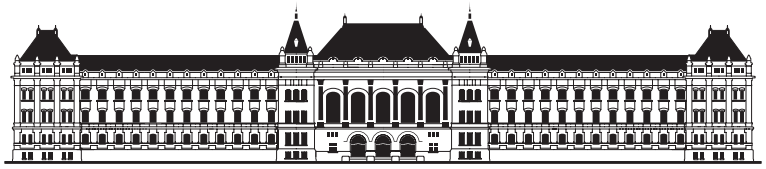

Budapest University of Technology and Economics
Department of Measurement and Information Systems
Fault Tolerant Systems Research Group

Design for Dependability Laboratory Exercises
Autumn Semester 2014/2015

Automatic Analysis of Domain-Specific Languages

Syllabus
v1.0

Author: Csaba Debreceni
(debreceni@mit.bme.hu)

September 26, 2014

Contents

1 Introduction

This document is a syllabus for the *Automatic Analysis of Domain-Specific Languages* session of the *Design for Dependability Laboratory Exercises* course. The purpose of this session is tto demonstrate a method how to validate our *Domain-Specific Langauges* (created by EMF) using the *EMF Validation Framework* then how to generate source code using the *Acceleo* engine and show the usage of *Dynamic EMF API*.

1.1 Background

The current section highlights our running example that will be used in the laboratory session.

1.1.1 Cyber-physical Systems

Cyber-physical systems (CPS) are on one hand close to *embedded systems* as they are also built from sensors, controllers and actuators, where the sensors gather heterogeneous information from the environment, the controllers observe the gathered information and order the actuator to modify the environment according to the observed information. On the other hand, CPS systems are aiming to harvest the benefits of elastic cloud based resources to provide even more complex automatiation services.

The metamodel of a simplified *Cyber-physical System* is depicted on Figure 1. In our terminology, the sensors, controllers and actuators are TASK instances and the nodes in the cloud infrastructure are COMPUTER instances. Each TASK can be allocated to one dedicated COMPUTER (*allocatedTo* reference) and uses some resources of the node that are defined as slots (*reqSlot*). Additionally, different type of tasks are on different level of severity (*low*, *medium*, *high*, *critical*). The COMPUTER class has two attributes where the attribute *availableSlots* shows the currently unused slots while the *defaultSlots* represents the maxiumum number of resources the node has.

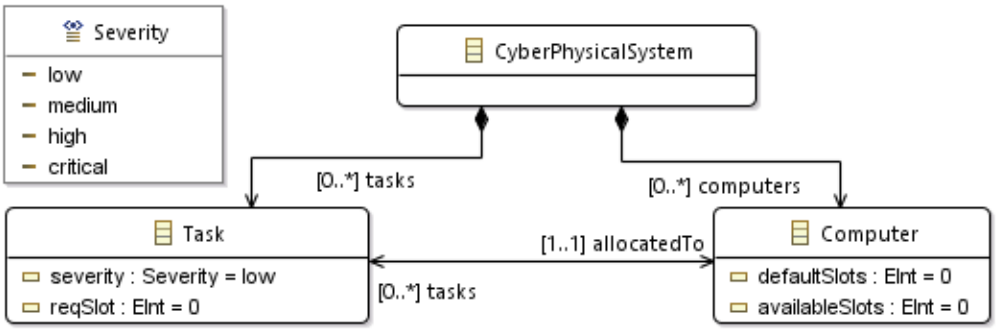

Figure 1: Simplified CyberPhysicalSystem metamodel

1.1.2 Laboratory

During this session, your task will be to extend the metamodel based on some novel requirements, define additional validation rules to the extended metamodel using the *EMF Validation Framework* and finally, create an *Acceleo* code generator to generate a simple *Alloy* code.

2 Technologies

2.1 EMF Validation Framework

The *Eclipse Modeling Framework (EMF)*[7] is considered as the *de facto* industry standard for modeling. During the past years, *EMF* was extended with additional services such as *EMF Validation Framework*[9]. The purpose of this framework is to provide *domain-specific language (DSL)* independent model validation over *EMF*.

The *validation framework* supports the definition of *batch* and *live* validation rules. In the case of using *batch* rules, users have to explicit execute the validation while the *live* rules are triggered by the *change notification* service provided by *EMF*.

The basic overview of *validation framework* is described in [10]. An advanced validation tutorial is explained in [3].

In this session, we will use a pre-configured validation plugin. The main task is to create additional constraints by extending the ABSTRACTMODELCONSTRAINT class and inserting a reference for the created classes into the plugin xml under the *org.eclipse.emf.validation.constraintProviders* extension point. The validation can be executed by the *Validate* button in the context menu of any EMF tree editor (generated editor, Simple Ecore Reflective Editor etc.).

2.2 Dynamic EMF API

The *Object Management Group* defines a 4-layered architecture ($M3$ - meta-metamodel, $M2$ - metamodel, $M1$ - instance model, $M0$ - reality) standard for model-driven engineering that separates the different conceptual level for defining a model. The *EMF* framework defined its own $M3$ variant called *Ecore*. Its simplified structure is depicted on fig. 2.

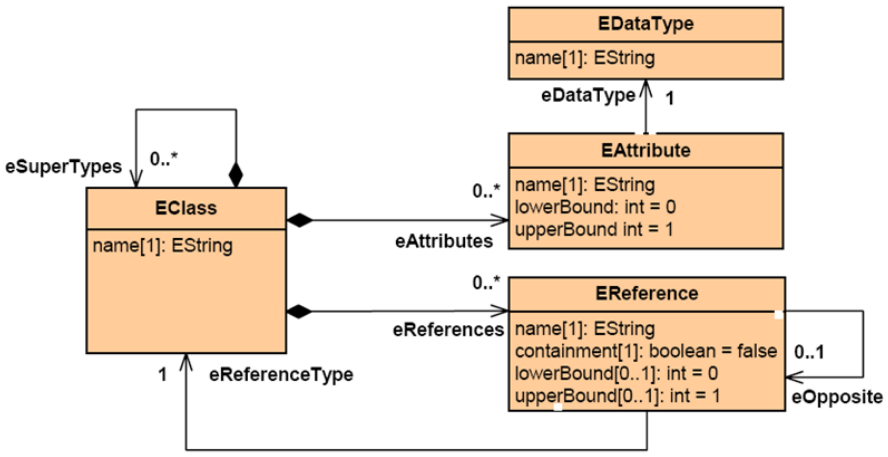

Figure 2: Ecore metamodel - OMG $M3$ standard

The main class is the ECLASS which can contain multiple EATTRIBUTEs and EREFERENCEs. The type of EREFERENCE is an ECLASS (*eReferenceType*) while the type of EATTRIBUTE is an EDATATYPE (*eDataType*) ie. EINT, ESTRING. ECLASS can also have multiple super types (*eSuperTypes*). For more details about the *Ecore* meta-metamodel, check out [8].

The *Dynamic EMF API* provides the ability to create in-memory metamodels and in-memory instance models for them. An advanced tutorial can be found in [6]. Unfortunately, we will not use these features in this session, but the API also allows to access the metamodel of an existing instance model. With this feature, we can traverse the entire metamodel. The following example shows how to get all the *EClasses* of the metamodel from an instance.

```
public Collection<EClass> getEClasses(CyberPhysicalSystem cps) {
  List<EClass> ret = new ArrayList();
  for(EClassifier ec : cps.eClass().getEPackage().getEClassifiers()) {
        if(ec instanceof EClass)
          ret.add((EClass) ec);
  }
  return ret;
}
```

Listing 1: Getting all the EClasses in Java

2.3 ACCELEO

ACCELEO[1] is a pragmatic implementation of the Object Management Group (OMG) Model to Text Language (MTL) standard. It is a template-based code generator with an advanced IDE. The framework defines its own language. The following simple example shows the syntax:

```
[comment encoding = UTF-8 /]
[module generate('/hu.bme.mit.cps/model/cps.ecore')/]

[template public generate(cps : CyberPhysicalSystem)]
  [comment @main /]
  [file ('cps.txt', false, 'UTF-8')]

  Hello ACCELEO World
  [for (task : Task | cps.tasks)]
    [if (task.severity = Severity.low)]
      [task.eClass.name] low
    [elseif (task.severity = Severity.medium)]
      [task.eClass.name] medium
    [elseif (task.severity = Severity.high)]
      [task.eClass.name] high
    [else (task.severity = Severity.critical)]
      [task.eClass.name] critical
    [/if]
  [/for]
  [/file]
[/template]
```

Listing 2: ACCELEO example

The generator files consist of modules and templates. The ACCELEO can be interpreted as an OO language where the modules are the classes and the templates are the methods inside a module. The first line defines the character coding of the generated file, then at least one metamodel path has to be connected to this module as it is in the second line. The main entry point of the generator will be the template with *@main* comment. The language uses only the *if-else*, *let*, *for* structures because it supports to define OCL queries. The following query returns all the *EClass*es of the metamodel.

```
[query public getEClasses(cps : CyberPhysicalSystem) : Set(EClass) =
    cps.eClass().ePackage.eClassifiers->filter(EClass)/]
```

Listing 3: ACCELEO query example

The ”Getting Started” documentation can be found at [2]. Use it wisely for your preparation.

2.4 Alloy

Alloy is a language for describing structures and defining constraints for them. It is also a tool for generating instances that satisfies the constraints on the given structure. The following example shows the main elements of the language that will be use in the current lab session.

```
enum MyEnum {
        literal1, literal2
}

abstract sig AbstractMyClass {
        attr1 : one MyEnum,
        attr2 : one Int,
        ref1  : many AbstractMyClass
}

sig MyClass extends AbstractMyClass {}

run {} for 2 MyClass
```

Listing 4: Alloy language example

Enumerations can be defined with the *enum* keyword and classes (signatures) with the *sig* keyword. After every properties of a structure (enum or class) put a comma (“,”) except the last one. To execute traversals of the search space, we can use the *run* command where the number of required instances can also be defined. For further information, check out [5]. Finally, the alloy analyzer can be downloaded from [4].

3 List of Questions

1. What kind of validation modes are supported in EMF Validation Framework? What is the difference between them?
2. What is the list of supported languages in EMF Validation Framework for defining constraints?
3. What is a module in ACCELEO? What is a template in ACCELEO?
4. What kind of control flow statements are supported in ACCELEO?
5. What is an ACCELEO UI Launcher project good for?
6. Draw the most important parts of the Ecore metamodel!
7. What is an EClassifier in Ecore? Give some known sub interfaces!
8. What is an EStructuralFeature in Ecore? Give some known sub interfaces!

4 Tasks

4.1 Extend the existing DSL

Extend the existing DSL with the following elements: the *alarm*, *smoke-detector* and *controller* elements are inherited from task while the *server* and *mainframe* elements are inherited from node. Additionally, the controller has a reference for at least two other tasks. Moreover, provide that neither TASK nor COMPUTER classes can be initiated.

4.2 Create an instance model for the DSL

In the runtime eclipse, initiate an instance model for your extended DSL. It should contains at least 10 model elements.

4.3 Provide additional validation rules

Create at least 3 validation rules from the following list (extra points for all):

- Models should not contain abstract elements.
- Every COMPUTER instance should contain only one *critical* task.
- Every TASK instance should have the correct severity level.
- Every CONTROLLER instance should control tasks with different type.
- The *availableSlots* attribute of every COMPUTER should be equal with *defaultSlots - sum(tasks.reqSlots)* where the tasks are the allocated TASK instances on the actual COMPUTER.

Validate your instance model (4.2).

4.4 Create an ACCELEO code generator

This task has more sub-tasks to provide some refinement steps but it is possible to implement the whole generator in a single step. The requirements for generator are the following: (i) generate enums and class signatures with properties and references, (ii) generate also the run command with the proper number of the existing classes in your model. A simple output is depicted on listing 5.

```
enum MyEnum {
        literal1, literal2
}

abstract sig AbstractMyClass {
        attr1 : one MyEnum,
        attr2 : one Int,
        ref1  : many AbstractMyClass
}

sig MyClass extends AbstractMyClass {}

run {} for 2 MyClass
```

Listing 5: Simple example output

The generator has to be well-commented and well-formatted. It will be awarded with some extra points if you use multiple *templates* and/or *queries*.

Sub-tasks:

- Generate enumerations with their literals
- Generate classes (abstract/not abstract, define ancestor)
- Generate properties, references (EStructuralFeature) with respect to types (EInt -> Int)
- Generate run command according to your instance model

Additional task for extra point:

- Order the generation as follows: (1) enumerations, (2) classes without ancestors, (3) classes with previously defined ancestors

4.5 Create an ACCELEO UI Launcher

Generate Alloy code from the previously created instance model in a runtime eclipse.

4.6 Alloy Analyzer

Download the alloy analyzer and try it with your generated *"als"* code.

References

[1] ACCELEO. https://www.eclipse.org/acceleo/.

[2] ACCELEO Getting Started. http://wiki.eclipse.org/Acceleo/Getting_Started.

[3] Advanced EMF Validation Tutorial. http://publib.boulder.ibm.com/infocenter/rsahelp/v7r0m0/index.jsp?topic=/org.eclipse.emf.validation.doc/references/extension-points/org_eclipse_emf_validation_constraintProviders.html.

[4] Alloy analyzer. http://alloy.mit.edu/alloy/download.html.

[5] Alloy language. http://alloy.mit.edu/alloy/documentation/book-chapters/alloy-language-reference.pdf.

[6] Dynamic EMF tutorial. http://www.ibm.com/developerworks/library/os-eclipse-dynamicemf/.

[7] Eclipse. http://www.eclipse.org/.

[8] Ecore package details. http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html#details.

[9] EMF Validation Framework. http://www.eclipse.org/modeling/emf/?project=validation.

[10] EMF Validation Overview. http://help.eclipse.org/kepler/index.jsp?topic=%2Forg.eclipse.emf.doc%2Freferences%2Foverview%2FEMF.Validation.html&cp=17_0_2.