  <input type="text" id="01" class="mandatory"/>
</p>

//TextField
<p>
  <label for="02">Keywords</label>
  <input type="text" id="02" />
</p>

//TextArea
<p>
  <label for="03">Abstract</label>
  <textarea id="03"></textarea>
</p>

//mandatory SelectionField
<p>
  <label for="08" >Publication type<span>*</span></label>
  <select id="08" name="type" class="mandatory">
    <option value="default"> </option>
    <option value="JA">Journal Article</option>
    <option value="BC">Book Chapter</option>
    <option value="CP">Conference Paper</option>
    <option value="WP">Workshop Paper</option>
  </select>
</p>

//DateSelectionField
<p>
  <label for="38">From date</label>
  <input type="date" id="38"/>
</p>

//TimeSelectionField
<p>
  <label for="38">From date</label>
  <input type="time" id="38"/>
</p>

//Column
<tr id=11_header>
  <th>First name</th>
  <th>Last name</th>
  <th>E-Mail</th>
</tr>
```

## Relationship page element

For lists and tables, <div> elements with the class "list" and "table", respectively, as well as with an id corresponding to their elementIDs are created. Furthermore, <fieldset> and <legend> elements are created. The text contained by the <legend> element corresponds to the label of the list or table with ' List' or ' Table' appended, respectively (e.g., Authors Table).

Example
<pre>//List &lt;div class="list" id="16"&gt;   &lt;fieldset class="row1"&gt;     &lt;legend class="legend"&gt;Journal List&lt;/legend&gt;     &lt;ul&gt;&lt;/ul&gt;   &lt;/fieldset&gt; &lt;/div&gt;  //Table &lt;div class="table" id="11"&gt;   &lt;fieldset class="row1"&gt;     &lt;legend class="legend"&gt;Authors Table&lt;/legend&gt;     &lt;table&gt;       ...       //add columns here       ...     &lt;/table&gt;   &lt;/fieldset&gt; &lt;/div&gt;</pre>

## Part B: Alloy Code Generation

In this part you have to develop a second model-to-text transformation to generate valid Alloy code that represent the content specified in the given EML models. The given EML models can be found in the input folder of the Alloy code generator project at.ac.tuwien.big.forms.alloygen. Figure 2 and Figure 3 depict these given EML models graphically.

Table 1 provides the mappings between EML modeling concepts and Alloy concepts that have to be used for developing the Alloy code generator.

**Table 1: Mappings between EML concepts and Alloy concepts**

EML concept	Alloy concept
Entity	Signature
Enumeration and literals	Singleton Subsig (keyword enum) Literals of enumerations shall be defined with their names (i.e., their values do not have to be considered).
Attribute with primitive type (String, Integer, etc.)	All attributes with primitive types are mapped to Alloy Int atoms.
Attribute with enumeration type	An attribute with an enumeration type has to be mapped to an Alloy atom with a relation to the singleton subsig created for the respective enumeration. .
Mandatory attributes	The atoms created for the mandatory

	attributes must be equipped with the appropriate multiplicity keyword. This is also necessary for non-mandatory attributes.
Relationship	<b>Relation</b> For the multiplicities 0..1, 1..1, 0..-1, 1..-1 the multiplicity keywords provided by Alloy have to be used. For other multiplicities use the set keyword and define a fact that constraints the multiplicities appropriately. (The constraint is described in detail below.)

For bidirectional references and relationships with multiplicities other than 0..1, 1..1, 0..-1, and 1..-1, additional constraints have to be generated by your Alloy code generator. Furthermore, for the example EML model `publicationEntityModel.forms` one additional constraint has to be defined for ensuring the validity of instances of the defined entities.

### Bidirectional references

Bidirectional references between two entities are created with two distinct relationships pointing in opposite direction, where each of the relationships has to maintain a pointer to the corresponding opposite relationship.

Example: In the example EML model `publicationEntityModel.forms`, the relationship 'authors' of the entity 'Publication' has set as its opposite relationship the relationship 'publications' of the entity 'Person'. Thus, the relationship 'publications' must have set as its opposite relationship the relationship 'authors'.

### Relationship multiplicities

Special care has to be taken for relationships with multiplicities other than 0..1, 1..1, 0..-1, and 1..-1. For relationships with such multiplicities, additional constraints have to be generated that ensure the validity of relationship instances.

Example: In the example EML model `courseEntityModel.forms`, the relationship 'isEnrolledBy' of the entity 'Course' defines a multiplicity 10..-1. For this multiplicity, an additional constraint has to be generated.

### Publication EML model

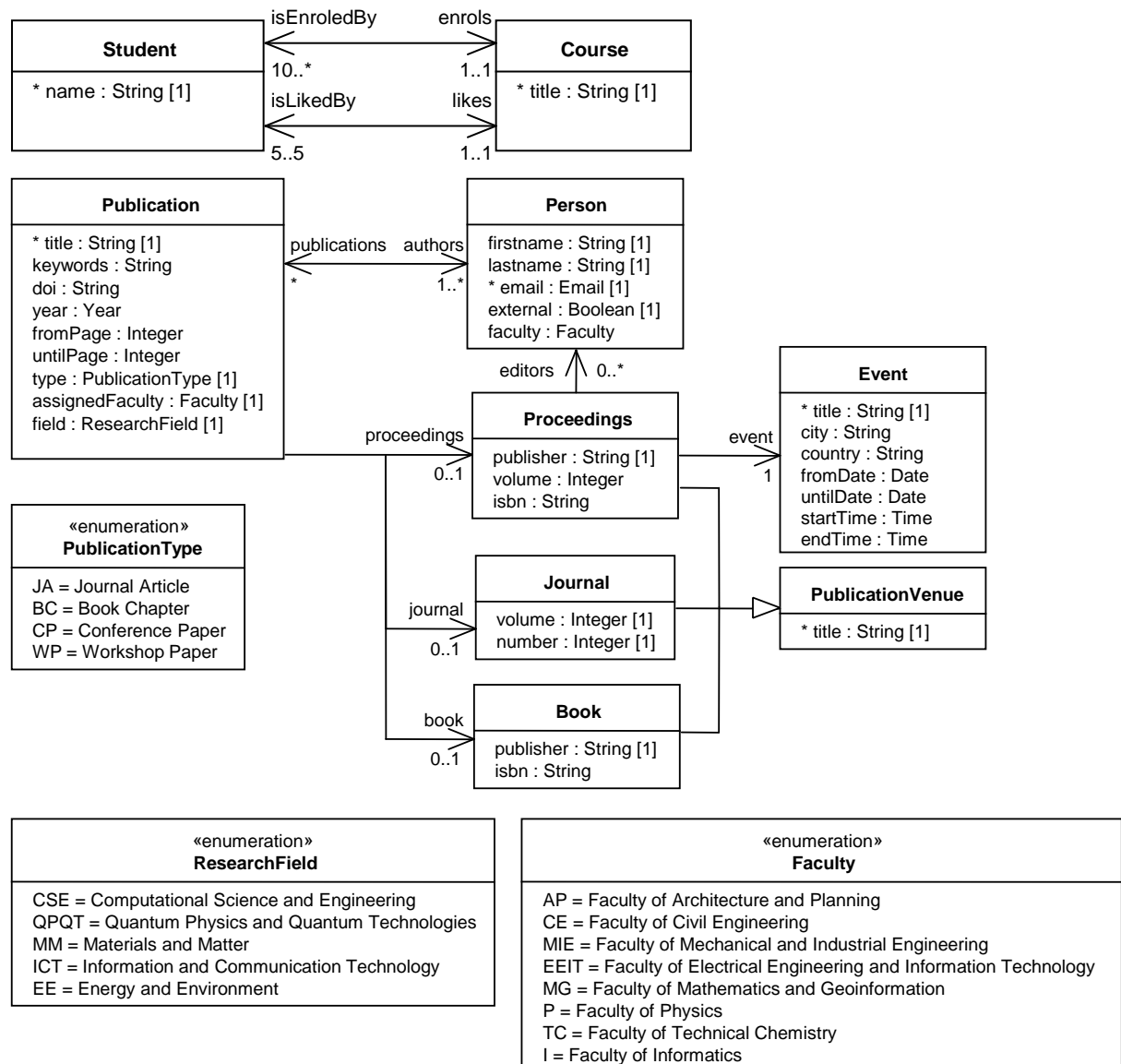
For the example EML model `publicationEntityModel.forms` (provided in the folder `input` of the project `at.ac.tuwien.big.forms.alloygen`), you have to define the following additional constraint. This constraint does not have to be generated by your code generator. Instead, define the constraint in the file `publicationTypeConstraint.als` (provided in the folder `constraint` of the project `at.ac.tuwien.big.forms.alloygen`).

An instance of the entity 'Publication' can have set as type one of the enumeration literals 'JA', 'BC', 'WP', and 'CP'. Depending on the set type, a 'Publication' instance has to have a relation to an instances of the entity 'Journal' (in the case of `type='JA'`) XOR an instance

of the entity 'Book' (in the case of type='BC') XOR an instance of the entity 'Proceedings' (in the case of type='WP' or type='CP').

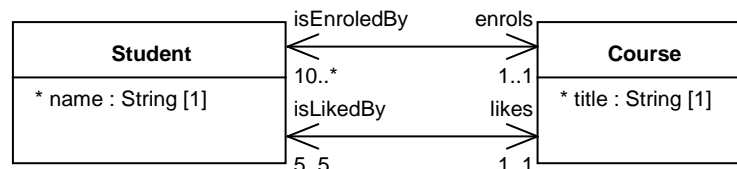
Example: A 'Publication' instance, which has set the type to 'JA' has to have a relation 'journal' to a 'Journal' instance. However, it may not have a 'proceedings' relation to a 'Proceedings' instance or a 'book' relation to a 'Book' instance.

**Figure 2: Graphical representation of the given EML model publicationEntityModel.forms**



Note: Because a keyword cannot be defined as identifier in Alloy, the attribute 'Publication.abstract' was removed.

**Figure 3: Graphical representation of the given EML model courseEntityModel.forms**





## 1. Setting up your workspace

As mentioned in the beginning, both code generators are based on the metamodel defined in Assignment 1. We provide the solution of this assignment as well as two skeleton projects, which have to be used for developing the code generators.

### 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\_WS14\_Lab4\_Resources.zip* and import the projects called *at.ac.tuwien.big.forms*, *at.ac.tuwien.big.forms.htmlgen*, *at.ac.tuwien.big.forms.alloygen*.

After you have successfully imported the three 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 two projects serve as base for your code generators and should have the Xtext Nature applied.

### Register form metamodel

Before you start implementing your code generators, you have to register the forms metamodel in your Eclipse instance, so that the Xtend editor with which you develop your generators knows which metamodel you want to use. You can register the provided metamodel by clicking right on *at.ac.tuwien.big.forms/model/forms.ecore* → *EPackages registration* → *Register EPackages into repository*.

## 2. Developing your code generators

Define the HTML code generator for FML and EML in the *Form2HTMLGenerator.xtend* file located in the *src* folder of the *at.ac.tuwien.big.forms.htmlgen* project and the Alloy code generator for EML in the *Form2AlloyGenerator.xtend* file located in the *src* folder of the *at.ac.tuwien.big.forms.alloygen* project.

Note: In this assignment you have to implement your solution with Xtend. Make sure that your Eclipse instance is setup as described in the Eclipse Setup Guide provided in TUWEL. Documentation about how to use Xtend can be found in the Xtend documentation<sup>2</sup> or in the lecture slides.

## 3. Generate your code

After you have finished developing your code generators (*Form2HTMLGenerator.xtend* and *Form2AlloyGenerator.xtend*) you can run them for generating code for the provided FML and EML models. The models are provided in the *input* folders of the code generator projects *at.ac.tuwien.big.forms.htmlgen* and *at.ac.tuwien.big.forms.alloygen.input*. For generating the code you have to run the workflow files located next to the *.xtend* files, which orchestrate the code generation (*Form2HTMLGenerator.mwe2* and *Form2AlloyGenerator.mwe2*).

To run the generation, select the respective workflow file and select *Run As* → *MWE2 Workflow*. This will start the code generation. Check the output in the Console to see whether the generation was successful. Please note that after each change in your generator, you have to re-run the workflow to update the generated code. If you start the code generator for the first time the following message may appear:

---

<sup>2</sup> Xtend documentation: <http://www.eclipse.org/xtend/documentation.html>

**\*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.

After the generation of your code was successful you will find your generated code files in the output folders of the generator projects (at.ac.tuwien.big.forms.htmlgen/output and at.ac.tuwien.big.forms.alloygen/output). The HTML generator generates the file output.html for the input file formModel.forms. The Alloy generator generates the files courseEntityModel.als and publicationEntityModel.als for the input files courseEntityModel.forms and publicationEntityModel.forms, respectively.

## 4. Testing your generated code

### Test your HTML code

**Markup validation:** You have to check the markup validity<sup>3</sup> of your generated output.html file using the W3C Markup Validation Service. Visit the site <http://validator.w3.org/> -> *Validate by Direct Input*, copy your generated HTML code into the text area and click *Check*. The following message should appear:

This document was successfully checked as HTML5!

You have to make sure that the validation of your HTML code is successful. However, you can ignore displayed warnings.

**Form testing:** Open your generated output.html file by right-clicking on it and select *Open with -> WebBrowser*. The following manual tests must work without any unexpected behavior.

#### Manual tests:

---

**Condition Test 1** After selecting a 'Publication type' at the welcome form 'PublicationForm' the correct elements have to be displayed. E.g., for 'Journal Article' this includes the header 'Journal' and legend 'Journal List'.

---

**Condition Test 2** In the form 'PersonForm', the selection field 'Faculty' is only visible if either no value or the value 'No' is selected for the selection field 'Faculty-external'. Otherwise, if the value 'Yes' is selected, the selection field 'Faculty' becomes invisible. When you select 'No', the selection field 'Faculty' becomes visible again.  
(For switching to the 'PersonForm', hit the button 'Add' located in the page 'Authors' of the welcome form 'PublicationForm'.)

---

**Regular Expression** Make sure that the value of input fields generated for text fields with regular expression are accordingly checked. For instance, if you enter the value 'asdf' in the input field 'Year' of the welcome form 'PublicationForm' and hit the button 'Save', the label of this input field becomes red and the publication is not saved.

---

**Mandatory Field** A mandatory field has to be marked with a star after its label (e.g. 'Title\*' of the welcome form 'PublicationForm'). Furthermore, if no value is provided

---

<sup>3</sup> Markup validity: [http://validator.w3.org/docs/help.html#validation\\_basics](http://validator.w3.org/docs/help.html#validation_basics)

---

ed for a mandatory field and you hit the button 'Save', the label of this field becomes red and the entity is not saved.

---

**Multiplicities** For lists and tables, the number of entered entities is checked against the multiplicity defined by the respective relationship. For instance, if you do not enter an author of a publication on the form 'PublicationForm' and hit 'Save', the message 'Authors table must have at least one element' has to be displayed. Similarly, when you have selected the publication type 'JA' and entered one journal, you should not be able to enter another journal.

---

Furthermore, we provide the HTML code expected to be generated from the example FML model `formModel.forms` (provided in `at.ac.tuwien.big.forms.htmlgen/input`) in the HTML file `output.html` located in the folder `expected_output` of the project `at.ac.tuwien.big.forms.htmlgen`. Compare the behavior of this expected output HTML file with the behavior of the output HTML file generated by your HTML code generator.

Note: Your code generator has to be able to generate suitable HTML and JavaScript code for any valid FML model.

## Test your Alloy code

In order to test your Alloy code you have to install alloy4.2<sup>4</sup>. After you finished the installation open your generated alloy files (`publicationEntityModel.als` and `courseEntityModel.als`). Execute the predicates provided in the Alloy file `tests.als` (folder `tests` of project `at.ac.tuwien.big.forms.alloygen`). Therefore, copy the predicates one by one at the end of your generated code and hit the 'Execute' button. Check, whether you obtain the expected result as described in the following. The Alloy console on the right side shows whether an instance was found or not.

### Publication EML model tests (`publicationEntityModel.als`):

---

**Valid Test 1** An instance should be found.

**Invalid Test 1** No instance should be found, because a 'Publication' instance cannot be of type 'BC' and have a 'journal' relation to a 'Journal' instance.

Note: Do not forget to add the additional constraint that you had to develop for the Publication EML model in order to ensure the correct relations between 'Publication' instances and 'PublicationVenue' instances depending on the publication type.

### Course EML model tests (`courseEntityModel.als`):

---

**Valid Test 2** An instance should be found.

**Valid Test 3** Only the empty set is found as valid instance.

Note: In order to find out whether the found instance is an empty set or not, click 'Show' in the menu bar and switch to 'Txt'. For the signatures 'Course' and 'Student', no instances should be shown.

---

<sup>4</sup> Alloy download: <http://alloy.mit.edu/alloy/download.html>

## Submission & Assignment Review

### Upload the following components in TUWEL:

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

Code Generator Projects
<code>at.ac.tuwien.big.forms.htmlgen</code> <code>at.ac.tuwien.big.forms.alloygen</code>

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

**Make sure to include all resources that are necessary for the code generation!**

If you are unsure whether you have provided all required resources, test your exported projects by importing them into a new workspace, running the code generators, and testing the generated code again.

**Also, do not forget to save the additional Alloy constraint for the Publication EML model in the file `at.ac.tuwien.big.forms.alloygen/constraint/publicationTypeConstraint.als`**

### Review: 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.