Final thesis

Design and Implementation of the ModelicaML Code Generator Using Acceleo 3.X

by

Ibrahim Bumin Kara

LIU-IDA/LITH-EX-A--15/012—SE

2015-05-07
Final Thesis

Design and Implementation of the ModelicaML Code Generator Using Acceleo 3.X

by

Ibrahim Bumin Kara

LIU-IDA/ LITH-EX-A--15/012—SE

2015-05-07

Supervisor: Olena Rogovchenko
Examiner: Peter Fritzson
Abstract

In today’s fast paced changing technology world, it is essential to adapt the new technologies and standards in order to stay compliant with the market. Software development is one of those key areas that have been growing highly fast and thus the need for staying up to date needs to be acknowledged by everyone in the area. The technology focused in this project is called ModelicaML[1] which is an integrated system modeling environment based on standardized languages such as UML[2] and Modelica[3]. Modelica Modeling Language (ModelicaML) is a graphical modeling language used for the description of time-continuous and time-discrete/event-based system dynamics. Currently, a ModelicaML code generator – that generates Modelica code from UML models – is implemented using the old Acceleo framework[4]. Within this thesis, the ModelicaML code generator was re-factored and implemented by using the latest Acceleo version 3.X[5]. The differences between the old and new Acceleo framework was also given an evaluation in the study in order to provide a hint of its future use for further improvements of the code generator.
Contents

Chapter 1 Introduction ........................................................................................................... 9
  1.1 Background ..................................................................................................................... 9
  1.2 Purpose and Objective ............................................................................................... 9
  1.3 Limitations .................................................................................................................. 10
  1.4 Intended Audience ..................................................................................................... 10
  1.5 Thesis Overview ......................................................................................................... 10

Chapter 2 Technologies ....................................................................................................... 11
  2.1 OpenModelica ............................................................................................................. 11
    2.1.1 Modelica ................................................................................................................ 11
    2.1.2 ModelicaML – A UML Profile for Modelica ......................................................... 11
  2.2 UML ............................................................................................................................ 12
  2.3 Papyrus ....................................................................................................................... 12
  2.4 Acceleo ........................................................................................................................ 12
    2.4.1 Acceleo 2.x ........................................................................................................... 12
    2.4.2 Acceleo 3.x ........................................................................................................... 13
  2.5 Object Management Group (OMG) .......................................................................... 13
  2.6 MOF Model to Text .................................................................................................. 14
  2.7 EMF ............................................................................................................................. 14

Chapter 3 Acceleo 2.x vs Acceleo 3.x .................................................................................. 15
  3.1 Acceleo 2.X ................................................................................................................ 15
    3.1.1 Module .................................................................................................................. 15
    3.1.2 Scripts .................................................................................................................... 17
    3.1.3 Execution Chain ................................................................................................... 17
    3.1.4 UML Model ......................................................................................................... 18
  3.2 Acceleo 3.x ................................................................................................................ 19
    3.2.1 Modules ................................................................................................................ 19
    3.2.2 Templates ............................................................................................................. 19
    3.2.3 Queries ................................................................................................................. 20
    3.2.4 Java Services ....................................................................................................... 21
    3.2.5 Basic Language Constructs ................................................................................ 22
      3.2.5.1 File Tags .......................................................................................................... 22
      3.2.5.2 For loops ........................................................................................................ 22
      3.2.5.3 If Conditions ................................................................................................. 23
      3.2.5.4 Let blocks ..................................................................................................... 23
      3.2.5.5 Comments ................................................................................................... 24
  3.3 Acceleo 3.x vs Acceleo 2.x ......................................................................................... 25

Chapter 4 Implementation ..................................................................................................... 27
  4.1 ModelicaML Code Generator based on Acceleo 2.X .................................................. 27
  4.2 Migration of the ModelicaML Code Generator from Acceleo 2 to Acceleo 3 ............... 29
    4.2.1 Translation ............................................................................................................ 29
      4.2.1.1 Translation of the Modules ............................................................................ 29
      4.2.1.2 Translation of the Java files .......................................................................... 30
      4.2.1.3 Translation of Properties File ...................................................................... 31
      4.2.1.4 Main Module ................................................................................................. 32
    4.2.2 Configuration ....................................................................................................... 34
    4.2.3 Testing / Debugging .............................................................................................. 35
      4.2.3.1 Debugging ..................................................................................................... 35
      4.2.3.2 Comparison of the generation results ........................................................... 36
    4.2.4 Creating the Plug-in ............................................................................................. 37
Chapter 5   Results & Discussion ................................................................. 39
  5.1 Results & Conclusion ........................................................................ 39
  5.2 Future Work ...................................................................................... 39
Appendix A ................................................................................................. 41
Acceleo 2.x to 3.x equivalences ................................................................. 41
  A.1 Miscellaneous ................................................................................ 41
  A.2 Eobject Services .............................................................................. 42
  A.3 Request Services ............................................................................ 42
  A.4 String Services .............................................................................. 43
  A.5 System Services ............................................................................. 43
  A.6 Xpath Services ............................................................................... 44
  A.7 Enode Services ............................................................................... 44
  A.8 Properties Services ......................................................................... 48
Appendix B ................................................................................................. 48
Example code: ......................................................................................... 48
References ................................................................................................. 53
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.</td>
<td>Model Driven Development with Acceleo 2.X</td>
<td>15</td>
</tr>
<tr>
<td>Figure 2.</td>
<td>Example Module Acceleo 2.</td>
<td>16</td>
</tr>
<tr>
<td>Figure 3.</td>
<td>Generation Chain Example.</td>
<td>17</td>
</tr>
<tr>
<td>Figure 4.</td>
<td>UML model example.</td>
<td>18</td>
</tr>
<tr>
<td>Figure 5.</td>
<td>Acceleo 3 – Code Generation Process.</td>
<td>19</td>
</tr>
<tr>
<td>Figure 6.</td>
<td>Example Module.</td>
<td>20</td>
</tr>
<tr>
<td>Figure 7.</td>
<td>Query Example.</td>
<td>21</td>
</tr>
<tr>
<td>Figure 8.</td>
<td>Java Service Example.</td>
<td>21</td>
</tr>
<tr>
<td>Figure 9.</td>
<td>Query to call Java Service.</td>
<td>21</td>
</tr>
<tr>
<td>Figure 10.</td>
<td>File Tag Example.</td>
<td>22</td>
</tr>
<tr>
<td>Figure 11.</td>
<td>For loops example.</td>
<td>23</td>
</tr>
<tr>
<td>Figure 12.</td>
<td>Types of files (ModelicaML Code Generator - Accelo 2.X)</td>
<td>27</td>
</tr>
<tr>
<td>Figure 13.</td>
<td>Example of Queries calling Java Services.</td>
<td>31</td>
</tr>
<tr>
<td>Figure 14.</td>
<td>Properties file.</td>
<td>32</td>
</tr>
<tr>
<td>Figure 15.</td>
<td>Main Module.</td>
<td>33</td>
</tr>
<tr>
<td>Figure 16.</td>
<td>Main Java – Properties file addition.</td>
<td>33</td>
</tr>
<tr>
<td>Figure 17.</td>
<td>Main Java – Properties file addition.</td>
<td>34</td>
</tr>
<tr>
<td>Figure 18.</td>
<td>Acceleo Debugger.</td>
<td>36</td>
</tr>
<tr>
<td>Figure 19.</td>
<td>Main Java – Comparison of the Generation Results (Acceleo 2 vs 3)</td>
<td>37</td>
</tr>
<tr>
<td>Figure 20.</td>
<td>Main Java – ModelicaML Plugin.</td>
<td>37</td>
</tr>
</tbody>
</table>
Acknowledgements

I would like to express my gratitude to my examiner Peter Fritzson for giving me the opportunity to work on this project and introduced me to the Modelica world.

I also would like to thank my supervisors Olena Rogovchenko and Wladimir Schamai for their technical/administrative and emotional support on this journey. Their suggestions and encouragement has kept my motivation high until the end. The preparation of this project would not have been possible without their help.

Finally, my deepest gratitude goes to my family for their true love and support throughout my life.

Linköping, January 2014
Chapter 1

Introduction

This chapter provides an introduction and descriptions of related backgrounds to a master thesis project (30 credit points) final report examined at the Department of Computer and Information Science (IDA) at Linköping University. The thesis is conducted as the final part of a 2 years master program named as Software Engineering and Management.

1.1 Background

The most important key enablers for model-based systems engineering (MDE[7]) can be listed as a consistent set of languages, graphical notations and their semantics, modeling methods and model transformations. Modelica Modeling Language (ModelicaML [1]) is designed to meet these needs by providing the engineers with an integrated system modeling environment which is based on standardized languages such as UML [2] and Modelica [3].

ModelicaML is a graphical modeling language, designed to be used for the description of time-continuous and time-discrete/event-based system dynamics. It is defined as an extended subset of OMG Unified Modeling Language (UML [2]). ModelicaML is one of the ways for authoring Modelica models by generating executable Modelica code. The graphical modeling capabilities of Modelica is enhanced by providing additional diagrams such as UML diagrams for presenting the composition, connection, inheritance or behavior of classes to be used in graphical model definition and documentation. Furthermore, ModelicaML incorporates concepts for formalizing and evaluating system requirements during system simulations.

1.2 Purpose and Objective

The current ModelicaML code generator is implemented using a framework named as Acceleo 2.X [4]. This framework has a new version up, Acceleo 3.X [5] which has been developed using OMG MTL specification in order to keep up with the latest code generation trends. The Acceleo team was convinced that the future of code generation lied on OMG MTL specification, thus the new version of Acceleo was built to satisfy the latest standards. With the new standards, it was aimed to provide a code generator that combines nice tooling, simple syntax and efficient code generation.

The purpose of this project is to design and implement the ModelicaML code generator using the latest Acceleo version 3.X by re-factoring the old generator prototype.
1.3 Limitations

The main focus was to re-factor and implement the current code generator by using the Acceleo 3.X, by keeping the structure as close as possible to the original one. Due to the time limitations and the complexity of the work, there hasn’t been held any study on how to enhance the structure of the code generator.

1.4 Intended Audience

This report aims a target audience familiar with software development, having interest in code generation and at least basic knowledge of UML. Most of the technologies mentioned in the report are explained in Chapter 2, however for those who needs a better understanding of the concepts that they are unfamiliar with, it is recommended to examine the related references provided.

1.5 Thesis Overview

The rest of this thesis is organized as follows: Chapter 2 provides descriptions for the related technologies that are applied in this thesis. Chapter 3 compares the new Acceleo 3.x version with the old versions and states the main differences between them, whereas Chapter 4 describes the implementation process of the new code generator based on the prototype from the former one. Chapter 5 concludes the thesis work by summarizing the results and stating the possible future works.
Chapter 2

Technologies

This chapter is provided to explain the technologies stated in this report. The idea is to provide descriptions of the used technologies for those who are not familiar with them or need to refresh their minds to avoid confusions and assure a better understanding. Those who are familiar with the concepts described in this chapter are free to skip reading the entire chapter.

2.1 OpenModelica

OpenModelica is a project to develop a modeling and simulation environment based on Modelica language. It is open-source and the intended usage is for industrial as well as academic purposes. The project is supported by a non-profit organization Open Source Modelica Consortium (OSMC) [7].

2.1.1 Modelica

Modelica is a non-proprietary, object-oriented, equation based language that was designed to model complex physical systems that contain mechanical, electrical, electronic, hydraulic, thermal, pneumatic, fluid, control, electric power or process oriented components.

The main goal is to model dynamic behavior of technical systems. Models are described by several different equations types such as differential, algebraic and discrete equations. It has been used in industry since 2000. The language is free and has been developed by a non-profit organization called Modelica Association since 1996.

2.1.2 ModelicaML – A UML Profile for Modelica

ModelicaML is a graphical modeling language, designed to be used for the description of time-continuous and time-discrete/event-based system dynamics. It is designed to provide an efficient way of creating, reading, understanding and maintaining the Modelica Models. The language is defined as a graphical notation and provides different views (composition, inheritance, behavior) on Modelica models. It is an extended subset of Unified Modeling Language (UML) and it reuses concepts from the OMG Systems Modeling Language (SysML [8]).

UML/SysML implements the modeler with powerful descriptive constructs; however, the modeler has loosely defined semantics, stated as “semantic variation points” in the UML/SysML specifications. The main purpose of ModelicaML is to implement the modeler with powerful executable constructs and execution semantics based on Modelica language. In order to achieve that, ModelicaML uses a limited set of the UML and extends the UML meta-model by using the UML profiling mechanism with new constructs that will enable introducing missing Modelica concepts. ModelicaML models needs to be translated into Modelica code since ModelicaML is just the graphical notation. For this reason, the
execution semantics are defined by the Modelica language and finally by the Modelica compiler that will perform the translation of generated Modelica code into an executable form.

2.2 UML

UML is a modeling language that is the most widely used today. The meta-model includes a set of graphic notation techniques, used to create visual models of object-oriented software systems. For instance, a basic representation of class diagrams, their attributes, methods and relationships can be defined by UML. UML can be used to define a significant number of concepts, however most of these representations are defined in an informal way thus we might need to define new meta-models that would allow a more formal vision of reality for complex systems.

2.3 Papyrus

Papyrus is an open-source component of the Model Development Tools (MDT [10]) subproject aiming at providing an integrated and user-consumable environment for editing EMF-based models and supporting UML and SysML. It offers diagram editors for EMF modeling languages and the connection required to integrate these editors (GMF-based or not) with other MBD and MDSD tools. It additionally provides an advanced support of UML profiles that lets users define editors for DSLs, based on the UML2 standards and its mechanisms. The main feature of Papyrus is its strong customization mechanisms which can be used to create user-defined Papyrus perspectives with the same look thus it would feel like a native DSL editor. The Papyrus framework is used in ModelicaML Eclipse plug-in.

2.4 Acceleo

2.4.1 Acceleo 2.x

Acceleo is an open-source project that aims to provide a code generator (transferring models into code) for the people who aim to profit from a Model-driven approach in order to increase their software development productivity. It is licensed under the Eclipse Public License (EPL).

Acceleo (combined with Model Driven Architecture approach) is currently considered as one of the most powerful code generator tool of its generation for Model Driven Development (MDD [6]) and Model Driven Engineering (MDE [6]).

The development of Acceleo has been performed in the French company Obeo [11] as a result of several man years. The first stable release Acceleo 1.0.0 was available since 1 April 2006 whereas the last stable release Acceleo 3.2 was out since 27 October 2011 [5].

The tool is based on extensive research advances and industrial best practices. The main advantages offered are:

- High customizability
- Interoperability
- Easy kick off
- Traceability
The core functionalities of Acceleo can be listed as:

- Open Source
- Full integration within Eclipse
- Native use of EMF
- Edition and development environment
- Compatibility with UML1, UML2, Merise, DSL …
- Ability to target any technology

### 2.4.2 Acceleo 3.x

In year 2009, Acceleo project was accepted in the Eclipse Foundation while it was moving to the new version Acceleo 3. The language used in defining the code generator changed to use a new standard from OMG for model to text transformation, MOFM2T. The project was mostly developed in java and it is available under the Eclipse Public License (EPL [13]).

Acceleo was included for the first time in the Eclipse simultaneous release train as an official project of the Eclipse foundation with the release of Eclipse 3.6 Helios. It is integrated in the Eclipse IDE as an Eclipse plug-in. As an Eclipse plug-in, it is built on top of some key Eclipse technologies such as EMF [14] and starting from the Acceleo 3 release the Eclipse implementation of OCL [13] (a standard language from OMG [2] to define constraints of the models and navigate on them) has been used. Acceleo 3 tends to combine nice tooling, simple syntax and efficient code generation with the entire source from the 2.x stream.

Main features of Acceleo 3 can be listed as:

- Complete integration with the Eclipse environment and the EMF framework [14].
- Code/model Synchronization
- Incremental generation
- Easy adaptation for any kind of industrial projects
- Ease of update and management of templates
- Syntax coloring, auto-completion and error detection
- Improved speed and stability of compilation
- Improved support for generators with dynamic meta-models
- Documentation in completion and gover
- Improved traceability

### 2.5 Object Management Group (OMG)

OMG [15], the Object was founded in 1989 and is an open membership, non-profit computer industry standards consortium that focuses on producing and maintaining computer industry specifications for portable and reusable enterprise applications. The specifications of OMG implement the Model Driven Architecture (MDA [9]) by aiming to maximize ROI through a full lifecycle approach for the integration that includes multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. The specifications include UML [16], CORBA (Common Object Request Broker Architecture), CWM (Common Warehouse Model) and various industry specific standards.
Specifications are organized by the following categories:

**OMG Modeling Specifications:**
- UML
- MOF
- XMI
- CWM
- Profile specifications.

**OMG Middleware Specifications**
- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM).

**Platform Specific Model and Interface Specifications**
- CORBAservicesiv MOF Model to Text Transformation Language, v1.0
- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications.

The specification defines the MOF to Text Template Language (Mof2Text [17]), version 1.0. Mof2Text is aligned with UML 2.0, MOF 2.0, and OCL 2.0.

### 2.6 MOF Model to Text

The MOF Model to Text (mof2text) standard [17] states how to translate a model to various texts such as code, reports, deployment specifications, documents, etc. It essentially states how to transform a model into a linear text representation. An efficient way to state this requirement is to use a template method wherein a set of text templates that specifies the text to be generated from models are defined. A template-based method is used in which a “Template” is a text-based template with placeholder for data to be extracted from the models. The placeholders are expressions specified by meta model entities with queries and they form the primary mechanisms that select and extract the data from models. The extracted data is then converted into text fragments using an expression language increased with a string manipulation library. In order to handle complex transformation requirements, template can be composed. In case of large transformation, structure may consist of several modules with public and private parts. The following example illustrates a Template specification that generates a Java definition for a UML class.

### 2.7 EMF

Eclipse Modeling Framework is an Eclipse-based modeling framework and code generation facility developed to simplify the loading, handling and storage of models. EMF provides tools and runtime support to produce Java classes for the model, adapter classes to enable viewing and command-based editing of the model and a basic editor. It is not specific to any meta-model; therefore all kinds of models may be handled. It is based on a standard of meta-model description called “Ecore”, a subset of EMOF standard [18]. Models can be specified by using documents (Java, UML, XML) or modeling tools before they are imported into EMF.
Chapter 3

Acceleo 2.x vs Acceleo 3.x

This chapter aims to explain the basic concepts of the code generation tool Acceleo by stating the differences between the two versions Acceleo 2.X and Acceleo 3.X. It will start with explaining the basic structures of the two versions and provide a comparison at the end.

3.1 Acceleo 2.X

![Figure 1. Model Driven Development with Acceleo 2.X](image)

Figure 1. Model Driven Development with Acceleo 2.X

### 3.1.1 Module

Modules are Acceleo plug-ins for code generation (Figure 1). Each module is a technology specific component and they provide on-the-shelf industrialization solutions with high reliability and fast implementation.

A module is an “.mt” file that consists of several templates describing the data required for the generation of the source code from a meta-model. Within each template, there are scripts functions to customize the generator accurately.
The module file must start with metamodel declaration in the following form:

```<% metamodel http://www.eclipse.org/uml2/2.0.0/UML %>`
```

Modules can depend on each other for their execution. In that case, the depending modules must be imported in the following form:

```<% metamodel http://www.eclipse.org/uml2/2.0.0/UML
import org.acceleo.modules.uml2.services.Uml2Services
import org.acceleo.modules.uml2.services.StringServices %>`
```

---

Figure 2. Example Module Acceleo 2
3.1.2 Scripts

Scripts consist of Acceleo statements that are used to generate text (Figure 2). They are located in template files (containing the structure of the file to be generated) or library files (containing scripts that can be used by several templates). Scripts are written in the following form:

```
<%script type="Class" name="fullFilePath"%>
<% if (hasStereotype("Entity")) { %>
    /<% jdbcPackage.toPath() %>/Jdbc<% name %>Dao.java
<% } %>
```

3.1.3 Execution Chain

Execution chains are used to execute a generation for a target application. Chains makes it possible to combine several generations and operations on models. They are defined in order to avoid the creation of long and tedious manual scripts for generation. We can define that chains are to Acceleo what “Makefile” scripts are to traditional development. Creating a chain involves the following steps:

- Adding a uml model to be used as the input of the generation
- Specifying the meta-model URI to be used
- Adding generation module or modules (.mt files)
- Adding the target folder for the output of the generation
- Specify the log file URI

![Figure 3. Generation Chain Example](image)
3.1.4 UML Model

UML models are added to the execution chains to be used as the input to the generators and the output source code are produced by the generator from them.

Figure 4. UML model example
3.2 Acceleo 3.x

3.2.1 Modules

A module (Figure 6) is a “.mtl” file that consists of templates that describe the necessary parameters to generate code, and/or queries that are invoked to extract data from manipulated models. Several scripts are usually written within each template in order to enable fine tuning of the generation. The module file must start with the module declaration in the following form:

```
[module <module_name>('metamodel_URI_1')]
```

Modules can depend on each other for their execution. In that case, the depending modules have to be imported in the following form:

```
[ import qualified::name::of::imported::module/]
```

3.2.2 Templates

Templates are scripts that consist of Acceleo statements, used to generate text (Figure 6). Templates are written in the following form:

```
[template public template_name(t : Type)]
...
[/template]
```
3.2.3 Queries

Queries are used to extract data from the manipulated models by using OCL. They must have return values or collection of values. The specification of queries allows them to return always the same value each time they are called with the same arguments. They are defined in "query .../" tags. Here are some examples of queries:

```
[query private getterPrefix(feature : EStructuralFeature) :
String = if feature.eType.name = 'EBoolean' then
  'is'
else
  'get'
endif
]
```

```
[query private valueGetter(class : EClass,feature: EStructuralFeature:
String = if feature.eType.isPrimitive() then
  primitiveValueGetter(class, feature) else
  class.name.toLowerFirst().concat('.get').concat(feature.name.toUpperFirst()).concat('()')
]
```

Calling a query:
```
[feature.getterPrefix()]
```
3.2.4 Java Services

It is possible to call Java services from Acceleo for multiple reasons. Since acceleo modules (".mtl" files) only allow for acceleo code, the Java methods must be included in their own java classes (".java" files) and can be called through “Java service wrappers”. Java service wrappers are Acceleo queries that invoke the Java code and can be automatically created from the Java file by Acceleo.

```java
package org.eclipse.acceleo.module.sample.services;

import java.util.List;

public class UML2Services {
    public boolean hasStereotype(Class clazz, String stereotypeName) {
        List<Stereotype> stereotypes = clazz.getAppliedStereotypes();
        for (Stereotype stereotype : stereotypes) {
            if (stereotype.getName().equals(stereotypeName)) {
                return true;
            }
        }
        return false;
    }
}
```

Figure 8. Java Service Example

```acceleo
(module uml2Services)('[http://www.eclipse.org/uml2/1.0.0/OCL/]')/
(query public hasStereotype(arg0 : Class, arg1 : String) : Boolean
   - invokes('org.eclipse.acceleo.module.sample.services.UML2Services', 'hasStereotype(org.eclipse.uml2.uml.Class, java.lang.String)', 'Sequence[arg0,arg1]')/)
```

Figure 9. Query to call Java Service
3.2.5 Basic Language Constructs

3.2.5.1 File Tags

They are used to inform Acceleo engine that the content of the [file …] tag has to be generated in an actual file. File tags are defined in the following form:

```
[file(<uri_expression>, <append_mode>, <outencoding>)]
(...)
[/file]
```

<uri_expression> indicates the output file name;
<append_mode> (optional) indicates whether the output text must be appended to the file or replace its content;
<outencoding> (optional) indicates the encoding to be used for the output file. This encoding doesn’t need to be the same as the module’s encoding.

![Figure 10. File Tag Example](image)

3.2.5.2 For loops

For loops are used to allow the code to be executed repeatedly. The syntax is shown in the following figure.

In order to specify the behavior to be run:
Before the iterations; before()
Between each iteration; between()
After the iterations; after() is used.
3.2.5.3 If Conditions

If conditions are used as in general programming languages and defined as the following:

```
[if (condition)]
...
[elseif (condition)]
...
[else]
...
[/if]
```

3.2.5.4 Let blocks

Variables in Acceleo are `final`, their value cannot be changed after they have been initialized. Variables only exist inside of a let block. Their lifetime consequently cannot exceed that of their template. They are only visible inside of the let block, which means that templates called within this block cannot "see" them.

```
[let (variableName : VariableType = expression)]
...
[/let]
```
3.2.5.5 Comments

Comments are entered in [comment/] blocks.

Examples:
1. [comment Here is some comment of utter importance/]

2. [comment]
   Here is some commented text, which cannot contain a right bracket character.
   [/comment]
3.3 Acceleo 3.x vs Acceleo 2.x

The main difference between Acceleo 2.X and Acceleo 3.X releases is that the language used in defining the Acceleo3.X code generator is a new standard from OMG [2] that is called MOFM2T [17]. Additionally, OCL [13] (a standard language from OMG) is used in Acceleo 3 to define constraints of the models and navigate through them. These changes in the languages used in the new Acceleo have resulted on different definitions of the expressions to be used in the new generators. The main equivalences of these expressions between Acceleo 2 and Acceleo 3 are provided in the Appendix section at the end of this report (Appendix A).

Furthermore, with the Acceleo 3 more improvements were introduced in the following areas:

- Compatibility and upgrading
- Tooling
- Editing
- Language simplifications
- More documentation
- Performance

You can refer to the Acceleo site for further details about the improvements [5].
Chapter 4

Implementation

This chapter explains the process of the work carried out in the implementation of the ModelicaML code generator using the latest Acceleo version 3.X by refactoring the old generator prototype. It starts with describing the current ModelicaML generator based on Acceleo 2.X and continues with explaining the migration process to Acceleo 3.X.

4.1 ModelicaML Code Generator based on Acceleo 2.X

This section aims to provide a basic picture of how the current generator is structured based on Acceleo 2 [4].

Figure 12. Types of files (ModelicaML Code Generator - Acceleo 2.X)
The ModelicaML code generator [Figure 12] consists of several types of files such as:

- **Template**: `.mt` files that determine the structures/layout of the files to be generated.
- **Library**: `.mt` files that contain scripts used by templates.
- **Java service**: `.java` files that contain set of operations coded in Java, permitting complex operations on primitive types and elements of the meta-model.
- **Properties file**: `.properties` file that contains default parameters in the form of key-value pairs.
- **Chain file**: `.chain` file used to execute a bulk generation for a target operation.

The ModelicaML code generator is defined to generate executable Modelica code from graphical models. For detailed information about ModelicaML, you can refer to ModelicaML website [1] where you can find further information and publications on the topic.
4.2 Migration of the ModelicaML Code Generator from Acceleo 2 to Acceleo 3

This section aims to describe the migration process of the ModelicaML Code Generator from Acceleo 2 to Acceleo 3. It starts with introducing the former ModelicaML Code Generator that was built using the former Acceleo 2.X version. The migration process to Acceleo 3 will be explained with regards to the main steps followed:

- **Translation**: Migration of the files composing the ModelicaML Code generator from Acceleo 2.X to Acceleo 3.X
- **Configuration**: Configuration of the ModelicaML Code generator files migrated to Acceleo 3.X
- **Testing/Bug fixing**: Testing and bug fixing by debugging the files that contains faults. Validating the Acceleo 3.X generator to verify it produces the same results with the generator based on Acceleo 2.X.
- **Plug in**: Creating the new plug in based on Acceleo 3.

4.2.1 Translation

This section describes the process for migrating the files composing the ModelicaML Code generator Acceleo 2 into Acceleo 3. It has been decided to start with the translation of the modules (.mt files) followed by the Java Services, properties file and the main generator file.

4.2.1.1 Translation of the Modules

The process of the migration started with defining the modules (.mt files) to be translated into Acceleo 3 modules (.mtl files). In the base generator, 19 modules have been defined which are categorized as:

- Activity to Modelica: 1 module
- Additional: 1 module
- Annotation: 1 module
- State Machine to Modelica: 8 modules
- Core: 7 modules
- ModelicaML to Modelica: 1 module

Acceleo 3 offers an automatic migration system that is designed to migrate an Acceleo 2 project into an Acceleo 3 project. It was decided to start with examining the performance and accuracy of this functionality. After migrating several modules using the migration functionality of Acceleo 3, the resulting modules (.mtl files) were examined. The generated files contained a significant amount of errors.

- Part of these errors were caused because of the unavailability of the modules that are defined as inherited.
- Part of the errors were caused due to incorrect or non-translation of the code and their interrelation with the correctly translated code.

After the examination of the automatic migration functionality, it was concluded that the functionality needs further improvement in order to be used for automatic migration of Acceleo 2 projects into
Acceleo 3 projects. However, the automatically generated files provide a general understanding of part of the expression differences between Acceleo 2 and Acceleo 3 (OCL):

- EObject Services,
- Request Services,
- Resource Services,
- String Services,
- System Services,
- Xpath Services,
- ENode Services,
- Context Services,
- Properties Services.

Several of these services can be partly translated automatically using the functionality. Due to the immaturity of the automatic migration functionality, it was decided to use the functionality to create an example base “.mtl” file for each module to be manually edited and further translated into Acceleo 3 expressions by studying the equivalences of the expressions [Appendix A] between the two versions.

Starting from the core modules, for all 19 modules a corresponding .mtl files were created using the automatic migration functionality of Acceleo 3. After each creation of the new .mtl Acceleo 3 module, the generated file was analyzed and the errors were tried to be fixed. The process for this first error fixing of the automatically generated files was:

- Define the types of errors
- Spot the errors caused by mistranslation of the expressions to fix them first
- Spot the errors caused by interrelation between translated and non-translated expressions
- Spot the errors cause by missing components that have not been migrated yet and mark them to handle after the dependent components were migrated
- Manually translate the expressions that were not migrated automatically
- Fix the syntax errors according to the Acceleo 3 syntax to manage the interrelations with errors
- Import dependent modules according to the Acceleo 3 syntax (not managed automatically)

After following this process for all 19 .mtl files that were created, the amount of errors that they contain have been reduced to a minimum amount. The remaining errors were caused by the missing components and configuration details therefore they were planned to be fixed after the creation of all related files and fixing the configuration.

4.2.1.2 Translation of the Java files

After the migration of the module files, it was decided to handle the Java files as the following step since Java files are called from the module files therefore they need to be imported in the module files.

Java Files can be useful to perform an action that Acceleo does not offer. In Acceleo 2, it is possible to invoke a Java method directly from the module files by calling it from the imported class. Unlike Acceleo 2, in Acceleo 3 Java methods cannot be invoked directly from the module files. The Acceleo non-standard library provides a service invoke that allows to call Java methods[3.2.4]. The Java services to be invoked must be wrapped into an Acceleo query[3.2.3].

In order to create the queries that will invoke the Java methods, an extra module file was created for each Java file that contains the queries to call each Java method from the corresponding Java file. Acceleo offers a functionality to create the queries automatically from the Java files inputted.
In the base generator, there were several main Java files defined and corresponding Java Service Wrapper modules were created. These modules later were imported into the related main modules in order to allow them to call the Java methods needed.

![Figure 13. Example of Queries calling Java Services](image)

4.2.1.3 Translation of Properties File

Properties file are used to store some properties that will be used during the generation. The parameters are stored in the form of key-value pairs. In our base generator, a properties file was used which did not require any translation within the file itself, however the usage of properties file has been changed/improved at the Acceleo 3 release. This will be explained in the next chapter when explaining the main module.

The main advantage of properties file is that it allows the creator of the generation to modify some properties without changing the generator file itself.
4.2.1.4 Main Module

In Acceleo 2, execution chains are created to apply generation projects to models. In the execution chain, the followings can be defined for a generation:

- Initial Model: path to the UML model as input to the generation
- Destination of generation: path to where the generations results will be stored
- Error log file: path to the file where the errors that occur during the generation are logged.
- Metamodel: Metamodel to be used in the generation
- Applied Generation module: path to the generation module

Whereas in Acceleo 3, most of this parameters are entered in the interface and no chain files are needed. In Acceleo 3, a “main module” needs to be created as the entering point of the generation, unlike Acceleo 2. The role of the main module is to delegate all of the modules which will create files – that contain the file tags [3.2.5.1]. The main model contains the @main annotation.

For our new generator, a main module was created that includes invokes to the 2 files that will create files (Figure 15).
When the main module is created, a Java Main module is automatically created to be used as the entry point of the Main generation module. This Main Java module was edited with the path of our properties file (Figure 16).

```
/*
propertiesFiles.add("org.openmodelica.modelicaML.generate.modelica.main.ModelicaML_config");
return propertiesFiles;
*/

/**
 * Adds a properties file in the list of properties files.
 *
 * @param propertiesFile
 * The properties file to add.
 * @param generated
 * @since 3.1
 * @see
 *
 * public void addPropertiesFile(String propertiesFile) {
 * this.propertiesFiles.add(propertiesFile);
 */
```
4.2.2 Configuration

The process for translation of the module files (.mt) were completed together with the creation of complementary components such as the main module, properties file and the java service wrappers (queries used to call the Java Services). The inherited modules were imported into each modules to establish their connection. After configuring all components, the new generator looked as in the following figure:

![Diagram of the new generator structure](image)

**Figure 17. Main Java – Properties file addition**

In the new generator, 35 files have been defined which are categorized as:

- **Activity to Modelica**: 1 module
- **Additional**: 1 module
- **Annotation**: 1 module
- **Cg helpers**: 1 java service files, 1 module (queries)
- **Main**: 1 main module, 1 java file, properties file
- **ModelicaML2Modelica**: 1 module
- **Services**: 3 java service files, 3 modules (queries)
- **State Machine to Modelica**: 8 modules
- **Core**: 7 modules
- **Services (core)**: 3 java service files, 3 modules (queries)

After the configuration is completed, the generator has worked and successfully managed to create all the files that the former generated created using the sample models.
4.2.3 Testing / Debugging

Numerous testing activities have been performed continuously during the migration from Acceleo 2 to Acceleo 3.

At the first stage, when the module files have been migrated individually, the Eclipse functionality that shows the errors contained in the files together with the message explaining the potential causes, was used to perform the first bug fixing in the migrated files. This processed have been repeated over and over while migrating more files and importing their dependencies.

After all the files were created and the configuration was established, it was time to test the generator with UML model inputs and compare the generated output with the output that was generated by the former generator (Acceleo 2).

The first testing of the new generator were performed using two different UML models. These UML models are used as the input to our generator to create modelica code. The generated files that contain the modelica code can be used for simulating a two tanked liquid flow system that is represented by the given models.

The first outputs were faulty as expected and led us to the second phase of our testing approach: Debugging.

4.2.3.1 Debugging

Acceleo provides a debugger by simply launching the generation in the Debugging mode. It works just like the Java debugger [19] and allows to put breakpoints to move instructions by instructions in the execution of the generation. The new Acceleo debugger also allows us to see the state of the variables and a given point in the evaluation process along with the values of the variables for a given stack frame.
Debugging was performed over and over in order to spot all the bugs and fix them. The errors found were mostly caused by:

- MOFM2T translation faults
- OCL expression faults
- Configuration faults
- Java service wrappers
- Referring to a null object (Null pointer exceptions)
- ENode Services (faulty method calls such as sorting methods)

**4.2.3.2 Comparison of the generation results:**

As mentioned at the beginning of this chapter, we had two main models that were used testing the new generator: A model that represents a Two Tanks system and another one for a Portable Water system. The process started with inputting these models to the former generator and saving the generation results to be compared with the generated results from the new generators using the same models. For this the basic Eclipse functionality “Compare With” was used. Each file was compared individually with the corresponding one from the other generator (Figure 20)
After the new code generator was created and tested it was time to create a plug-in project that contains everything needed to launch the Acceleo generations. The plug-in project facilitates the deployment of the Acceleo generators in the Eclipse platform. The process of creating the plug-in was relatively simple by a user interface that is offered by Acceleo. The created Plug-in project is configured to include information on what type of files to be inputted to the generator and where the generated files should be located. Our plug-in project is named as “org.openmodelica.modelicaml.generate.modelica.ui” and looks like the following figure:

The plug-in is used in the deployment of the generator created with Acceleo.
Chapter 5

Results & Discussion

This chapter aims to provide the results of this project and conclude the thesis work by stating the potential future improvements.

5.1 Results & Conclusion

We can categorize this project as an upgrade project that aims to re-implement the current ModelicaML code generator by using the latest version of the used framework Acceleo that is now based on new standards. The main objective of this project was to re-factor the current ModelicaML code generator – that was developed by using the former Acceleo2.X framework – by using the new Acceleo 3.X release. The new release is based on the new OMG MTL specification and in the definition of the generators a new setup of standards are used such as MOFM2T and OCL.

The transformation was performed following the steps that have been defined in the previous chapters and the MolicaML code generator was created using the Acceleo 3.X framework. The initial expectation from the new generator is to produce the exact same results that the former generator does when given the same input. After testing the new generator with several model inputs, it was observed that the generated files were the same as the former generator produces.

The performance of the new generator was not evaluated. The main reason for that is the scope of this project was limited. The scope was clearly creating the code generator by re-factoring the former generator. There has not been any work done on how to improve the performance of the generator. Furthermore, the infrastructure of the generator is significantly complex and requires more knowledge and expertise in the area to be able to analyze it for further improvements.

5.2 Future Work

The ModelicaML code generators, both the former one built with Acceleo 2.X and the new one built using the new Acceleo 3.X framework, can be analyzed from a technical perspective in details to explore improvement areas such as:

- Performance
- New functionality
- Efficiency
Appendix A

Acceleo 2.x to 3.x equivalences

This appendix section demonstrates some of the expression equivalences between Acceleo 2.x to Acceleo 3x[].

A.1 Miscellaneous

<table>
<thead>
<tr>
<th>Acceleo 2 Expression</th>
<th>Acceleo 3 Expression (OCL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;%metamodelhttp://www.eclipse.org/uml2/2.0.0/UML%&gt;</td>
<td>[module module_name('<a href="http://www.eclipse.org/uml2/2.0.0/UML')/">http://www.eclipse.org/uml2/2.0.0/UML')/</a>]</td>
</tr>
<tr>
<td>&lt;%import module.mt%&gt;</td>
<td>[import module/] or [import org::eclipse::acceleo::mypackage::module/]</td>
</tr>
<tr>
<td>&lt;%import service.java%&gt;</td>
<td>Services no longer need to be imported</td>
</tr>
<tr>
<td>&lt;%list + object%&gt;</td>
<td>[list-&gt;including(object)/]</td>
</tr>
<tr>
<td>&lt;%list1 + list2%&gt;</td>
<td>[list1-&gt;union(list2)/]</td>
</tr>
<tr>
<td>&lt;%string + object%&gt;</td>
<td>[string + object/]</td>
</tr>
<tr>
<td>&lt;%list - object%&gt;</td>
<td>[list-&gt;excluding(object)/]</td>
</tr>
<tr>
<td>&lt;%list1 - list2%&gt;</td>
<td>[list1-&gt;asSet() - list2-&gt;asSet()/] (Note that this no longer keeps duplicates, which is not strictly equivalent to Acceleo 2)</td>
</tr>
<tr>
<td>&lt;%list1 \p\p list2%&gt;</td>
<td>[list1-&gt;union(list2)/]</td>
</tr>
<tr>
<td>&lt;%list1 &amp;&amp; object%&gt;</td>
<td>element = object/)</td>
</tr>
<tr>
<td>&lt;%list1 &amp;&amp; list2%&gt;</td>
<td>[list1-&gt;iterate(element; result : Sequence(T) = Sequence{ } if list2-&gt;includes(element) then result-&gt;including(element) else result endif)/]</td>
</tr>
<tr>
<td>&lt;%list1 == list2%&gt;</td>
<td>[list1 = list2/]</td>
</tr>
</tbody>
</table>
A.2 Eobject Services

<table>
<thead>
<tr>
<th>Acceleo 2 Expression</th>
<th>Acceleo 3 Expression (OCL)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;%x.eAllContents%&gt;</code></td>
<td>[x.eAllContents()]/</td>
</tr>
<tr>
<td><code>&lt;%x.eAllContents(“EClass”)%&gt;</code></td>
<td>[x.eAllContents(EClass)}/</td>
</tr>
<tr>
<td><code>&lt;%x.eClass%&gt;</code></td>
<td>[x.eClass()]/</td>
</tr>
<tr>
<td><code>&lt;%x.eContainer%&gt;</code></td>
<td>[x.eContainer()]/</td>
</tr>
<tr>
<td><code>&lt;%x.eContainer(“EPackage”)%&gt;</code></td>
<td>[x.eContainer(EPackage)}/</td>
</tr>
<tr>
<td><code>&lt;%x.eContainingFeature%&gt;</code></td>
<td>[x.eContainingFeature()]/</td>
</tr>
<tr>
<td><code>&lt;%x.eContainmentFeature%&gt;</code></td>
<td>[x.eContainmentFeature()]/</td>
</tr>
<tr>
<td><code>&lt;%x.eContents%&gt;</code></td>
<td>[x.eContents()]/</td>
</tr>
<tr>
<td><code>&lt;%x.eCrossReferences%&gt;</code></td>
<td>[x.eCrossReferences()]/</td>
</tr>
<tr>
<td><code>&lt;%x.eResource%&gt;</code></td>
<td>[x.eResource()]/</td>
</tr>
<tr>
<td><code>&lt;%x.eResourceName%&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;%x.getRootContainer%&gt;</code></td>
<td>[x.ancestors()-&gt;last()]/</td>
</tr>
<tr>
<td><code>&lt;%load(“/UML2_LIBRARIES/UML2PrimitiveTypes.library.uml2”)%&gt;</code></td>
<td>No direct equivalent: Acceleo 3 accepts more than one model element as arguments of a generation. An alternative would be to use a custom Java service for this need.</td>
</tr>
</tbody>
</table>

A.3 Request Services

<table>
<thead>
<tr>
<th>Acceleo 2 Expression</th>
<th>Acceleo 3 Expression (OCL)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;%list.select(“eAttributes.nSize() &gt; 0”)%&gt;</code></td>
<td>[list-&gt;select(eAttributes-&gt;size() &gt; 0)}/</td>
</tr>
<tr>
<td><code>&lt;%list.select(“eAttributes.nSize()”, “0”)%&gt;</code></td>
<td>[list-&gt;select(eAttributes-&gt;size() = 0)}/</td>
</tr>
</tbody>
</table>
A.4 String Services

<table>
<thead>
<tr>
<th>Acceleo 2 Expression</th>
<th>Acceleo 3 Expression (OCL)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;%x.length%&gt;</code></td>
<td><code>[x.size()]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.toUpperCase%&gt;</code></td>
<td><code>[x.toUpper()]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.toLowerCase%&gt;</code></td>
<td><code>[x.toLower()]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.toU1Case%&gt;</code></td>
<td><code>[x.toUpperFirst()]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.toL1Case%&gt;</code></td>
<td><code>[x.toLowerFirst()]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.substring(1, 3)%&gt;</code></td>
<td><code>[x.substring(2, 3)]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.substring(1)%&gt;</code></td>
<td><code>[x.substring(2)]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.replaceAll(&quot;test&quot;, &quot;replaced&quot;)%&gt;</code></td>
<td><code>[x.replaceAll('test', 'replaced')]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.replaceAll(&quot;\\w&quot;, &quot;$0&quot;)%&gt;</code></td>
<td><code>[x.replaceAll('\w', '$0')]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.replaceFirst(&quot;test&quot;, &quot;replaced&quot;)%&gt;</code></td>
<td><code>[x.replace('test', 'replaced')]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.replaceFirst(&quot;\\w&quot;, &quot;$0&quot;)%&gt;</code></td>
<td><code>[x.replace('\w', '$0')]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.trim%&gt;</code></td>
<td><code>[x.trim()]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.startsWith(y)%&gt;</code></td>
<td><code>[x.startsWith(y)]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.endsWith(y)%&gt;</code></td>
<td><code>[x.endsWith(y)]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.equalsIgnoreCase(y)%&gt;</code></td>
<td><code>[x.equalsIgnoreCase(y)]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.matches(y)%&gt;</code></td>
<td><code>[x.matches(y)]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.charAt(4)%&gt;</code></td>
<td><code>[x.substring(5, 5)]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.indexOf(y)%&gt;</code></td>
<td><code>[x.substring(5, x.size()).indexOf(y) + 4]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.indexOf(y, 4)%&gt;</code></td>
<td><code>[x.substring(5, x.size()).indexOf(y) + 4]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.lastIndexOf(y)%&gt;</code></td>
<td><code>[x.substring(5, x.size()).lastIndexOf(y) + 4]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.lastIndexOf(y, 4)%&gt;</code></td>
<td><code>[x.substring(5, x.size()).lastIndexOf(y) + 4]/</code></td>
</tr>
<tr>
<td><code>&lt;%x.split(y)%&gt;</code></td>
<td>No direct equivalent, use String::tokenize(String)</td>
</tr>
</tbody>
</table>
A.5 System Services

<table>
<thead>
<tr>
<th>Acceleo 2 Expression</th>
<th>Acceleo 3 Expression (OCL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;%i()%&gt;</td>
<td>[i/]</td>
</tr>
<tr>
<td>&lt;%args(0)%&gt;</td>
<td>arguments are explicit and named in Acceleo 3</td>
</tr>
</tbody>
</table>

A.6 Xpath Services

<table>
<thead>
<tr>
<th>Acceleo 2 Expression</th>
<th>Acceleo 3 Expression (OCL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;%x.ancestor%&gt;</td>
<td>[x.ancestors()/]</td>
</tr>
<tr>
<td>&lt;%x.parent%&gt;</td>
<td>[x.eContainer()/]</td>
</tr>
<tr>
<td>&lt;%self%&gt;</td>
<td>[self/]</td>
</tr>
<tr>
<td>&lt;%x.child%&gt;</td>
<td>[x.eContents()/]</td>
</tr>
<tr>
<td>&lt;%x.descendant%&gt;</td>
<td>[x.eAllContents()/]</td>
</tr>
<tr>
<td>&lt;%x.precedingSibling%&gt;</td>
<td>[x.precedingSiblings()/]</td>
</tr>
<tr>
<td>&lt;%x.preceding%&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;%x.followingSibling%&gt;</td>
<td>[x.followingSiblings()/]</td>
</tr>
</tbody>
</table>

A.7 Enode Services

<table>
<thead>
<tr>
<th>Acceleo 2 Expression</th>
<th>Acceleo 3 Expression (OCL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;%eObject.adapt(&quot;EObject&quot;)%&gt;</td>
<td>[EObject/]</td>
</tr>
<tr>
<td>&lt;%eObject.adapt(&quot;ENodeList&quot;)%&gt;</td>
<td>[if (eObject.oclIsUndefined())][Sequence { }][else][Sequence{eObject}]/[/if]</td>
</tr>
<tr>
<td>&lt;%eObject.adapt(&quot;String&quot;)%&gt;</td>
<td>[if (eObject.oclIsUndefined())][/][else][eObject.toString()][/][if]</td>
</tr>
<tr>
<td>&lt;%eObject.adapt(&quot;Boolean&quot;)%&gt;</td>
<td>[not eObject.oclIsUndefined()/]</td>
</tr>
<tr>
<td>&lt;%eObject.adapt(&quot;int&quot;)%&gt;</td>
<td>[eObject-&gt;size()/]</td>
</tr>
</tbody>
</table>
can only return empty enodes in Acceleo 2, so migration is simply "nothing"
Design and Implementation of the ModelicaML Code Generator using Acceleo 3.x

```plaintext
<%boolean.adapt("String")%> (boolean.oclIsUndefined())[Sequence { }][else][Sequence {boolean}]/[/if]
<%boolean.adapt("Boolean")%> [not boolean.oclIsUndefined() and boolean]/
<%boolean.adapt("int")%> [if (not boolean.oclIsUndefined() and boolean)][1/][else][0]/[/if]
<%boolean.adapt("double")%> [if (not boolean.oclIsUndefined() and boolean)][1.0/][else][0.0]/[/if]
<%boolean.adapt("List")%> [if (boolean.oclIsUndefined())][Sequence { }][else][Sequence {boolean}]/[/if]
<%boolean.adapt("ENode")%> [boolean]/
<%int.adapt("EObject")%> can only return empty enodes in Acceleo 2, so migration is simply "nothing"
<%int.adapt("ENodeList")%> [if (int.oclIsUndefined())][Sequence { }][else][Sequence {int}]/[/if]
<%int.adapt("String")%> [if (int.oclIsUndefined())][else][int.toString()]/[/if]
<%int.adapt("Boolean")%> [not int.oclIsUndefined() and int > 0/]
<%int.adapt("int")%> [if (int.oclIsUndefined())][0/][else][int]/[/if]
<%int.adapt("double")%> [if (int.oclIsUndefined())][0.0/][else][int.toString().toReal()]/[/if]
<%int.adapt("List")%> [if (int.oclIsUndefined())][Sequence { }][else][Sequence {int}]/[/if]
<%int.adapt("ENode")%> [int/]
<%double.adapt("EObject")%> can only return empty enodes in Acceleo 2, so migration is simply "nothing"
<%double.adapt("ENodeList")%> [if (double.oclIsUndefined())][Sequence { }][else][Sequence {double}]/[/if]
<%double.adapt("String")%> [if (double.oclIsUndefined())][else][double.toString()]/[/if]
<%double.adapt("Boolean")%> [not double.oclIsUndefined() and double > 0/]
<%double.adapt("int")%> [if (double.oclIsUndefined())][0/][else][double.toString().toInt()]/[/if]
<%double.adapt("double")%> [if (double.oclIsUndefined())][0.0/][else][double]/[/if]
<%double.adapt("List")%> [if
```
Design and Implementation of the ModelicaML Code Generator using Acceleo 3.x

```
<%double.adapt("ENode")%>
<%list.cast("EClass")%>
<%list.filter("EClass")%>
<%current()%>
<%current(1)%>

iterator variables are named in Acceleo 3

<%node.debug%>
<%node.trace%>
<%node.trace("name")%>
<%nPut("key", object)%>
<%nGet("key")%>
<%nPeek%>
<%nPop%>
<%nPush%
<%list.nMinimize%>
<%list.minimize%>
<%list.nContains(item)%>
<%list.nFirst%>
<%list.nLast%>
<%list.nGet(0)%>
<%list.nGet(0, 1)%>
<%list.nReverse%>
<%list.reverse%>
<%list.nSize%>
<%list.nSort%>
<%list.nSort(name)%>
```

(double.oclIsUndefined())[Sequence{ }][else][Sequence{double}])/[if]
A.8 Properties Services

<table>
<thead>
<tr>
<th>Acceleo 2 Expression</th>
<th>Acceleo 3 Expression (OCL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;%getBestProperty(&quot;partial.key&quot;)%&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;%getProperty(&quot;key&quot;)%&gt;</td>
<td>[getProperty('key')/]</td>
</tr>
<tr>
<td>&lt;%getProperty(&quot;propertiesFile&quot;, &quot;key&quot;)%&gt;</td>
<td>[getProperty('propertiesFile', 'key')/]</td>
</tr>
</tbody>
</table>

Appendix B

Example code:

Main.mtl

```
[comment encoding = UTF-8 /]
[module main('http://www.eclipse.org/uml2/3.0.0/UML')]
[import org::openmodelica::modelicaml::generate::modelica::uml2modelica::core::_0_main_uml2modelica/]

[template public mainPackage(aPackage : Package)]
[comment @main /]
[aPackage.ownedElement->filter(Classifier).stored_definition()]
[aPackage.package_code()]
[/template]
_1_file_name.mtl
```
Design and Implementation of the ModelicaML Code Generator using Acceleo 3.x

[module _1_file_name('http://www.eclipse.org/uml2/3.0.0/UML')]

import
org::openmodelica::modelicaml::generate::modelica::uml2modelica::services::umlse
rvices[/]
import
org::openmodelica::modelicaml::generate::modelica::uml2modelica::services::Strin
gUtils[/]
import
org::openmodelica::modelicaml::generate::modelica::cg::helpers::cgconfigurationm
anager[/]
import
org::openmodelica::modelicaml::generate::modelica::uml2modelica::core::helper[/]

[template public prefix(aNamedElement : NamedElement) post(trim())]
getProperty('c_outputFolder')[/]
[/template]

[template public file_name(aClassifier : Classifier) post(trim())]
[if (aClassifier.getCGNameSpace()->notEmpty())]
[if (qualifiedName.startsWith(aClassifier.getCGNameSpace()))]
file_name_string()[/if]
[/if][else]
file_name_string()[/if]
[/template]

[template public file_name(aPackage : Package) post(trim())]
[if (aPackage.getCGNameSpace()->notEmpty())]
[if (qualifiedName.startsWith(aPackage.getCGNameSpace()))]
file_name_string()[/if][else]
file_name_string()[/if]
[/template]

[template public file_name_string(aPackage : Package) post(trim())]
[prefix()][/][qualifiedName.replaceSpecCharExceptThis('::').replaceAll('::',
'\\/')./package.mo [comment]CHECK THIS LATER .replaceAll('::',
'\\/').[[/comment]
[/template]

[template public file_name_string(aClassifier : Classifier) post(trim())]
[if (aClassifier.ownerIsAValidPackageFolder() = true and
aClassifier.hasStereotype(getProperty('s_package')) or
aClassifier.hasStereotype(getProperty('s_operator')))]
[prefix()][/][qualifiedName.replaceSpecCharExceptThis('::').replaceAll('::',
'\\/')./package.mo
[else][comment]<%-- for a Modelica class --%>[/comment]
[if (aClassifier.isStoredDefinition())]
[prefix()][/][qualifiedName.replaceSpecCharExceptThis('::').replaceAll('::',
'\\/')./mo
[/if]
[/if]
[query public ownerIsAValidPackageFolder(aClassifier : Classifier) : Boolean =
if (owner.eClass().name = 'Model' or owner.eClass().name = 'Package') then true
else if ((owner.hasStereotype(getProperty('s_package')) or
owner.hasStereotype(getProperty('s_operator'))) and not
(hasStereotype(getProperty('ModelicaModelProxy')))) then
if (owner.owner.eClass().name = 'Model' or owner.owner.eClass().name = 'Package')
then true
else if (owner.owner.hasStereotype(getProperty('s_package')) or
owner.owner.hasStereotype(getProperty('s_operator'))) then
true
else false
endif
else false
endif
else false
endif
endif
/
]

[query public isStoredDefinition(aClassifier : Classifier) : Boolean =
if (aClassifier.ownerIsAValidPackageFolder() and
aClassifier.isModelicaRestrictedClass() and
not aClassifier.isProtected() and not aClassifier.isRedefine() and
not (aClassifier.hasStereotype(getProperty('ModelicaModelProxy'))) and
not aClassifier.isReplaceable()) then true else false endif
/
]
References

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/