



Faculty of Science, Technology and
Communication
Computer Science/ Telecommunications
University of Luxembourg
6, rue Coudenhove-Kalergi
L-1359 Luxembourg



TESMA : Requirements, Design and
Implementation of a Teaching Specification
Management and Assessment tool

Benjamin JAHIC



FACULTY OF SCIENCE, TECHNOLOGY AND COMMUNICATION

Requirements, Design and Implementation of a Teaching Specification, Management and Assessment tool

Thesis Submitted in Partial Fulfilment of the
Requirements for the Degree of Master in Information
and Computer Sciences

Author:
Benjamin JAHIC

Supervisor:
Prof. Dr. Nicolas GUELFY

Reviewer:
Ass. Prof. Dr. Nicolas NAVET

Advisor:
Dr. Benoit RIES

August 2016

Contents

Declaration of Authorship	1
Abstract	3
Acknowledgements	5
1 Introduction	7
1.1 Problem Description	7
1.2 Solution Overview	8
1.3 Research	9
1.4 Thesis Overview	10
2 Background	11
2.1 Domain specific languages	11
2.2 Excalibur / Messir	12
2.3 Software Engineering Environment	12
2.3.1 Eclipse	12
2.4 Software Engineering Workbench : DSL Development	13
2.4.1 Xtext	13
2.4.2 Xtend	13
2.5 Software Engineering Workbench : UI Development	14
2.5.1 Eclipse Modeling Framework	14
2.5.2 Sirius	14
2.5.3 Standard Widget Tooling	14
2.5.4 Xtext Web Editors	14
2.5.5 DSLForge	15
2.6 Software Engineering Workbench : Document Generation	15
2.6.1 Texlipse	15
2.6.2 Microsoft Excel	15
2.7 International computer science standards	16
2.7.1 Software Engineering Body of Knowledge	16
2.7.2 Computer Science Curricula 2013	16
2.8 Related Syllabus tools and methods	16
2.9 Related work on quality of DSLs	18
3 TESMA Requirements	21
3.1 Environment Model	21
3.1.1 Actors	21
3.2 Use Case Model	24
3.2.1 Usecase model: Create and Manage a Teaching Program	24

3.2.2	Usecase instance : Creating and Managing a Teaching Program . . .	25
3.3	Concept Model	28
3.3.1	Class-types for actors	28
3.3.2	Class types for educational institutions	31
3.3.3	Relations between the class types for the actors, and the educa- tional insitititons	33
3.3.4	Classtype ctCourse	33
3.3.5	Classtype ctTask	36
3.3.6	Classtype ctTest	36
3.3.7	classtype MeasuredVariable	36
3.3.8	Program, and Course Certification	37
3.4	Quality of DSLs and non-functional requirements	38
3.4.1	Quality	38
3.4.2	Dependability	40
3.4.3	Maintainability	41
4	TESMA Design, and Realisation	43
4.1	Architecture	43
4.2	Textual Editor	44
4.2.1	Description	44
4.2.2	Domain-Specific Language	46
4.2.3	Features	53
4.3	Graphical Editor	54
4.3.1	Design and Features	54
4.3.2	Technology	54
4.4	Documentation Generation	55
4.4.1	Generator	55
4.4.2	Document types	56
4.5	Quality, Dependability, and Maintainability	57
4.5.1	Quality	57
4.5.2	Dependabilty	60
4.5.3	Maintainability	61
5	Case Study	63
5.1	TESMA Program Specification tool	63
5.2	University of Luxembourg	63
5.2.1	Description	63
5.2.2	Specification	66
5.2.3	Institution director's role	66
5.3	Master in Information and Computer Sciences	67
5.3.1	Description	67
5.3.2	Specification	67
5.3.3	Certification	69
5.3.4	Actors	70
5.4	Software Engineering Environment	70
5.4.1	Description	70
5.4.2	Specification	71
5.4.3	Actors	75
5.5	Document Generation	76
5.6	Analysis of the quality of the DSL	77

5.7	Results	78
6	Limitations	79
7	Future Work	81
8	Conclusion	83
9	Appendix	89
9.1	Appendix A : TESMA Grammar	89
9.2	Model	89
9.2.1	ctInsitution	89
9.2.2	ctProgram	90
9.2.3	ctCourse	90
9.2.4	ctCourseOrganisation	91
9.2.5	ctPeriod	91
9.2.6	ctTask	92
9.2.7	ctArtefact	92
9.2.8	ctTest	92
9.2.9	ctCategory	93
9.2.10	ctCriteria	93
9.2.11	ctMeasuredVariable	93
9.2.12	ctGrades	93
9.2.13	ctActor	94
9.2.14	ctBoard	94
9.2.15	ctGroup	94
9.2.16	ctInstructor	94
9.2.17	ctPromotion	95
9.2.18	ctStudent	95
9.2.19	ctFieldCoverage	95
9.2.20	ctStandard	95
9.2.21	ctField	95
9.2.22	Auxiliary class types	96
9.2.23	Auxiliary class types	96
9.2.24	Auxiliary data types	96
9.3	Appendix B : TESMA - Generator	97
9.4	Appendix C : TESMA - Validator	107
9.4.1	Validator	107
9.5	Appendix D : TESMA - ICAIT 2016 Paper submission	115
9.6	Apendix E : TESMA Sample Grading Sheet	120
9.7	Apendix E : TESMA Sample Grading Table	121

List of Figures

3.1	TESMA summary use-case model	24
3.2	Usecase instance of specifying a teaching program	27
3.3	Class types for physical actors	28
3.4	Class types for logical actors	29
3.5	Class diagram for actors	30
3.6	Class types for educational institutions, and their teaching programs . . .	31
3.7	TESMA High-level class diagram	34
3.8	Class types related to a course	34
3.9	TESMA course specification metamodel	36
3.10	TESMA test, and task metamodel	37
3.11	TESMA evaluation metamodel	38
3.12	TESMA certification metamodel	39
3.13	Quality Requirements	39
4.1	TESMA software architecture	43
4.2	TESMA process overview	44
4.3	TESMA textual editor	45
4.4	General TESMA UML model	47
4.5	Course class type and it relations	50
4.6	Test class type and its relations	52
4.7	Documentation Generation - Report Structure	55
4.8	Documentation Generation - Report	56
5.1	TESMA specification of an institution	65
5.2	TESMA Program Specification - MICS	68
5.3	TESMA Course Specification - SEE	71
5.4	TESMA Task Specification - SEE tasks	74
5.5	TESMA Documentation Generator	76
9.1	Grading sheet sample	120

Declaration of Authorship

I, **JAHIC Benjamin**, declare that this thesis titled Requirements, Design and Implementation of a Teaching Specification, Management and Assessment tool, and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Date:

Signature:

Abstract

The definition and organisation of programs related to their courses of educational institutions is a very complex and exhaustive task. There is a demand for such a solution by the educational institutions, as they need a detailed program descriptions for students and instructors. This task gets even more complicated if these programs and courses needs to be certified according to some international learning standards.

At the moment, the availability of such methods or tools is very limited, except for some ad-hoc guidelines, which are use by some few universities, e.g. the Cornell University. Most of the institutions (e.g. University of Luxembourg) allows the professor to us their own methods for specifying their courses. Hence, most of the institutions are sharing similar problems, but using their own defined methods (e.g. naming conventions for its programs).

At an university, professors are working in various domains and using therefore their own methods for specifying their courses, which results often in an incomplete program and course description. Methods such as SWEBOK (Software Engineering Body of Knowledge) and CS2103 (Computer Science Curricula 2013) which are program certifications according to an international learning standard are almost not known and used. Thus, programs and courses from different institution cannot be compared, since there is no common structure and process for specifying them.

In this master thesis, we present TESMA (Tool for Educational Specification Management and Assessment of teaching programs), a tool based on a domain-specific language, which is dedicated to the teaching domain, for specifying, managing, and assessing programs. The Messir (Scientific Approach to Requirements Engineering) development method has been used for defining the concept and the requirements of the tool. Our research concentrates on the domain-specific language (DSL) in order to define requirements and improving the quality of the DSL's.

We focus on the development of an intuitive and maintainable domain-specific language, usable by people coming from different domains, e.g. software engineers, natural sciences, social sciences, linguistic, and so on. This thesis describes the requirements, the concepts, the realisation, and implementation of the tool, which are based on a domain-specific language of high quality for specifying programs. The quality of our DSL is assessed by a complete used cases related to the University of Luxembourg.

Acknowledgements

At the end of my thesis I would like to thank all those people who made this thesis possible and an unforgettable experience for me.

First of all, I would like to thank my supervisor, Prof. Dr. Nicolas Guelfi, for making this thesis possible, for supporting me during the thesis. The knowledge I obtained during our meetings and discussions were an unforgettable experience. Thank you for making it possible to visit the EclipseCon 2016 in Toulouse, which helped me to extend my knowledge in software development. Thank you for giving me the opportunity for continuing my studies in a phd program with you as my supervisor at the University of Luxembourg.

I would like to thank my advisor, Dr. Benoit Ries for guiding and supporting me during the thesis. You have set an example of excellence as a researcher, mentor, instructor, and role model. He offered his continuous advice and encouragement throughout the course of this thesis. Thank you, for all the work and time you invested for helping me during my thesis.

I would especially like to thank my amazing family for the mental and emotional support, and constant encouragement I have gotten during my studies. In particular, I would like to thank my parents, my brother, and my sister. I could not have done this without you.

I would like to thank my friends and classmates for the support during my studies, for listening and offering me advice. Thank you for all the debates, dinners, and game nights as well as editing advice, rides to the airport, and general help and friendship were all greatly appreciated.

Finally, I'm excited and motivated to continue with my phd with Prof. Dr. Nicolas Guelfi as my supervisor and Dr. Benoit Ries as my advisor. I'm glad to continue my work with you and gain more experience for my future life.

Chapter 1

Introduction

In this chapter, we will briefly describe our project, and research to you. We will start by the need for the development of program, and course specification tools. We analysed the state of the art, the problematic,, and propose a solution to the problem. We describe a small overview of the solution, and continue with our research topic, the quality of domain-specific languages. Finally, we will summarise the content of this thesis for guiding the reader through this thesis.

1.1 Problem Description

Every educational institution, like universities, have a need for detailed course and program descriptions. Currently, most universities are working manually on their program descriptions, which leads to some big problems. Due to the manual descriptions, program, and course descriptions are often incomplete and lead to misunderstandings. Work has to be redone multiple times, because data is missing inside the descriptions. It is exhaustive and expensive in terms of workload, if the program director needs to inform all the instructors about the missing parts in their description. These iterations of program and course descriptions need to be avoided. A program, a set of courses, needs to be clearly described for encouraging a student to inscribe for it. Moreover, the course syllabi, course content descriptions, have to be complete and intuitive for the program directors and students. Students want a clear course syllabus, which help them to follow the course. They need all information about the course organisation, course content description, course evaluation and so on. Expect of some guidelines, which we will present in the next chapter, we did not find any methods supported by a tool, which helps to describe a program or a course.

This lack of data in the program and course describes often comes from a lack of knowledge in how to structure these descriptions. The instructors often do not exactly know how to start or what to write inside the syllabus, which than leads to an incomplete and unorganised course syllabus. This complete description of programs and courses is an exhaustive job, where the concerned people need to invest a lot of time in thinking, structuring and writing the descriptions. There is a need for a method supported by a tool in order to work more effectively of facilitating these specifications for the teaching staff.

The lack of data is not only a problem for program directors and students, it may cause problems at the course registration office of an institution. These offices need a complete description of the course for registering the course at the university's database, but the secretaries often complains about the incomplete and erroneous course registrations form.

These forms often need to be redone and resent to the teacher, who wants to register the course. Again we start iterating the process for obtaining a complete course registration form.

Another problem is that professors spend a lot of time in preparing introductory presentations, course overviews, evaluation sheets, and grading tables. This job is almost identical for every course, but demands a very high workload for the professors. There is a need for automatising the process in generating these documents out of the specification of a course in order to reduce the exhaustive jobs.

Students often complain about the course content descriptions, course structure and evaluation. Sometimes the course descriptions don't have detailed information about the content or of the student's evaluation. Students would like to be informed about the course structure and course content before starting the course. There is a need for structured documentation about the programs and courses for informing the incoming students. Additionally, students are also interested in the outputs of a course, which are often written by the instructors themselves. These outputs can be topics of an international standard such as SWEBOK and CS2013, which are standards in the computer science and covering different knowledge areas in different domains. Standardising the outputs of a course will provide homogeneous program and course descriptions. Homogeneous descriptions, containing course descriptions, lecture information, evaluation descriptions, are structures, simple and easy-to-understand, since they all have the same structure. Students would prefer such documents in order to better compare different programs and courses. It facilitates the future orientation of their studies.

In summary, we can conclude that there is a high demand for a method supported by a tool for specifying programs and course descriptions. Since, we concentrate on educational institutions, we want to satisfy a large group of people, who will use our tool for specifying their programs. These people are not all from the computer science domain, which leads to some more problems. We propose therefore to develop a textual domain-specific language, which can be used by a large group of people for specifying their programs. The domain-specific language needs to be of high quality, since it will be used by a large group of people from different domains. Our goal is to develop a usable domain-specific language for educational institutions. We want to reduce time, costs and workload and satisfy the user of the domain-specific language like program directors, dean, secretary, instructors and so on.

1.2 Solution Overview

We have seen a lot of problems and needs so far, which need to be solved. We propose a tool and a method for designing, managing and assessing courses as solution for the problem. Our tool, named TESMA, covers the problems described above and proposes a domain specific language, which allows the users to describe their programs and courses. TESMA stands for Teaching Educational Specification Management Assessment. The tool is supported by a textual domain-specific language with a graphical editor and a document generation. The textual editor's main feature is to specify precisely programs and courses. The user is able to describe the course content, specify the lectures using periods and their evaluations. He is able to specify the tasks and tests, which need to be done during the course runtime. After having specified the programs and courses, the specifier has the possibility to generate several documents, which contain the complete descriptions about the institutions and their infrastructure, about programs and their program structure as well as the course syllabi with the introductory presentation slides,

evaluation sheets, grading tables for the different lectures, grading sheets for the students and course registration sheets for the universities office. The following stakeholder are the users of our tool : professors, course directors, secretaries and students.

We focus on designing our generator producing a report, which contains a complete program description, which satisfy all the concerned parties. Therefore, we will specify several requirements for our tool, which help us to design a method and a domain-specific language for specifying programs. Our method is supported by a tool for designing, managing and assessing a course. Additionally, it should improve the communication between the concerned parties.

The TESMA domain specific language (DSL) is designed to be intuitive and usable by non-computer scientists. The DSL uses natural language for the naming conventions. It is easy to understand and usable for non-computer experts. Additionally, the textual domain-specific language can be modified using a graphical editor, which facilitates the specification of courses. Users, who are not feeling comfortable with domain-specific languages, may use the graphical editor for specifying and update the program descriptions. The main objectives of our tool is to reduce the costs and time needed for specifying courses, to improve the communications between the stakeholders of the tool, to improve and complete the program and course descriptions.

We will automatise the process of creating documents like teaching material, grading sheets, grading tables, introductory presentations and course syllabi. We will generate homogenous documents of every program and course of a institution. The advantage of these generated documents is that students may easily compare different compares. Each time, the students read a course syllabi, they do not have to adapt to the instructor's personal specification, since the generated documents are homogenous and are based on a unique style proposed by us.

Finally, the specifier may use international learning standards, like SWEBOK and CS2013 for the computers science are, which can be specified for describing course outputs. The students will have a precise topic coverage of a standard, which can be used to compare with other programs. Thus, certifying a course to an international standard would facilitate the inscription in other institutions. These standards can be chosen and extended by the institution.

1.3 Research

A part of this thesis is dedicated to the development of a domain-specific language for specifying program and course descriptions. This domain-specific language needs to be used by a large group of people coming from different domains. Some of them are non-computer experts. For satisfying and motivating such a large group of people for using the tool and generating the usable documents for the different stakeholders. We have to develop a domain-specific language of good quality. Many research has been done in the domain of the quality of domain-specific languages. For developing a domain-specific language of good quality, the software engineer has to ask himself the following question "What requirements do I need to fulfil for developing a DSL of high quality?". It is important to know what a domain-specific language of good quality is. Our research starts with answering the following question :

How do we define the requirements for the quality of educational program description domain-specific languages?

The answer of this question defines the requirements for obtaining a domain-specific language of high quality. Defining the quality of a domain-specific language is a difficult task, since quality may have different interpretation depending on the person. Quality is a vague requirement, which can be subdivided into more specific requirements. These requirements can be grouped into different categories, which define the quality of a domain-specific language. Especially, since we are developing a domain-specific language for a number of educational institutions, we want to satisfy the users by providing a domain-specific language satisfying all these requirements for improving the quality. Additionally, as the users of the DSL are not all computer experts, we need to define a large set of quality requirements. The quality requirements of the domain-specific language will be defined, analysed and evaluated in this thesis.

In general, these stakeholders should be satisfied with our domain-specific language and the requirements should be designed for satisfying their needs:

- The *Domain-experts* are the university staff of the educational institutions using the domain-specific language for example for specifying programs and courses in TESMA.
- The *Software engineers* are developing and maintaining domain-specific languages and its grammar.

It is very important to have a DSL of high quality since we want to motivate the stakeholders to use our tool. The reason why we split the requirements for two different kind of users is to better define their needs. A maintainer has slightly other needs than a specifier who is using the editors for specifying their programs.

In this thesis, we propose TESMA, a program and course specification tool based on a domain-specific language, for answering this question. Our design decision and non functional requirements led us to the definition of a set of requirements, which can be used for creating a domain-specific language of high quality, which can be used at educational institution by a large group of people.

1.4 Thesis Overview

This thesis presents a method supported by a tool for specifying programs and courses. The tool allows the specifier of the course to generate documents like teaching material, grading tables, grading sheets, introductory slides and so on. The tool allows the instructors to maintain their course descriptions. The tool, which is based on a textual domain-specific language, covers a part of the current problems related to program and courses specification, which we discussed at the beginning of the chapter. The problems will be answered during this thesis. We will show the concept of TESMA and discuss the requirements and the realisation. We will propose a small example of a program specification at the University of Luxembourg. Finally, we will present the limitations and the future work of our tool.

Chapter 2

Background

In this chapter, we will present the technologies used for developing TESMA. We briefly describe how they are used, and motivate our choice. Secondly, we describe a number of existing open-source, and proprietary tools. Therefore, we compare the tools with the TESMA tool suite. Finally, we present the current research status on the quality of domain-specific languages, and our contribution, and thought on DSLs of high quality.

2.1 Domain specific languages

Van Deursen, Klint and Visser present in their paper [1] the purpose of domain-specific languages (abbr. DSL). According to them, a domain-specific language, also called little languages, is a particular language of a specific domain. The language describes techniques, which can be used in the domain to solve an issue. They propose the following definition for a domain-specific language :

"A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain"

There exists a number of DSL's like HTML, SQL, which are languages used in the computers science domain for web page illustrations resp. for Database Management, another daily life example would be cutting meat with a knife.

In their paper, they talk about advantages and disadvantages of using a DSL. According to them, DSLs describe solutions at a level of abstraction of the problem domain. This allows the domain experts to understand, validate and modify themselves the DSL programs. They think that DSLs enhance productivity, reliability, maintainability, testability and portability. With the help of validation rules DSLs can be checked and errors can be highlighted, so that they can be corrected by the users. These rules help also to optimise the instance of a DSL by validating the model.

However, according to the authors there are some disadvantages. The developments of a DSL is time consuming. There are high costs in engineering a DSL and teaching the users for using it. The lifecycle of a DSL can be short if we are working in a very specific domain, where often changes occur in the domain problem. Additionally, there is some potential loss of efficiency if we compare it to traditional hand-coded software, where we can optimise the execution time or change the core implementation.

In this thesis, we agree with Klint et al. However, we think that the requirements needs to be extended. We split domain-specific languages into two categories. We distinguish

textual DSL's (tDSL) with Graphical DSL's (gDSL). A tDSL can be used to describe operations and sequences. tDSL's can also be used to combine several expressions in that DSL. gDSL's are used for graphical illustrations and are better to describe relationships or if the design, which is defined by the user of the DSL, needs to be analysed or changed.

An advantage of domain-specific language is that the language can even be used by people who are not experts of the domain. If we provide a simple DSL or usable gDSL, we make the DSL usable by a large group of people (even non computer experts). DSL's can be developed for Domain experts, who use them for understanding, validating or modifying a solution to a problem. Additionally, daily users use the DSL to facilitate some task and save time for time exhausting tasks. By using a textual and graphical DSL in combination, we cover a big part of our problem, since the DSL can be used by non computer scientist and covers all the requirements we need. We use tools like Xtext [3] and Sirius [4], for developing our tDSL resp. gDSL.

2.2 Excalibur / Messir

Messir is a software engineering approach support by the Excalibur tool, for specifying software systems. The tool is based on a flexible requirements description language. Excalibur proposes the following main features : an extended use case modelling approach, a method for scientific requirements engineering and the generation of specification report containing the a software design and requirement specification. The tool helps the software engineer to specify the actors, use case models, environmental, concept and operational model using a domain-specific language with a graphical editor. Additionally, the software engineer may insert additional textual explication for generating the specification document. We used Excalibur for specifying the concept model of TESMA and proposing some use-case models, which we will be presented in this thesis.

2.3 Software Engineering Environment

2.3.1 Eclipse

Eclipse [2] is an integrated development environment (IDE) which is based on an extensible plug-in system for customising the environment. The Eclipse contains a base workspace, where projects and metadata of the working session is stored. Eclipse is mostly written in Java, but it can also be used to write in many other languages like C/C++ or php by adding the corresponding plugin.

In our tool, we used eclipse for developing the TESMA tool suite because plugins like Xtext, Sirius, Texlipse and SVN are available for developing creating our own Eclipse-plugin. The different plugin tools will be described in the next section. The big advantage of eclipse is the flexibility of configuring a environment for your own needs, the regular updates and since we are mostly working with Java Eclipse runs on every existing platform. Additionally, for maintenance, eclipse is very powerful in case we want to extend our project and add additional features in the future. Therefore, we want to develop a maintainable tool for further updated of TESMA. A disadvantage of eclipse is a complicated usage, which is the reason why the environment needs to be set up by computer experts.

2.4 Software Engineering Workbench : DSL Development

2.4.1 Xtext

Xtext [3] is an open-source framework for developing programming languages and domain specific languages (DSL's). Xtext is available as Eclipse plugin and can be easily integrated inside the environment. It is a language infrastructure, where you can define a language as you want. Xtext does not only generate a parser, but also a class model for an abstract syntax tree. In Eclipse, Xtext offers a text editor as well as a infrastructure, which is needed to develop a domain specific language. The developer is able to create a model to work with and to write a concrete syntax for.

The big advantage is Xtext runs on a Java Virtual Machine, which is the reason why we are able to execute our DSL on every existing platform. After having specified your grammar, Xtext generates a model out of the grammar, which can be used for graphical illustrations or writing it in a DSL language. Sirius can be used to illustrate graphically the domain specific language. Therefore, Xtext is very powerful, since for all changes done in the grammar, the tool regenerates a new model, which can be directly reused inside the graphical editor. Additionally, Xtext has released a not fully developed Web Interface, which can be used for specifying your courses. However, we need to wait for further updates, which will arrive soon. The web interface helps us to provide a convenient and user-friendly tool for non-computer experts. They simply have to use the web interface in order to specify their courses.

Xtext offers features such as validations, templates and generations. These features facilitates the specification of the programs and the course. Validation rules checks if the specification written in TESMA is syntactically correct. Templates help the user to write effectively the syntactical correct specifications and to quickly understand the structure of TESMA.

2.4.2 Xtend

Xtend [3] is a statically-typed programming language based on Java. Xtend provides many advantages over Java in order for writing effectively and efficiently source code and large text generations. According to the developers of Xtend, unlike other Java Virtual Machine languages, Xtend has zero interoperability issues with Java. Everything written with Xtend interacts exactly as expected with Java. In TESMA, we use Xtend for simplifying code generation, by writing a lot of pure text, where data needs to be inserted. In Java, this task would be exhaustive, which is the reason why we have chosen Xtend. The user may add some java code inside this pure text in order to add some data specified by the user. The language supports pure text handling and source code handling inside text. The source code needs to be written in between brackets. Xtend improves the declaration of variables. The variables recognise automatically their type after inspecting the value with whom they are initialised. Additionally, the user does not have to define new types each time, he wants to create a new variable. Some syntactical improvements, like the loss of the semicolon are considered in Xtend. In general, we are creating a lot of Latex code, for our documentation generation, where we will mainly use Xtend to write the report.

2.5 Software Engineering Workbench : UI Development

2.5.1 Eclipse Modeling Framework

The Eclipse Modeling Framework [4] is an open-source Java framework for modelling. It eases the generation of code out of a specified model. EMF is used for building tools based on some data models. The model is specified in an XML file. That XML file is taken as input for generating Java classes, which can be used by the programmer for implementing a solution to a problem.

In this thesis, we do work indirectly with EMF. Xtext and Sirius are based on an EMF model, which is used to generate the Java classes for implementing the Validator rules or the Generation code. The EMF model is generated out of the grammar defined in the Xtext plugin. The model is used then by Sirius to be represented and modified graphically.

2.5.2 Sirius

Sirius [4] is an open-source software project developed by the Eclipse Foundation. It is used for developing a graphical modelling workbench. Sirius is based on the Eclipse Modelling Framework and the Graphical Modelling Framework. The models may be specified using graphical, table or tree editors. Sirius is using a set of rules, which can be defined by the user. These rules describe, for example, the different components, which are shapes, edges etc, in your graph. These components can be configured manually. Sirius provides the ability to customise existing environments by specialisation and extension. After creating your model, you are still able to dynamically change or adopt your models.

Sirius provides a very effective tool, for creating a graphical view out of a domain specific language. It covers our needs, and can be dynamically adopted to grammar. Features like faster accesses and templates can be implemented manually inside Sirius.

2.5.3 Standard Widget Tooling

Standard Widget Tooling (SWT) [5] is an open source graphical widget toolkit based on a Java Platform. Depending on the operating system, on which SWT is used, it provides efficient, portable access to the user-interface facilities. SWT is designed to be a high performance graphical user interface, however it does not fit our needs, since it is not compatible with Xtext like Sirius. Maintenance would be harder to ensure, which is very important for TESMA. Since the tool needs to be adoptable and very flexible. We want to continuously add new features and change the grammar to the user's needs.

2.5.4 Xtext Web Editors

Since, the new release of Xtext 2.10, Xtext has a feature called Xtext Web Editors, which allows us to create out of our domain specific language a web interface, where you can write in your domain specific language. The web interface is currently not fully developed, since it does not support multiple DSL files. Multiple files handling is considered to be in a future release. However, a web interface is very important for us, since it facilitates the usage of TESMA for non computer experts. A simple web interface is easier to install and to maintain than Eclipse installations on every single personal computer. We would bypass the disadvantage of the environmental set up on every single computer, by providing this web application.

2.5.5 DSLForge

DSL Forge [6] is a tool similar to the Xtext Web Editor used for representing the textual editor with the domain-specific language in the web browser. Concerning the web interface, it is more slightly powerful than Xtext, since it offer more features, like package explorers and a file system. Additionally, the validations rules and the generation handler work like a charm inside the web application. In other words, the web user interface contains a whole snapshot of the eclipse environment in which the domain expert may use his DSL. However it does not offer all features of the eclipse environment. Adding plugins other plugins to the web interface, is currently not possible. A big disadvantage of DSL Forge is that we are restricted in using either web interface or a runtime eclipse, which is very bad and does not allow us to use more plugins like Sirius and Texlipse. Additionally, we want to make the tool usable by a large group of people, there fore we want to offer both possibilities in order to satisfy all users, those who prefer Eclipse, and those who prefer a web interface. Currently, the Xtext web Editors allows both possibilities (the web UI and the Eclipse tool), which is the better alternative for some future release. We are waiting for some updates on the Xtext Web Interface before using it.

2.6 Software Engineering Workbench: Document Generation

2.6.1 Texlipse

Texlipse [7] is a Eclipse plugin project, which we use to write our documents generated by the TESMA tool. It supports a PDF generation out of a unique tex file. LaTeX itself is a document preparation system. It is often used for scientific documents but it can be used for almost any form of publishing. Mainly, the writer uses plain text for writing their documents. It is at that moment not formatted text like in Microsoft Word. The writer uses tags and commands to define, what Latex should do with the plaintext. The user is able to define the general structure of a document (such as article, book, and letter), to stylise text throughout a document. We use Latex mainly for writing the report, since we do not have to worry much about the styling. Once the style is set up, we simple have to write our plain text and the document will be automatically generated out of the domain specific language. The user's do not have to be latex experts since the system generates everything automatically.

2.6.2 Microsoft Excel

Microsoft Excel is a spreadsheet, which was developed by Microsoft for Windows, Mac OS X, Android and iOS. Inside Excel, we have many features like calculation, graphing tools, pivot tables, and a macro programming language. It is a very easy-to-use tool and can be handled by any person, even non-computer experts. In educational institutions, excel is often used to generate grading tables and evaluation sheets. The user creates a template and prepares all formulas needed to calculate the grades, after that he only needs to fill in his grades and the final grade will be calculated automatically. The teachers and professors get an overview of the whole class, and provides his own tables for each of their classes. In our tool, we use the Apache.poi library [8], which helps us to generate the Microsoft Excel documents in Java. The library provides all features needed to generate and handle Excel sheets in Java.

The big advantage of Excel is the simplicity to use it. This is the reason, why we

have chosen Excel to create our spreadsheets for the evaluation. TESMA generates out of the specification the Grading tables and evaluation sheets needed and provides Excel files to the user. These files can later on be used by the professors for setting up the grades and having all information about their students.

2.7 International computer science standards

2.7.1 Software Engineering Body of Knowledge

The Guide to the Software Engineering Body of Knowledge (SWEBOK) [9] describes generally accepted knowledge about software engineering. It contains 15 knowledge areas, which summarise all basic concepts and include a reference list pointing to more detailed information in each area. SWEBOK itself is a guide, which also has also gained international recognition as ISO Technical Report 19759. SWEBOK is very easy to understand due to his structure. Each knowledge area of the computer science domain, has some topic in that specific field, which itself contains some subtopics. An example would be the knowledge area: Software Design \Rightarrow Software Design Fundamentals \Rightarrow Software Design Process. The big advantage of such a standard is that the student easily understand the key concept of the SWEBOK knowledge areas and they will exactly know what it covers by following a specific course or a program. Additionally, the professors could compare their course with courses from other institutions. This allows the professors to rate their courses according to these comparisons. SWEBOK can so be used as motivation for updating and redesigning a course.

2.7.2 Computer Science Curricula 2013

Another Guide is the Computer Science Curricula 2013 (CS2013) [10], which is a more detailed guide and describes like the SWEBOK guide generally accepted knowledge about software engineering. It covers 18 knowledge areas and has also different topics inside the different areas. However, the subtopics are more detailed than the SWEBOK criteria. The advantage of a more detailed international standard is that the course can be better rated internationally by comparing it to other courses. However, due to the precise information, students sometimes would not understand everything what is meant by CS2013. In that case it makes sense to combine SWEBOK and CS2013 for defining course standards. In that case, we would cover all the user's needs and provide the maximal amount of understandable information.

2.8 Related Syllabus tools and methods

There have been done a lot of research in the area of educational program and course specification. Most work has been done for high schools, where software tools have been developed for the US market for the K-12 classes ¹. A number of universities developed some sort of guidelines for writing a course syllabus. These guidelines are sometimes also called curriculum tools. However, we are developing a method and a software tool, for specifying educational programs and courses using a domain specific language .

In the following sections, we will present the different guidelines and software tools to show the importance and to position our tool according to the work already done in the world:

¹classes of primary and secondary education

Many researchers and instructors provided information about teaching and educational documents. They analysed the importance of course syllabi and showed that there is currently a need for creating specifications of programs and courses for educational institutions. These institutions need inform the concerned people, such as students, instructors and secretaries with a maximal amount of information of the programs and courses. In general, if a instructor cares about his students and teaching, he needs to find a way for improving his abilities to help the students develop their intellectual skills. [29]

Parkes talks in his article about the purpose of course syllabi [30]. Course Syllabi may have different definitions for a institution. According to PARKES a course syllabi can be seen as a contract, permanent record or learning tool. A contract is simply a contract between teachers and student, by specifying several policies, such as grading policies, clear and accurate course calendar, attendance policy. The permanent record is a information sheet about the course, such as course descriptions, instructors, credits and hours taught for example. Finally, it should also be considered as a learning tool, which helps the student to understand the course contents, and helps him for passing the course.

Keller showed in his study [31] that agreements in between instructors and students differ from each other. The results of his study indicate that faculty and students differ in their opinion on the importance of several syllabi components. In our tool, we considered a similar impact and generated different documents for students, instructors and institutions

Many guidelines have been presented by several famous universities like the University of Washington and the Cornell University etc. [22] [23] [25]. However the guidelines are still kept general and are not going much into detail, which is a difference to our approach. Their ideas were mainly to provide a natural language description of different tasks to perform for arriving at a course syllabus. In our approach, we provided a process, which is adopted to the TESMA tool and can be used by instructor's from different domains. It focuses on the universities domain, but it could still be used in K-12 classes.

Slattery talks in his article [24] about the design of a syllabus. According to them, the first impression, when looking at the course syllabus counts mostly. For them, the design of a syllabus should be attractive without being distracting, and should be consistent with the tone of the course. In general, we also think that a course syllabus must be attractive, but often instructors do not know how to write an attractive syllabus. Our tool, creates a unique design for all the courses, which makes it very easy to understand, since you have to concentrate on one design, what the instructors wants to present about the course. The student gets more comfortable and the tool aims to help the professors to write descriptions of courses and programs of good quality.

PDF Syllabus Builder [17] is an open-source tool for online instructors, course developer and instructional designers. The users are guided step-by-step in populating six sections: information, course objectives, course resources, course activities, policies and grades. The tool uses a template PDF form where the user can fill out several fields to write the final course syllabus. TESMA covers many additional features. The difference to the PDF Syllabus Builder is that we use a domain specific language in the Eclipse environment for writing the course syllabus and we generate several documents like course syllabus, inscriptions formulars, grading tables and grading sheets, which are not considered by the PDF Syllabus Builder.

Build your own curriculum [18] and JumpRope [19] are a web-application for Collaborative, Consistent Curriculum, Instruction and Assessment. The users are able to design their courses, manage the classrooms, add certain international standards depending on

the course and describe their courses. These descriptions can be printed out and used then. The tools are mostly developed for K-12 classes, but is very similar to our approach. Many tools, also provide the feature of class managements and students grading, which is also considered by us. There have been some tools developed around, which are similar to ours but the main difference is that we want to cover the universities and try to make a tool, which can be used by professors from several domains. The tool additionally need to be maintainable and flexible according to the universities needs, which is not always the case for such web application. Since the tools are often developed for K-12 classes, mostly teachers who have performed some educational studies are teaching the students, so they all are from one domain. In our case, at the universities professors are not always from one domain like the non-computer scientist domain. To be general we have developed a domain specific language (or our user interface), which mostly uses natural language to specify their courses. In addition, our tool generates a full document with all the description needed for the institution, but not only. Our tool supports the feature of printing out some introductory presentation, which can be used in the first lecture to describe the course content orally.

Interesting tools like Integrated School Management Software [20] from slightly other domains but has some intersection with our tool. This tool is a educational portal, which covers course descriptions and grading. The tool can be used by parents to check their children grading. Again this tools, covers the K-12 classes, but has some intersection to the universities domain. Another tool, like PowerVista [21]. RollCall contains processing for tracking degree granting programs, accreditation statistics, relationships with outside sponsoring organisations, invoicing, student progress alerts and marketing to prospective students. The tool is developed for non-K-12 users.

2.9 Related work on quality of DSLs

In this section, we describe the work done in the domain of quality of DSLs. In the past, a lot of work has been done for defining the quality requirements for domain specific languages. A number of people present methods for developing a DSL of good quality and a few others present some quality requirements.

However, we think that there is a need of regrouping the quality the requirements. Quality is a very general requirement, which is often interpreted differently from user to user. We want to regroup already existing quality sub-requirements with other requirements to better explain the quality of domain-specific languages.

Which requirements are used for defining the quality of domain-specific languages?

Kolovos et al. present the requirements for domain-specific languages in their paper [11]. The authors define a number of core requirements for DSLs, like Conformity, Orthogonality, Supportability, Integrability, Longevity, Simplicity and Quality. We will use a part of the requirements for improving the quality of DSLs. According to them, these requirements are :

- The *longevity* requirements describes that a domain specific language should persist during a sufficiently long period of time.
- The *Simplicity* requirements ensures that a DSL should be as comprehensible and user-friendly as possible for learning and using the language.

- The *Quality* ensures that the DSL should provide a general mechanism for building a well-designed system. According to them, this may include reliability, security, safety etc.
- The *Scalability* guarantees the easiness for developing large-scale descriptions.
- The *Usability* describes the requirements such as economy, accessibility, and understandability of the domain specific language.

We agree with their requirements and we use them for describing the TESMA language. However, we think there are some more requirements, which can be used to describe the quality. We think that quality as a requirement does not describe sufficiently the needs. It makes sense to divide the quality requirement in different smaller requirements. Quality can have different meaning for different users, which made us reorganise the requirements listed above.

Van Deursen [12] talks in his article about the maintenance of domain-specific languages. The author presents the benefits of using domain-specific languages for guaranteeing maintainability, like domain-experts can easily understand, validate and modify the software by adapting the specification written in the DSL. (Usability). It is easier to modify and understand the impact of the modification in a DSL. (Simplicity). The domain knowledge is often explicitly available to everyone and so easier applicable for specifying in the DSL. The author listed a set of "maintainability factors", which will reduce the costs for modification inside the DSL. These factors are:

- Ease of expressing anticipated modifications
- Small development costs per application
- Small code size
- code readability
- system modularity
- locality of changes

From the list above we see that there are few factors, which improve the quality of the DSL and by that improving the maintainability e.g .system modularity, code readability, locality of changes etc. For TESMA we considered maintainability as a very important requirements, which we want to ensure. Having a maintainable DSL improves automatically the quality of the DSL. We agree with Van Deursen idea and support the fact that a DSL of high quality needs to be maintainable. While designing and developing TESMA, we had similar thoughts in terms of maintainability of the DSL. In this master thesis, we will discuss, how maintainability affects the quality of domain specific languages in TESMA.

Barisic et al. presented a lot of articles on the quality and usability of domain-specific languages. In paper [13], the authors presents additional requirements for defining the quality of DSL. The authors claim that increasing the quality of the domain specific languages, means that we decrease the need for maintenance. This means that we reduce modifications and updates on the domain-specific language. They propose the following requirements for the quality of domain specific languages :

- Quality in Use
 - Usability
 - Flexibility
 - Safety
- External and Internal Quality (
 - Functionality
 - Reliability
 - Operability
 - Efficiency
 - Maintainability
 - Portability

For defining the requirements, the authors used the Software ISO Quality Standard : 25010.3. Barisic stated that the usability is the most important attribute for having a DSL of high quality, since it describes the following sub-requirements: Effectiveness (i.e. determines the accuracy of finishing a sentence in the DSL), Efficiency (i.e. determines the levels of effectiveness is achieved with respect to physical effort, time or financial costs), Satisfaction (i.e. describes the freedom from inconvenience and positive attitude toward the use of the language) and Accessibility (i.e. the complexity and memorability of the keywords inside the domain specific language). Additionally, since the software industry seem to not invest much in the usability evaluation of domain specific languages, the authors present an evaluation of the usability of domain specific languages by the same authors in [15]. These 4 sub-requirements are used to evaluate the Quality in Use [14] of DSL in a real context. In this thesis, we followed similar requirements and tried to define the usability of TESMA. However we think that the external requirements might be also interesting for having a DSL of good quality, since it does not only depend on the usage. The International Organization for Standardization (ISO) help us to understand the quality of software. In general, we agree with the authors on the usability requirements, since the user evaluates mostly, if the DSL is of good quality. However, there are more requirements, which needs to be fulfilled to have a DSL of good quality.

In this master thesis, we reused a part of the requirements and tried to restructure it with respect to the ISO standard. We think that the usability is not the most important requirement. The usability requirement is highly dependant on the user of the domain specific languages, which makes it very hard to guarantee usability for a very high set of users. However by combining several requirements, it improves the quality and satisfies a larger set of users of the domain-specific language. We regrouped the quality requirements in 2 main groups. One part of the requirements are for the domain-experts, who are using the domain-specific language for specifications and the other part is for the software engineers, who have created or who will maintain the domain-specific language. We think that in order to have a DSL of high quality, we need to ensure these requirements. These requirements are also important since we generate documents with our DSL. For sure, if the DSL is not flexible and simple enough, the user will not be able to generate documents of correct content. We do not go into deep detail in this section, since we will discuss the quality of our DSL in chapter 3.

Chapter 3

TESMA Requirements

In this chapter, we present the requirements for TESMA. We start by presenting the actors. We describe them, and we present some use cases. The use cases describe the main idea of specifying a teaching program, and we illustrate this with a small use case instance. Additionally, we illustrate the concept model, which contains all class types, and data types used for developing TESMA. At the end, a short discussion will be performed over the document generator, i.e. who generates the specification and teaching material.

3.1 Environment Model

In this section, we describe the actors ¹ and their requirements. These actors have different requirements and needs. We tried to generalise their requirements, such that it can be applied by a large group of the corresponding stakeholder. We analysed the current situation at the university of Luxembourg. We tried to generalise the requirements such that they are applicable on a larger set of universities. We first start by describing the actors, which are concerned for the Teaching Program Specifications.

3.1.1 Actors

Program Specification and Management is needed in almost every educational institution. These descriptions affect different people, who have of course different needs. We analysed the needs and developed TESMA according to them in order to reduce the timing costs for writing program specifications.

institution director's requirements

The institution director is the representative of the institution. In universities, the director is often called the dean. He is creating and validating the institution's descriptions and program proposals. Additionally, he validates new program proposals, new course proposals and modifications done on already existing programs or courses. The institution director has a general overview of the whole institution and validates the work done by his employees (e.g. program director, instructors). These employees always need to pass through the dean's final decision if they want to contribute inside the institution's teaching program.

The institution director's requirements are :

¹The users of the TESMA tool suite

- managing and specifying the institution's descriptions
- managing the institution's program proposals
- generating a general description of the institution and the proposed programs in printed format
- accepting, validating, and creating new programs
- accepting, validating and creating new courses inside a program
- accepting, validating and updating the program and course according to the modifications

Program director's requirements

The program director is the representative of his programs. He is specifying the program and accepting the courses to be taught in. He is discussing the program proposals with the quality manager and the institution director in order to validate the program at the highest level.

The program director's requirements are :

- specifying and managing his programs
- generating a detailed description of the program and the proposed courses inside in printed format
- generating a program certification sheet according to an international or institution's standard
- accepting and validating new courses inside the program
- accepting and validating course modifications inside the program

Instructors's requirements

The instructor is the head of a course. He is specifying and teaching the course with a teaching team. The teaching team can have multiple members. In general, the teaching team is organising the course and writing the specification of the course. They are organising the timing slots for the lectures and the tasks done during the teaching term. Additionally, they need some documents for teaching the course. These documents are listed below and needs to be generated by the TESMA system.

The instructor's requirements are :

- specifying and managing his courses and their descriptions
- specifying and managing the lectures
- specifying the teaching team's tasks
- specifying the students tasks
- specifying the course evaluation
- evaluating and grading the student's following his courses
- generating the course evaluation sheets and course grading tables in Excel files

- generating course introduction slides out of the course specification
- generating a course registration sheet for the administration office
- generating course description, organisation and evaluation information sheets for the student's and for the program director

Student's requirements

The students are enrolled inside a program and taking different courses inside the program. The students want to be informed about the courses. Course material, course content organisation and course evaluation are the 3 most important data needed for the students. The students organise themselves by studying the course content and the course material. Additionally, students want to be able to prepare the evaluations by using the evaluation descriptions. The students often ask for the content of the evaluation and the material to study, which can be communicated by such description sheets.

The student's requirements are:

- course material (e.g. introductory presentations, slides and books)
- lecture organisation sheets
- course content description
- description of the evaluations

Secretary's requirements

The secretary is responsible for the program and course registrations in an institution. After being validated by the institution director, the secretary confirms and inserts the data inside the university's database. Therefore, the secretary needs registrations formulars, which contains the required field for registering the course.

The secretary's requirements are:

- Program and course registration forms for administrative registration inside an institution

Quality Officer's requirements

The quality officer is responsible for the course certification according to a standard chosen by the institution. The quality officer checks whether the program and course contents are adequate to the proposed difficulty level of the study. In general the quality officer is an optional actor, but it is recommended in order to get provide a study program of higher quality. He completes the intermediate step, where each new program or course submission should pass through before being validated by the institution director. Additionally, the quality officer can contact the international standard committees which evaluate the courses of their domain and assign to them a international learning standard.

The quality officer's requirements are:

- generating a certification sheet according the institution's standards
- Program and course specifications in PDF format for assigning them to an international learning standard

3.2 Use Case Model

3.2.1 Usecase model: Create and Manage a Teaching Program

Using the Messir method, we illustrated in Figure 3.1 the TESMA high-level summary use-case model of managing a teaching program. We define a program as set of courses. The use-case model shows the roles of the different actors, which we will describe in this section.

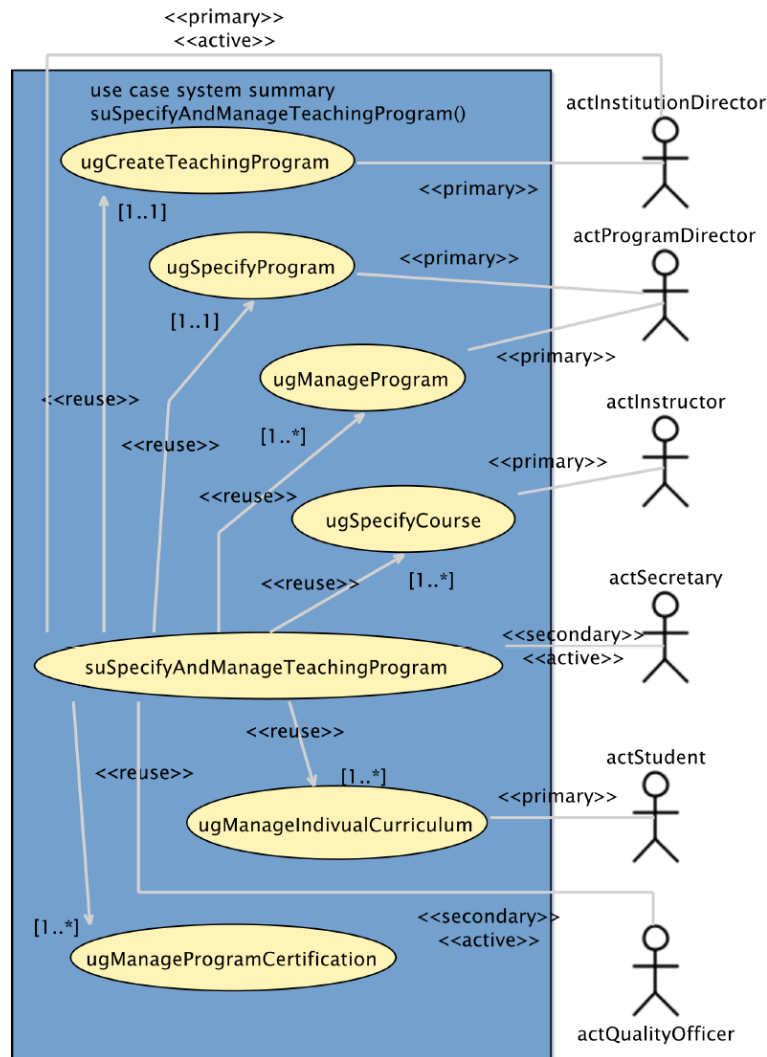


Figure 3.1: TESMA summary use-case model

As seen in the requirement section, the actors of TESMA have different needs. TESMA gives different tasks to the users in order to specify a Teaching Program. The use case model illustrates the tasks performed by the actors. We describe the general idea of how to create and manage a teaching program according to the TESMA methodology. The tasks are given by the following user-goals (ug) :

- **ugCreateTeachingProgram** : The institution director creates/hosts the institution, where he creates a new teaching program proposed by a program director. In general, he is the validator of a new program proposed by some program director or by himself.

- `ugSpecifyProgram` : The study director specifies the program and organises the program contents. he is deciding which courses are given in the program. Additionally, the program director is organising the lectures given per term.
- `ugManageProgram` : The study director manages the program during the runtime. For that he is making updates or changes in the program contents. Additionally, he is discusses the program descriptions, which the quality manager in order to certify the course.
- `ugSpecifyCourse` : The Instructor proposes and specifies a new course. In general, he proposes a new course to the study director, who creates it inside the program. The instructors specified the course content, the lectures, the course material, the evaluations and the teaching team.
- `ugManageIndividualCurriculum` : The students are managing their courses. The students see the information about the lectures given in a course and how they are evaluated. They are able to prepare their work during the lecture, by following the course specification.
- `ugManageProgramCertification` : The quality officer is contacted by the institution director, program director or by the instructor in order to certify their specifications.

The user goal summarises the TESMA specification process. In order to specify a teaching program efficiently, we divided the specification into 3 main steps :

1. Specification
2. Management
3. Certification

The actors are performing one or more of these 3 steps in order to produce a program specification and to define their courses. We will continue by specifying a use case instance for the specification of a teaching program by taking the use case model and the requirements of the different actors into considerations.

3.2.2 Usecase instance : Creating and Managing a Teaching Program

In this section, we present a high-level scenario for specifying a complete Teaching Program. In Figure 3.2, we represent the specification in a sequence diagram containing all steps for specifying a Teaching Program. We created the usecase instance with the consideration of the requirements of the different actors specified before. The sequence diagram presents a high level overview of the tasks performed by the actors. Before starting the actual specification, the system administrator creates and initialises the TESMA environment. The system administrator creates the institution and their sub-institutions with all the actors concerned by the specification. The actors are mostly employees at the university and they represent the TESMA actors.

After initialising the system and creating the actors, the institution's director is able to specify the institutions and their descriptions. He describes the whole institutions, his structure, the goal, university partner and some contact point. By completing the institution's specification, the institution's director is able to create some new programs, which will be specified by a program director. The program director specifies the program descriptions and collects the course specifications of the instructors to organise the

program. The instructors are now able to specify courses for the program, which are accepted by the program director and validated by the institution's director. When the program director receives all courses, he starts managing the program. Then, the program director specifies the program organisation by structuring the courses into terms. For specifying the course, the instructors have to specify the course description, course content and course evaluations.

The last step consists of certifying a program or a course. A program or a course can be certified according to some internal teaching standard or by some international standard like SWEBOK or CS2013 for computer science. The big advantage of these certification are that we are able to compare the course in the international teaching meetings. It allows an user to compare the 2 different programs or courses and makes it easier to decide whether some students are accepted in an university or not in case of changes. But not only for student, it is also important for different teachers to compare the courses and programs with other universities. The institution can better prepare a program, by being similar to other universities or proposing completely new programs.

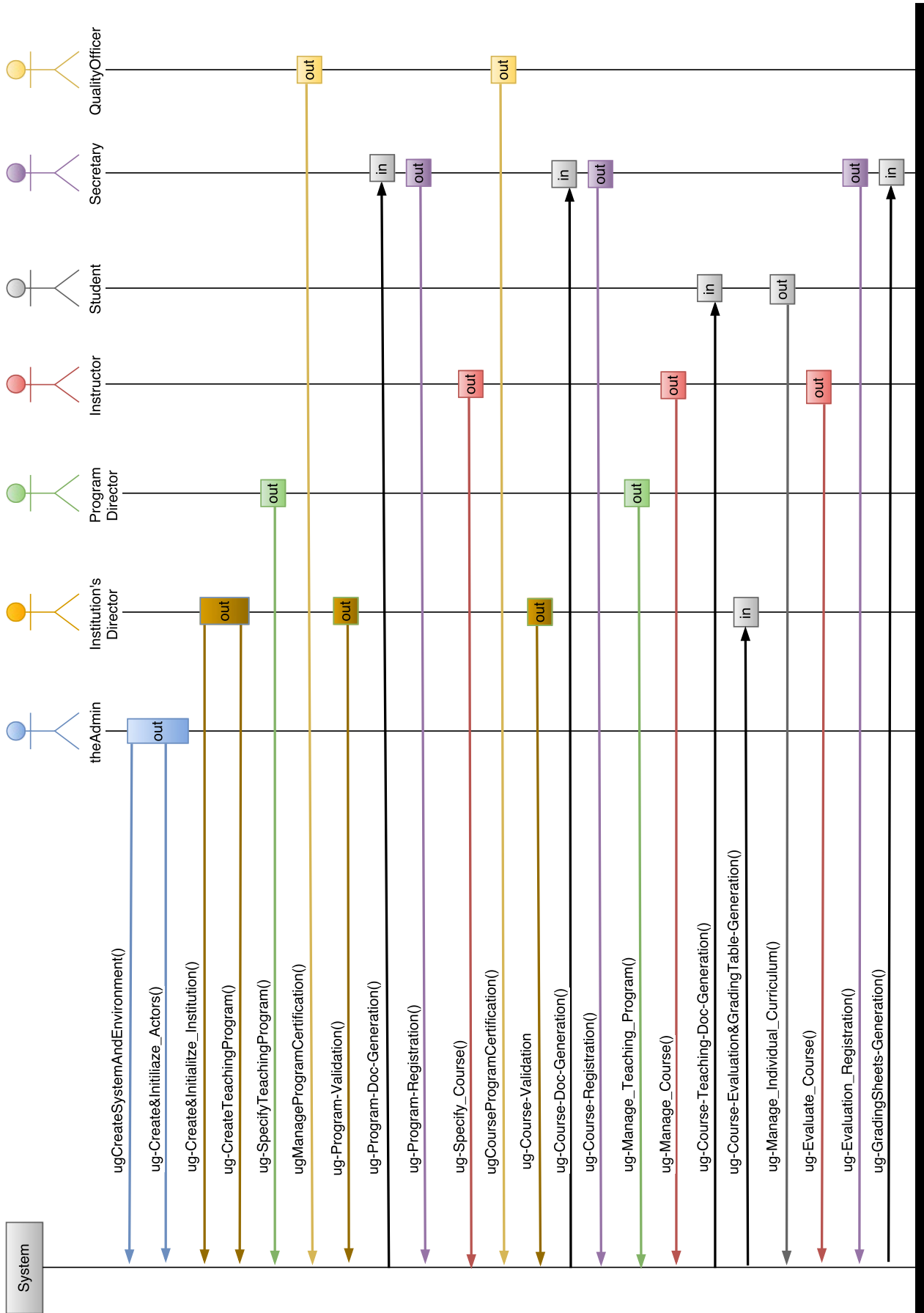


Figure 3.2: Usecase instance of specifying a teaching program

3.3 Concept Model

In this section, we describe the TESMA concept, which contains all class types, their attributes, and relations. The concept model describes our model of TESMA. It gives a high-level description of the TESMA model.

3.3.1 Class-types for actors

Several actors need TESMA for writing program specifications. In Figure 3.3, we illustrate the class types of the actors, who are using TESMA.

C ctInstitutionDirector	C ctStudent
Ⓐ firstName : ptString Ⓐ lastname : ptString Ⓐ email : ptString	Ⓐ firstName : ptString Ⓐ lastname : ptString Ⓐ email : ptString
C ctProgramDirector	C ctInstructor
Ⓐ firstName : ptString Ⓐ lastname : ptString Ⓐ email : ptString	Ⓐ firstName : ptString Ⓐ lastname : ptString Ⓐ email : ptString
C ctQualityManager	C ctSecretary
Ⓐ firstName : ptString Ⓐ lastname : ptString Ⓐ email : ptString	Ⓐ firstName : ptString Ⓐ lastname : ptString Ⓐ email : ptString
C ctSystemAdministrator	
Ⓐ firstName : ptString Ⓐ lastname : ptString Ⓐ email : ptString	

Figure 3.3: Class types for physical actors

The following physical actors are directly concerned by the program specification, and part of the TESMA tool. Currently, the definition of the different actor is kept very simple, since we concentrate on the development of our DSL. However, the model will be extended in the future work for being a base for a complete TESMA environment with user handling, and user registrations. The functions executed by the different actors are kept general, and only described in natural language for the future work.

- System Administrator

Attributes : A system administrator is defined by its firstname, lastname, and email.

User-goals : The system administrate executes ugCreateSystemAndEnvironment, and ug-Create&Initialize_Actors.

- Institution director

Attributes : A institution director is defined by its firstname, lastname, and email.

User-goals : The institution's director executes ugInitialiseInstitutions, ugCreateTeachingProgram, ugProgramValidation, and ugCourseValidation.

- Program director

Attributes : A program director is defined by its firstname, lastname, and email.

User-goals : The program director executes ugSpecifyTeachingProgram, and ugManageTeachingProgram.

- Instructor

Attributes : A instructor is defined by its firstname, lastname, and email.

User-goals : The instructor executes ugSpecifyCourse, ugManageCourse, and ugEvaluateCourse

- Student

Attributes : A student is defined by its firstname, lastname, and email.

User-goals : The student executes ugManageIndividualCurriculum.

- Secretary

Attributes : A secretary is defined by its firstname, lastname, and email.

User-goals : The secretary executes ugProgramRegistration, ugCourseRegistration, and ugEvaluationRegistration.

- Quality Officer

Attributes : A quality officer is defined by its firstname, lastname, and email.

User-goals : The quality officer executes ugManageProgramCertification, and ugManageCourseCertification.

These actors are part of the concept. These actors will be related to the courses, and programs specified with TESMA. We will continue by describing some logical actors.

In educational institutions people often work in groups of teachers, or students, which forced us to create logical actors. The big advantage is that, we are able to specify a course for a group of students, and teachers, without being redundant. It simplifies the usage of our model, and make is more performant. Figure 3.4 illustrate the different class types for logical actors.

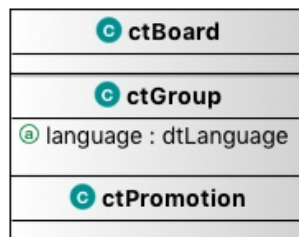


Figure 3.4: Class types for logical actors

The following logical actors are created for grouping :

- A *Board* is a group of instructors. A Board is in general a Teaching Team, or Evaluation Team of a course. The instructor inside the Board are specifying the course, teaching the students, and evaluating the students. They are also updating their course specifications.

- A *Promotion* is a complete class of students inscribed in a course. The class consists of group of students, containing one, or more students. Each group of students have one supervisor, who teaches the students. This structure allows the user to specify individual work but also group work. It can also be used to subdivide the class in three different teaching teams, where each instructor does his part of the course.
- A *Group* is simply a group of students, who have a instructor as supervisor inside the group.

Using the physical, and logical actors we are able to define the actors class diagram. In Figure 3.5, we illustrate the actors class diagram to show the relation between the actors. In general, a instructors are part of a teaching board. This board can contain one, or more instructors. A class following a course is a promotion, which consists of groups of students. A group contains at least one student, where each group is managed by a instructors.

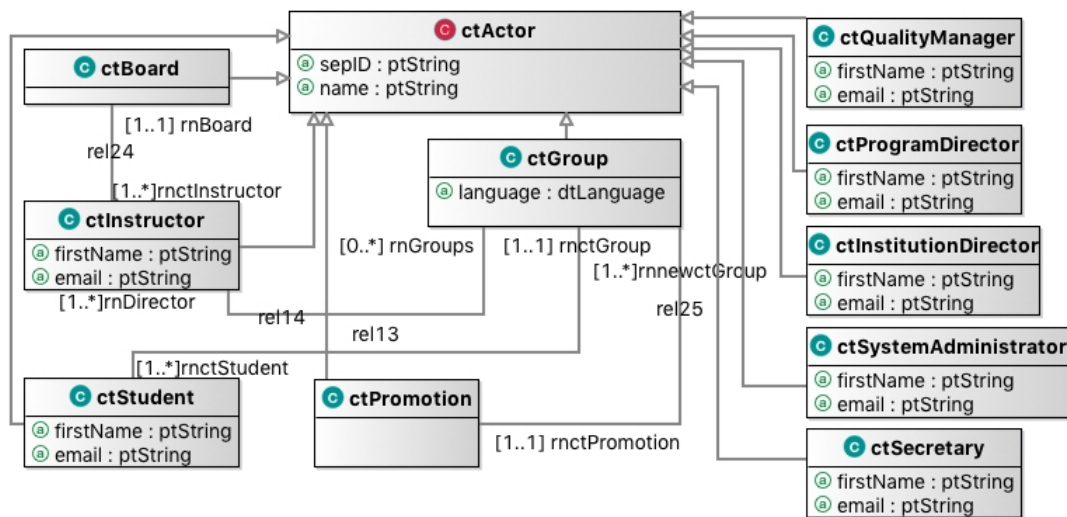


Figure 3.5: Class diagram for actors

The actors are design in, order to be adaptable to institution's needs. By defining the logical actors, we want to cover a larger group of institutions. Additionally, the logical actors allow us to avoid redundant information in the specification. This means that we can specify for example evaluations criteria for a group of students. Since our goal is to save time for the course specification, we designed the model to be efficient in terms of timing costs, and information redundancy.

3.3.2 Class types for educational institutions

In this section, we present the different class types needed in an educational institution. These class types are used for the specification, and defining a whole program specification. We begin from the highest level (i.e. the education institution), and present the different class types inside it step-by-step. In Figure 3.6, we show the different class types needed for a educational institution. In general, we see three levels, which are defined by the institution, the program, and a course. These three levels are related together, and have their own internal structure. In the following paragraphs the different class types are introduced.

ctInstitution	ctProgram	ctCourse
<ul style="list-style-type: none"> ⓐ sepRegion : ptInteger ⓐ name : ptString ⓐ iID : ptInteger ⓐ address : ptString 	<ul style="list-style-type: none"> ⓐ name : ptString ⓐ email : ptString ⓐ description : ptString ⓐ requisites : ptString ⓐ requirements : ptString ⓐ pID : ptInteger ⓐ weblink : ptString ⓐ isced : ptInteger 	<ul style="list-style-type: none"> ⓐ name : ptString ⓐ statute : ptString ⓐ credits : ptInteger ⓐ cID : ptInteger ⓐ description : ptString ⓐ requisites : ptString ⓐ requirements : ptString ⓐ timing : ptString ⓐ weblink : ptString

Figure 3.6: Class types for educational institutions, and their teaching programs

ctInstitution

An *institution* is an educational building, where students of different ages can expand their knowledge in some domains, and get a certification after passing all the knowledge evaluation tests. In general, every institution have different internal structures. Smaller institutions consist often of one educational branch e.g primary schools. Bigger institutions, like universities are often split in some sub institutions in, order to differentiate the knowledge areas. The institutions propose some programs, which the students are able to follow after registering for the program.

We define an instituton with the *ctInstitution* class type. The *ctInstitution* is defined by the following attributes :

- An ID for identification
- The insituton's name
- The address, where the institution is located
- The region, which describes the region where the institution is located, if it consists of multiple buildings.

A institution contains a number of programs. Students may inscribe for a program. We need a definition of a program, which we will describe now.

ctProgram

A *program* is a set of courses proposed at an educational institution. Students have to chose a program, and register for it to follow the courses inside. A program can be

subdivided into sub-programs for distributing the courses for different groups of students. The student have to register for the program. This registration is often costly, and needs to be specified by the program director. In, order to attempt to a program the students needs to fulfil some pre-requisites. The pre-requisites can be some exam, or some diplomas, which a students needs to have. The courses proposed in a program are divided into terms.

We define a program with the *ctProgram* class type. The *ctProgram* is characterised by the following attributes :

- An ID for identification
- The program name
- The contact email address
- The requisites, which needs to be fulfilled for following the program
- The requirements, which describe the required knowledge for following the program. (e.g. language, technical etc)
- A description of the program
- A link to the program website
- The international standard classification of education level

A program consists of a set of courses. Therefore, we will continue by specifying the course class type.

ctCourse

A *course* is a part of a term, where a subject is taught by a instructors. Usually a teacher, or a professor are leading a lecture. In higher education, there exists several types of lectures like :

1. In a lecture, the instructor presents one, or more subject during the term. The students do not need to interact with the instructor
2. In a seminar, the students prepare, and present different subjects, or topics during a term
3. In a colloquium, readings are assigned for each session, which will be discussed during the meeting with the course followers
4. In a tutorial course, students are working individually, or in groups on a topic, and discuss the progress with the instructor during a meetings work. The instructor is here a guide for the students.
5. In a directed individual study course, a student applies for an area of study e.g research topic. In general, this type of course is more concentrated, and in-depth than a lecture. The topic needs to be approved by a institution director. In general, the student is guided by a faculty member.
6. In a laboratory course, the students are mostly working on practical works in a laboratory. However, before starting a laboratory session, theory can be taught within a lecture.

During a course the students have to do some tasks. Either they have to attempt to a lecture, and listen to the instructor's presentation, or they have some practical work. Homework, or preparation are also considered as course tasks. If a set of tasks has been done the instructor can evaluate the student by proposing a test on this tasks. This can be done by covering a whole course, or by doing intermediate tests during the course run time. This is applicable for any type of lecture cited here.

We define a course with the *ctCourse* class type. The *ctCourse* is characterised by the following attributes :

- An ID for identification
- The course name
- The credits
- The course description
- The requisites for attempting to a course
- The knowledge requirements for following a course
- The timing information such as academic year, term, or timing slot
- A link to the course website

After completing the course model, we will define the relations between institutions, programs, and courses. These are not contained in each other. Thus, they do not depend on each other, and may be specified separately.

3.3.3 Relations between the class types for the actors, and the educational institutions

TESMA allows independent specifications of institutions, programs, or courses. These 3 elements are the main class types of TESMA, and are not contained in each other. The specification of these class types may be done by different actors. We illustrate the relational class diagram of these class types in Figure 3.7

The institution can be subdivided into sub-institution. An institution, like a university, is often subdivided into different faculties. These can be specified, and presented separately. All these institutions contain a list of programs in which a student can inscribe for. The programs again can be divided into different subprograms, if it concerns a larger group of students of different domains. Finally, the program is a set of courses, which are taught by an instructor, and followed by students. This illustrates a general overview of a big part of institutions in the world. Institutions are free to choose their structure, and specify any component they want. The flexibility of TESMA allows the institutions to specify their own needs without having a predefined schema.

3.3.4 Classtype *ctCourse*

This section describes the course class type, and it is internal class types used for the specification. These class types are used to describe a complete course. A complete course consists in general of lectures, tasks, and tests. These 3 information are needed to describe the course content. To describe the course running, the concerned instructor needs to sort the lectures, tasks, and tests according to the timing slots in a term. Additionally, evaluation categories, and criteria needs to be defined for evaluating the

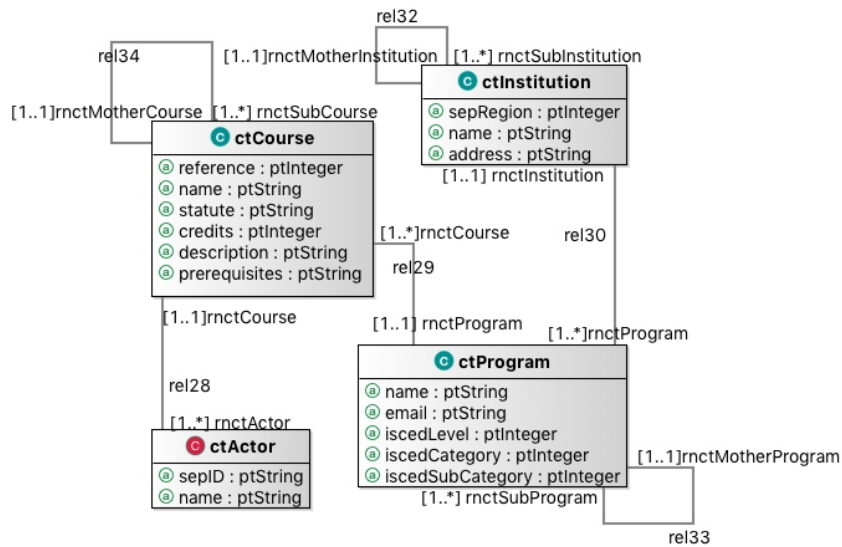


Figure 3.7: TESMA High-level class diagram

students who are following the course. In Figure 3.8, we see an general overview of the different class types, which are used for specifying a course.

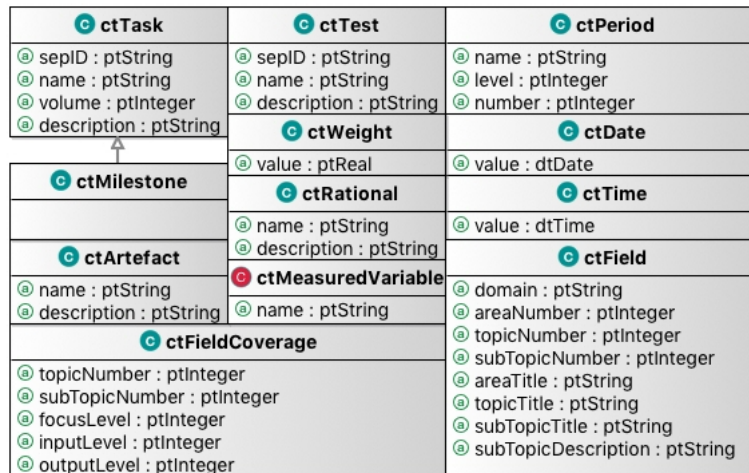


Figure 3.8: Class types related to a course

The following class types are used for specifying a course :

- The *ctTask* class type represents the work that has to be done by the instructors, or by the students. The task can be either a student's task, or a instructor's task. A task is defined by its id, task-name, volume, and his description. The volume represents the workload for the actor.
- The *ctMilestone* class type is a task, which is covered in a test. These tasks are mostly student's tasks, and to be covered by the students for the test.
- A *ctTest* is a evaluation of several tasks done by the students to grade the students for passing the course. The test defines, whether the students passed the course,

or not. The test is defined by an id, a test-name, and a description.

- A *ctPeriod* is a subdivision of a term, where a course is taught. The subperiod is defined by a start-date, end-date, start-time, and end-time.
- *ctDate* is a reference to a day in a calendar system, which is composed of a day, month, and year.
- *ctTime* is a particular point in time of a day. The time is composed of two numbers separated by a double point representing the hours and minutes.
- A *ctArtefact* is an attachment to a task, which can be course material, presentation, or any other supply needed for the course. An artefact is defined by its name, and a description.
- A *ctWeight* describes the importance of an evaluation. In general, we can say that the higher the weight, the higher the importance of the evaluation is. Additionally, the grade of the evaluation increases in importance, if the weight increases. The weight is defined with a natural number.
- A *ctRational* is the description of a part of a test. The rational describes the subparts of the test in detail. A test can have multiple rationals inside it. An rational is defined by its name, and a description.
- The *ctMeasuredVariable* is an abstract class, which covers the evaluation grading system. The instructor specifies his grading methodology. The measured variable will be covered in the next section.
- The *ctFieldCoverage* describes which fields of an international learning standard are covered by a course, or a program. It is defined by an id, a focus level, and an input/output level. These are defined by the Bloom taxonomy, which is a standard for classifying learning objectives. [33]
- The *ctField* class describes the a knowledge-area, topic,, or subtopic of an international standard. A field is defined by the ids of the knowledge-areas, topics, or subtopics, and their descriptions.

We will now start talking about the relations, where the course class type is involved it. We will describe all class types and relations used to describe the Course specification model.

Figure 3.9 illustrated a general overview of the course specification model. A course is defined with his attributes. These attributes describe the general overview of a course e.g. the course name, credits, a course description in pure text, the requisites and so on. To specify the course runtime content, the course can be subdivided into periods during a term. These periods can be defined for a date range, or a time range. The reflexivity property of the period allows a period to contain some sub-periods. These sub-periods contain the same timing ranges. Tasks, and Tests are related to a course, or to a period. The advantage of such an approach is that the user have a free choice. The user may want to define his lectures in deep detail by defining all the periods inside a term, or to define his course generally by specifying only the tasks, and test, which are requested. The idea is that the instructor can specify his course according to the institution's internal rules.

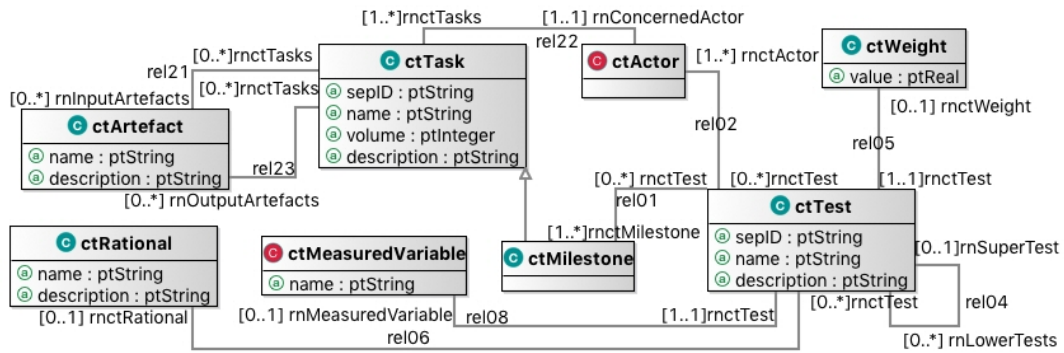


Figure 3.10: TESMA test, and task metamodel

- A *numerical variable* is a grading type, which is based on natural numbers. The instructor can either chose a continuous, or a discrete variable. Therefore the user has either to measure the grade, or to count the grade. In this case the grade can be either defined by a set of natural numbers for defining a grade, or by an set defined by the lower bound, the upper bound, and the steps between these bounds. The advantage of the numerical approach that it makes much more easier to compare tests, and to define the criteria for passing a course. However, the numerical grade are often not exactly reflecting the knowledge of a student.
- A *categorical variable* is a grading style, which is done in natural language. Therefore the instructor can chose either to define the evaluation using nominal, or ordinal variable. Using the nominal variable, the instructor proposes a set of non-ordered variables, which can be used for evaluating a course, e.g competences, objectives. The instructor could grade the students according some evaluation criteria without having a numerical grade. The non-ordered variables are often very difficult to be interpreted since most institutions use numerical variables for describing the evaluation. However nominal variable may be even more precise in demonstrating the exact problems of a students. The ordered variables describe a method for grading the test with an ordered set of natural language criteria, like excellent, very good, good and so on.

Finally, after having specification a program or

3.3.8 Program, and Course Certification

This section describes the program, and course certification according to an international standard. A program, a course, or a task can be related to a international standard. An international standard is defined by a knowledge area, topic, and subtopic, which are represented as fields in our model. A program, course, or task can cover a part of the international standard, which is defined by the FieldCoverage. However, it does not have to be international standard, the institution can specify his own standard, which needs to be fulfilled by the program. In Figure 3.12, we illustrate the certification of a course, program, or task.

So far, We presented the functional requirements, the environmental model and the concept model, we will now focus on the non-functional requirements and the quality of domain-specific languages

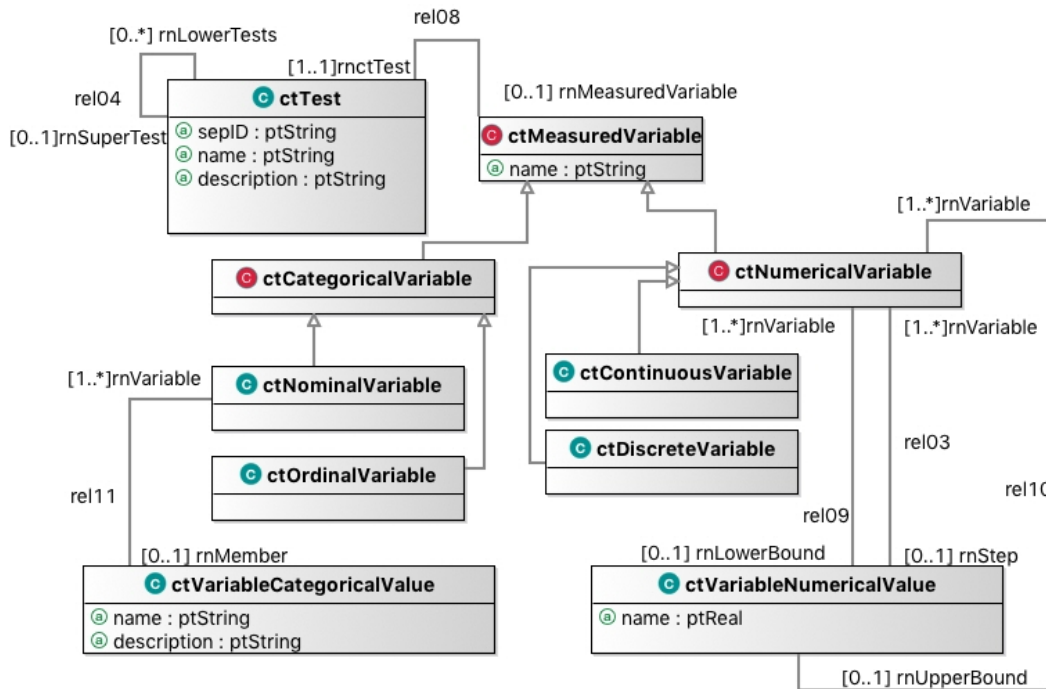


Figure 3.11: TESMA evaluation metamodel

3.4 Quality of DSLs and non-functional requirements

In this section, we present the non-functional requirements for the TESMA tool suite. We defined a set of requirements, which covers the quality of the domain-specific language. We will describe each of these requirements, and motivate the decision we made for taking them. Additionally, Since, we are generating documents, and providing a user interface, we will define the different requirements needed for the 3 parts of TESMA.

Figure 3.13 represents a schema of the requirements. The requirements are regrouped by their domain. Even if the requirements are regrouped, and split into the 3 main groups : Quality, Dependability, and Maintainability, this 3 core requirements depend on each other. A dependable system is maintainable, Since, the maintainability is part of the dependability requirements. Additionally, creating a maintainable, and dependable DSL improves also the quality of the domain-specific language. The following subsections describe the three core ideas.

3.4.1 Quality

The first requirement, we need to guarantee is the quality requirement. The quality requirement describes the domain-specific language. It provides information about the usability, readability, longevity, genericity, and simplicity of the DSL. These requirements are defining together the quality. It is important for us to have a domain-specific language of high quality in order to make the DSL usable by a larger group of people. Not only computer-experts should be able to use it, but also instructors, and program directors (our teaching domain experts) from other domains. Additionally, we want to allow other institutions to use our tool, which again makes the group of users very large. Hence, the domain-specific language has to be comprehensible, and easy to understand, to reduce

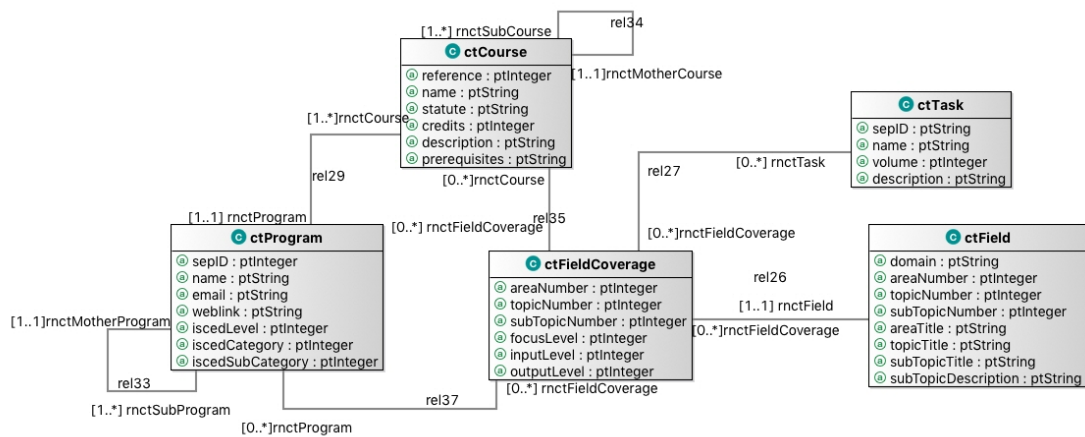


Figure 3.12: TESMA certification metamodel

Domain-Specific Languages		
Quality	Dependability	Maintainability
Readability	Integrity	Modularity
Longevity	Availability	Functionality
Simplicity	Reliability	Scalability
Usability	Safety	Flexibility
Effectiveness	Confidentiality	
Efficiency		
Accessibility		
Satisfaction		
Genericity		

Figure 3.13: Quality Requirements

the costs of teaching, and learning the language. We will start describing the different requirements, and explain the need in TESMA.

Readability

The *readability* requirement defines the structure of the specification. It ensures that the specification written in the DSL has to have a structure, is easy to follow, and contains explicit keywords, which are easy to follow. The different classtypes of the specification should be organised hierarchically, and their attributes should be listed one after another, to make it readable.

For TESMA the readability attributes is very important, Since, we have to structure the course content, and program contents to make it readable. The user should feel comfortable, and being able to understand the course content by reading the specification in the DSL.

Longevity

The *longevity* requirement ensures that the DSL should be used over a non-trivial period of time. In general, the developers see if the DSL is used over a long period of time, than the DSL may be of good quality. By ensuring the longevity requirement, we improve the quality of the DSL automatically.

TESMA should be usable over a longer period of time in order to allow the institution to compare their specification with other specifications of other institutions. This allows the institution to improve and update regularly their program.

Simplicity

The *simplicity* requirement ensures that a DSL is easy-to-learn,, and easy-to-understand. The users have not to invest a big amount of time in learning the domain-specific language.

The simplicity attribute should be definitely ensured in TESMA. Since, we are having multiple institutions, and users, we want to ensure that TESMA is easy to learn for non-domain experts.

Usability

The *usability* requirement is divided into 4 different sub-requirements : effectiveness, efficiency, accessibility, and satisfaction.

- The *effectiveness* requirement ensures that the accuracy of completing the implementation of the specification in the DSL is high. TESMA should be intuitive to use, and encourage the user to finish his specification, without giving up.
- The *efficiency* requirement ensures that the costs for using the domain-specific language is kept low. These costs can be physical efforts, timing costs or financial costs.
- The *accessibility* requirement ensures the easiness of finding the appropriate keywords for specifying parts of a program. In TESMA, the actors should be able to rapidly find the keywords, and the DSL structure for specifying their programs.
- The *satisfaction* requirement ensures the positive attitude towards using the domain-specific language. In our case, the institutions should be encouraged to use TESMA for their program specifications.

Genericity

The *genericity* requirement ensures that a DSL can be used by a large group of people, and that it is adaptable to their needs. The DSL needs to ensure configuration according to the user's needs, and adapt the grammar to their laws. TESMA needs the genericity requirement in order to cover the institution needs. Since, there are plenty of institution's around the world, we want to cover a large set of institution by providing a generic DSL.

3.4.2 Dependability

The dependability attribute ensures that a software can be trusted during a certain period of time. Additionally it ensures the following requirements : Availability, Integrity, Reliability, Safety, Confidentiality, and Maintainability. We will explain these attributes

one by one expect of the Maintainability attribute. Since, maintainability has a direct link to the quality, we subdivided it into sub-requirements, which will be presented in the next section. The dependability attribute is very important to ensure the longevity of the DSL, and make the system stable, and trustable.

Availability

The *availability* requirement ensures the readiness of a correct system behaviour. We want to ensure that the user is able to specify at any time the class-type.

Integrity

The *integrity* requirement ensure the absence of incorrect system modifications. We want to ensure that the specified programs are preserved, and not modified by the system. This attribute plays a big role for the genericity. Since, adapting the DSL to the universities needs could cause unexpected system behaviours.

Reliability

The *reliability* requirement ensures the continuity of a correct system behaviour. The reliability attribute should ensure that the system should continue working as expected while specifying a program.

Safety

The *safety* requirement ensure that there are no critical consequences on the actors, and the DSL environment. The actor should be able to use safely the textual editor.

Confidentiality

The *confidentiality* requirement ensures that the specifications are only accessible by the authorised persons. In our case, for example a students should not be able to enter the instructor's view for specifying a course. This requirement is very important for ensuring correct, and trustable data.

3.4.3 Maintainability

Finally, the domain-specific language needs to be maintainable for the actors. The maintainability is part of the dependability requirement. However, we think that it might be interesting to sub-divide the maintainability attributed for getting a DSL of high quality. The maintainability requirements needs to be ensured for the software maintenance team or the software engineers in order to do updates on the DSL. Therefore, we want to ensure the following requirements :

Flexibility

The *flexibility* requirement ensure that the DSL is adaptable to the user's changes. The user should be able to add data to the generated documents or change the design of documents according to his needs. Additionally, the grammar, and the document generator should be constructed in order to allow small changes to the institutions.

Modularity

The *modularity* requirements ensures that the specification can be split into different modules. These modules can be different files or different regions. It applies a better readability, and allows to user to structure himself his own program description according to his needs. The modularity may improve the quality of the domain-specific language.

Functionality

The *functionality* requirement ensure that a system provides the capacity for providing useful functions. These functions can be DSL validator rules or generator rules. These functions should be useful, and easy to maintain. Ensuring the functionality attributes improves also the maintainability of a system.

Scalability

The *scalability* requirements ensures that the system can ensure a high amount of work. TESMA needs to keep a large amount of data, which will probably increase from year to year. The DSL has to be scalable, and by that the software maintainer should be able to easily update the DSL according to the institution's needs. If attributes needs to be added, they have to quickly find the location, and update the corresponding fields. Additionally, enlarging the generated document is also important for the actors.

In this chapter, we presented the actors, who will use the TESMA tool suite and their requirements. We described the concept model, talked about the different class types, which we need for designing our grammar. We presented some examples of specifications by using use case models, and use case instances. Lastly, we talked about our research topic and presented the non-functional requirements for designing a domain-specific language of high quality. Since, we will now continue with the Design and Realisation of the TESMA tool suite.

Chapter 4

TESMA Design, and Realisation

In this chapter, we present the design and the realisation of TESMA. We start by presenting the architecture, the TESMA tool, the DSL class diagrams and the design decisions for improving the quality of a DSL followed by the TESMA validator and grammar checker. Talking about the document generator, who generates the specification, and teaching material.

4.1 Architecture

In this section, we talk about the design and the development of a first version of the tool. TESMA is based on three main architecture components illustrated in 4.1. These three components are based on stable plugins, e.g. Xtext, Xtend, and Sirius, which are based on the Eclipse Modelling Framework (EMF).

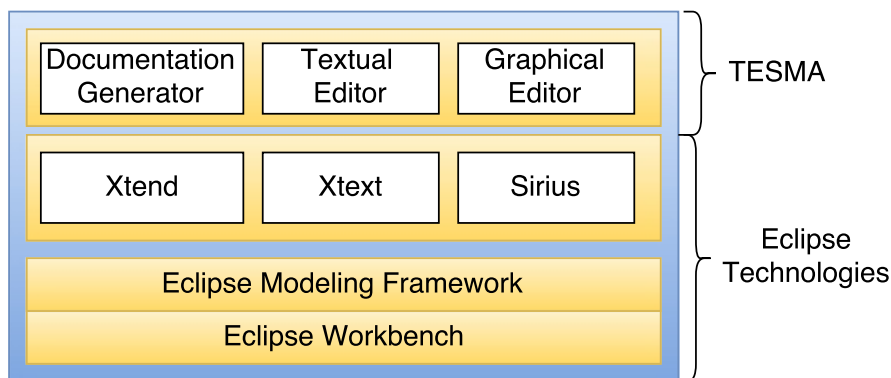


Figure 4.1: TESMA software architecture

One major design choice was to create a domain-specific language based on the Xtext framework for specifying programs and courses. Xtext is an open-source framework that eases the development of domain-specific languages and offers functionalities to develop a *textual editor* for the TESMA domain-specific language. It supports a validator for implementing a grammar checker, which facilitates the program specification in TESMA. Since, Xtext is based on the Eclipse Modelling Framework, an EMF model is generated out of the grammar definition. This EMF model can be used as a underlying-core library for representing the model in a user interface with Sirius. Sirius is an open-source software Eclipse project based on EMF that eases the creation of custom graphical modelling workbenches. It allows a simple graphical illustration of the model, but allows updates on

the model according to the users modifications. Since, the EMF model is the underlying-core library for Sirius and Xtext, both are interacting according to the specification written with the DSL. The model updates the textual and the graphical editor. The process of creating the model with Xtext and representing it with Sirius is illustrated in Figure 4.2. Unfortunately, the user interface is currently not available in the current version of TESMA, since we concentrated in creating a DSL of high quality for generating an advanced model. A big part of the user interface and the functionalities are already included in the concept.

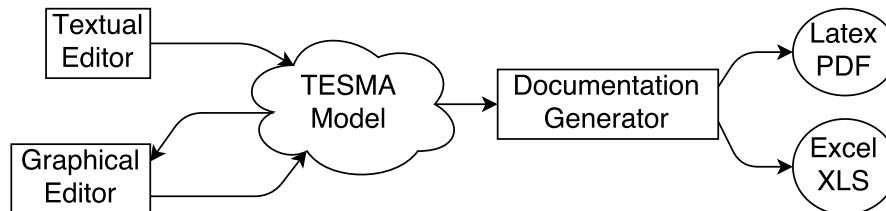


Figure 4.2: TESMA process overview

After having completed the specification of the program or a course, an actor may generate documents, which can be used as teaching material or for distributing information about the course content. The last component of TESMA is the Document Generator, which is based on Xtend. The document generator takes the EMF model, which is generated out of the specification of the program as input. The Generator creates the full report in LaTeX version, which is translated to a PDF file. This report contains all descriptions of the programs and courses, which has been specified with the domain-specific languages. Additionally, some other documents are generated, which are attached to the report. These documents will be presented in the section Document Generator.

4.2 Textual Editor

The first TESMA component, we will describe is the textual editor. The textual editor is used for writing the specification of the programs with the domain-specific language. We will present the features of the textual editor, the tool, the grammar, and the design decisions of the grammar.

4.2.1 Description

The technology used to develop our textual editor is an open-source framework called Xtext, which provides features for creating a domain-specific language by defining a grammar. In our case, we defined the grammar for specifying programs and course descriptions. It generates out of the grammar an EMF model. This model can be used in other tools for graphical illustration or document generation. The textual editor is integrated in an Eclipse Environment containing different views for managing the program specification. These views are illustrated in Figure 4.3. The main views of Eclipse are the Package Explorer and the textual editor view. These views are used during the specification of a program.

The package explorer is used for organising and structuring the program specification project. The program and course specifier may divide his project into several folders of different types containing the specification files of the institutions, programs, courses, tasks, tests and so on. We set the file extension of the specification files to ".tes", which

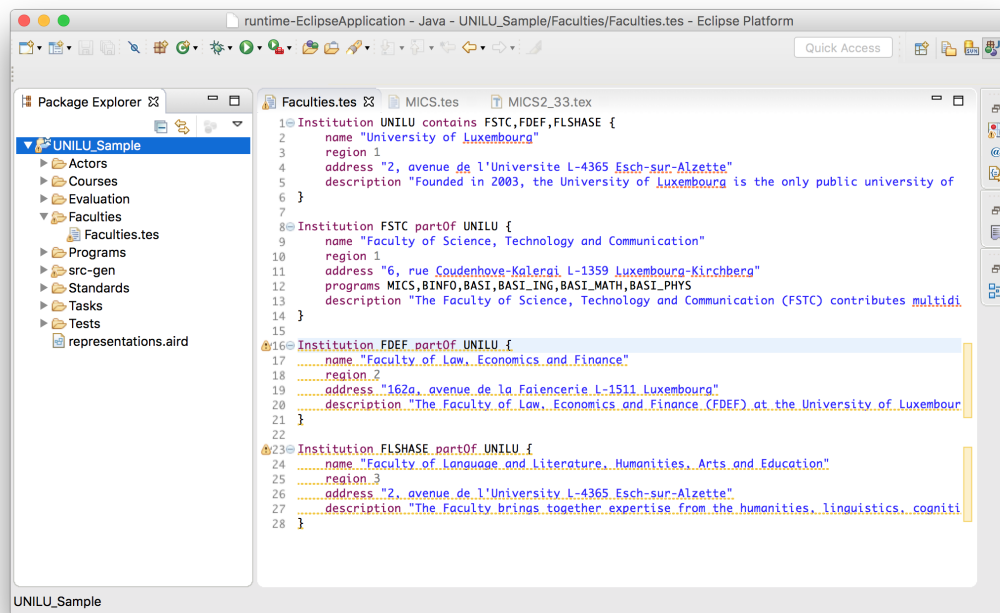


Figure 4.3: TESMA textual editor

is an abbreviation for TESMA. These files can be placed in different category folders. In Figure 4.3, we proposed a possible project structure for dividing the project into several files. The specifier may create different files for his specification. The class-types can be specified independently of each other. This guarantees a better readability and accessibility of the specified information by providing a structure of the project.

The textual editor is used for writing the specification in the TESMA domain-specific language. It supports a number of features to help the user to write correctly the specification. Currently, not all features are ready to use, but we have planned to implement them in a future version of TESMA (see chapter 7). The validator and the document generator are ready to use. The document generator can be executed from the package explorer. It takes the model generated out of the specification as input and creates a full report in natural language. The validator is a grammar checker and shows errors and warnings to the user if they type mistakes inside the language. A model outline can be used for illustrating the model in a hierarchical order, which help the user to have an overview of his specification. In general, all these features help the user to write their specification. All the static keywords used for declaring class types or attributes, are written by the system and the specifier may only fill out the missing fields. In any case, he can extend the template by writing manually additional attributes. These are listed by proposals, which are a list of keywords, which can be placed after another completed attribute. It provides information about possible attributes which can be written after finishing another one. Lastly, the configuration feature allows to the institution to adapt the keywords used in the domain-specific language to the university's policy. They have the possibility to choose their own naming conventions.

In general, the textual editor helps using the domain-specific language and specifying the programs. In the next section we will talk about the domain-specific language and their design.

4.2.2 Domain-Specific Language

Our main design choice was to design an intuitive, customisable, and loosely coupled domain-specific language. In this section, we will show you the implemented TESMA model and its grammar. Figure 4.4 illustrates a general UML model of the implemented TESMA grammar. This model is generated out of the grammar written by us.

In the UML model, a common superclass called Model is placed at the top of the figure. According to our design, the user has the possibility to start with one of the following class types for specifying a program:

- ctInstitution
- ctProgram
- ctCourse
- ctStandard
- ctActor
- ctTask
- ctTest
- ctCategory
- ctCriteria

These class types are independent of each other and can be created separately and without containing each other. This allows the usage of the DSL by several actors at the same time. A program director can specify a program, while several instructors are specifying a course. After completing the specification, the actors can relate 2 instances of a class type together by a relation, following the UML class diagram. We stick in some cases on containments, like the ctPeriod class type, which needs to be defined inside a ctCourse class type, because we want to improve the readability of attributes and relations inside a course. These class types are used as grouping classes. Finally, the user is able to define his tasks in different periods of a term and structure his course. We will talk about these design choices in more detail later in section 4.2.2.

The full TESMA grammar is shown in section 9.1. We will now describe the different class types now and talk about the design decisions.

ctInstitution

The *ctInstitution* class type is created by the institution's director and is describing a educational institution or parts of it.

For specifying an institution, we have defined by the following attributes :

- The *name* attribute defines the identification of the class-type.
- The *hide* attribute is used for hiding the specified instance inside the generated document.
- The *region* attribute is used to define the region, where the institution is located.
- The *address* attribute is used to specify the postal address of the institution.
- The *description* attribute is used to generally describe the educational institution.
- The *currentSituation* attribute is used to describe the current situation, perspectives, and goals of the institution.
- The *gastronomie* attribute is used to describe the restaurants, canteen, and pubs etc. for the students.
- The *contact* attribute to define any contact person or contact information for the institution.

An institution is related to some other class types. Inside an institution we define several programs, for which a student needs to register. We define now the list of relations for an institution :

- The *rnctMotherInstitution* relation describes that an institution can be part of a global institution.
- The *rnctSubInstitution* relation describes that a global institution may have some sub institutions.
- The *programs* relation is used for relating an institution to some programs.

We designed the institution such that the specifier may describe the global institutions and transmit some more information about the educational institution to the students for example. We designed the institution class type to be modular and flexible. The institution's director is able to specify his corresponding institution's architecture

ctProgram

The *ctProgram* class type is created by the program director and used for describing generally a program and its content.

For specifying a program, the program director has the possibility to define the following attributes :

- The *name* attribute defines the identification of the class-type.
- The *hide* attribute is used for hiding the specified instance inside the generated document.

- The *isced* attribute is used for specifying the International Standard Classification of Education levels.
- The *description* attribute defines a general description of the program.
- The *prerequisites* attribute defines the knowledge to have and conditions for registering to the program.
- The *requisites* attribute defines the material, tools, or books to have for mastering a program.
- The *costs* attribute defines the prize for registering to a program.
- The *language* attribute defines the languages used during the program.
- The *email* attribute defines the contact email address for the program.
- The *weblink* attribute defines a weblink for more information.

Inside this program the user has to specify several courses, which will be attempted by the students. We will now describe the several relations of the program.

- The *rnctMotherProgram* relation defines the relation to the core program.
- The *rnctSubProgram* relation defines the relation between a core program and several subprograms.
- The *institutions* relation describes to which institution the program belongs to.
- The *courses* relation describes, which programs belong to the program.

However, this field can be extended by auxiliary class types. These auxiliary class types are defined as *ctTerm* and *ctModule*. The specified has the possibility to group the courses of a program inside a term or modules or both in order to structure the courses in timing periods for example. He is able to specify an ordered list of courses. *ctTerm* is used to divide the courses in groups of periods. These groups illustrate when courses are taught at the same time. *ctModule* is used for grouping courses, which depend on each other. A student may pass a module by attempting an satisfactory average grade of the courses inside the module. These specification are the personal choice of the institution.

The program is designed to be simple and easy-to-understand. Natural language has been used to define the attributes and the structure is intuitive and simple to follow.

ctCourse

The *ctCourse* class type is created by the instructor. It is used for describing a course, the course content and the course organisation. We will start by describing the general course attributes, which illustrate the general course description.

- The *name* attribute defines the identification of the class-type.
- The *hide* attribute is used for hiding the specified instance inside the generated document.
- The *label* attribute describes the full name of the class type.

- The *reference* attribute describes an internal identification number for illustrating on the website.
- The *academicyear* attribute describes the year in which the course is taught.
- The *hoursPerWeek* attribute describes the hours per week, where the course is given.
- The *totalHours* attribute describes the total number of hours during a semester.
- The *description* attribute provides a general description of the course.
- The *credits* attribute describes the weight/ECTS/credit points of a course.
- The *languages* attribute describes in which languages the course is given.
- The *weblink* attribute describes the weblink for more information about the course.
- The *nature* attribute describes the type of course (required or elective).

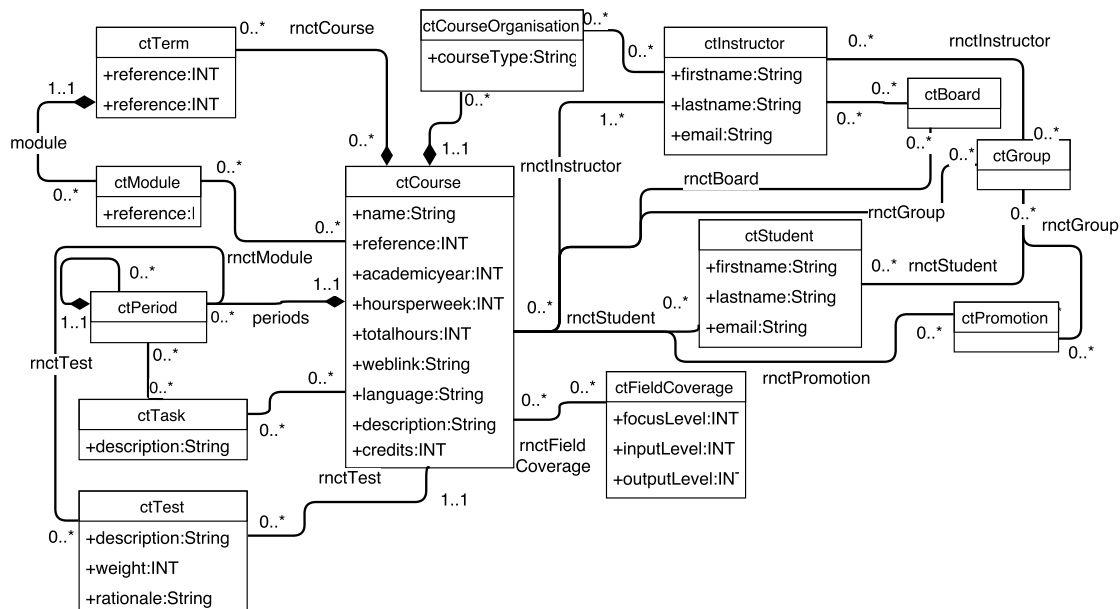


Figure 4.5: Course class type and it relations

The course class type is related to many other class types, which help the user to specify the course in detail. Figure 4.5 illustrated the course class type and its relations. These relations and structures improve the readability and accessibility of information for a course :

- The *rncourse* relation describes if the course belongs or is only inside a program.
- The *corecourse* relation describes to which mother course the sub-course belongs to.
- The *rncourseTerm* relation describes to which term the course belongs to.
- The *rncourseModule* relation describes to which module the course belongs to.

- The *rnctCourseModerator* relation describes, who is responsible for the course.
- The *rnctGroup*, *rnctBoard*, *rnctPromotion* and *rnctStudent* are actors linked to the course.
- The *rnctCourseOrganisation* is a relation to the auxiliary class type *ctCourseOrganisation*, which helps to specify the instructor's role during the course runtime. The specifier may assign a role to the instructor according to the type of course.(lecture, practical, tutorial, other). Additionally, the specifier can add the number of hours and the weight to the concerned instructor.
- The *rnctPeriod* is a relation to the auxiliary class type *ctPeriod*, which helps to organise the lectures during a term. The periods contain a start and end-date as well as a start and end time. The periods may contains other periods, tasks, or tests.
- The *rnctTask* is a relation to the *ctTask* class type. The *ctTask* class type describes the tasks, which need to be done by the instructors or students. The specifier can assign these task to the concerned person. These tasks are specified with a description in pure text and the specifier may add some artefacts, which could be needed during the task.
- The *rnctTest* is a relation to the *ctTest* class type. The *ctTest* class type describes the test, which need to be done by the students. We will describe the Tests in the next section.
- The *rnctFieldCoverage* is a relation to the *ctFieldCoverage* class type. The Field-Coverage describes the coverage of a standard defined by the institution. The standard may be an international standard or an internal standard. The user has to use the Bloom taxonomy for specifying the levels in the field coverage. [33]

ctTest

The Test class type defines the tests, which need to be done by students. They are used to evaluate the students and to cover the tasks, which have been proposed during the lectures. These attributes define a test :

- The *weight* attribute indicates the importance of the test in comparison to the others done during the term.
- The *description* attribute gives a description of the test.
- The *rationale* attribute illustrates more detailed information needed for doing test. This can be course material, information about the organisation and so on.

The Test class types have different relation helping to transmit more specific information to the student. The specifier has the ability to add the grading categories and criteria to the test, as well as the grading approach. The course specifier may define which teacher will be responsible for the gradings. In Figure 4.6, we illustrate the test class type and its relations.

- The *rnctTask* relation describes the tasks, which are covered by the test. (The Learning material)
- The *rnctCategory* relation describes the grading categories, which will be evaluated.

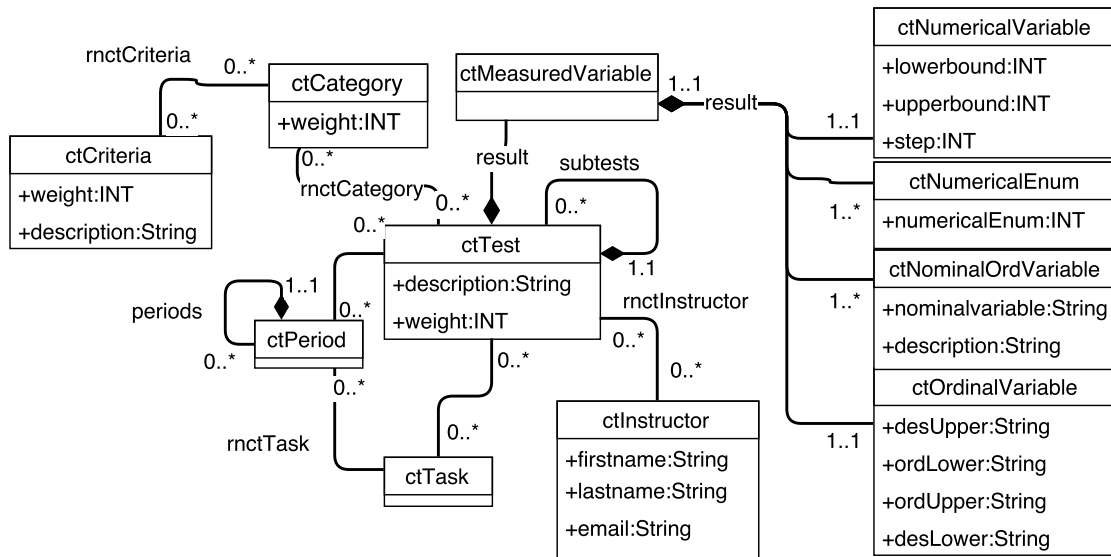


Figure 4.6: Test class type and its relations

- The *rncInstructor* relation describes the instructors, and will evaluate the students.
- The *result* relation describes the grading approach of the instructors.
- The *rnLowerTest* relation divides a global test into smaller test or sub-tests.

Standards

The standard are describing an international or internal teaching standard of the institution. The standard can be chosen by them. Examples of such standards are SWEBOK or CS2013 in the computer science domain. These standards can be specified in the domain specific language. A course, a program or a task can cover fields of the standard, which can be specified by the respective person. In general, these standards are used for certifying the course. The course can be compared to other courses at other institutions.

Evaluation

The user is able to chose between 4 different grading approaches:

- Numerical Variable defines a natural number taken from a set of natural numbers. The set is defined by a lower bound, upper bound, and the steps from the lower bound to the upper bound. (can be infinite)
- Numerical Enumeration defines a finite set of grading numbers.
- Nominal Ordinal Variable defines a grade in natural language with a description.
- Ordinal Variable defines a grade out of a set of grades in natural language.

The evaluation can be specified in different ways, in order to guarantee the genericity of the DSL according the user's needs. The instructor may chose his own grading approach and specify it in order to inform the student.

4.2.3 Features

The textual editor has a number of features, which helps the user to specify the programs. These features help while writing the specification in the domain-specific language.

In this section, we will briefly describe the different features provides by the textual editor. This tool has a number of features which are:

1) Validator

The goal of the validator is to check the syntax validation rules. During the specification of a program, the specifier needs to respect some rules. The validator highlights the errors in the textual editor. Here is a brief summary of the implemented rules :

- *Uniqueness* : The class type's identification must be unique.
- *Completeness* : The class type's specification must be complete to be ready for the document generation.
- *Correctness* : The attributes of a class-type must be correct.
- *Relations* : The relations to the different class-types must be unique, complete and correct.

These four requirements are the general overview of the implementation of the validator. These requirements needs to be fulfilled by all the class types. We plan to evolve the requirements in the future so that we ensure a stable and functional system. The implementation of the full validator can be found in chapter9.4

2) Templates

One feature, which is needed the proposals of templates. When the user creates a new program, we want to provide templates, with the required attributes, which needs to be specified. This helps the user to understand quickly the syntax of the DSL. These templates follow the rules of the grammar and places the cursor to the first place, where data needs to be inserted by the specifier.

Another template feature are the so called proposals. The user gets suggestions for the next keywords of the TESMA language, which can be used. These keywords can be the labels of an attribute or of an class type. Using these proposals, the specifier knows, which keywords he's able to place afterwards.

3)Model Outline

The Outline proposed by the Xtext Framework, illustrates the content of the file in a hierarchical schema. The specifier can easily find instances and the attributes which s/he needs to modify. An user may inspect the Outline view for Information retrieval, which is very important for us to improve the quality of the DSL. We will handle the design choices with respect to the quality of DSL later in this chapter.

Excel-Import-Export

A planned implementation for the textual editor is an Excel import and export of the specified instances. The instructors should have the possibility to specify his course in Excel templates. We want to reach a large amount of users by providing the capability of specifying their courses in Excel and import it then inside our system.The imported

file should add the data to the specification then the instructor may generate his teaching material. For updating the specification, the specifier may change the specification directly or export it in a Excel file and modify there the corresponding fields.

This approach provides flexibility and modularity in the complete system.

Customisability of the domain-specific language

A planned implementation to guarantee the maintainability of the DSL is to allow the user to configure the domain-specific language according to their naming conventions. Using a configuration file, the user may change the proposed names of the keywords used for class types or attributes with their own names. The user will feel more comfortable with an environment adapted to their habits.

4.3 Graphical Editor

The graphical editor has not been implemented yet in the current version of TESMA. However, we plan to implement a graphical editor in the next version. In this section, we will describe the possible future features of the graphical editor. Additionally, we will talk about our design decisions, which we planned to implement in order to cover the non-functional requirements.

4.3.1 Design and Features

The graphical editor provides a representation of the TESMA model in a tabular view. Our objective is to improve the quality, dependability and maintainability requirements with our user interface. The Graphical Editor should help the specifiers with less computer experience to specify their programs and courses. The tabular view shall be a modular and flexible table representing all the data. The model should be illustrated in hierarchical table view, so that less important data can be hidden. The specifier should have the possibility to apply filters on his table. The filter will ensure the accessibility requirement. The idea is to sort the data in the table, to provide an better accessibility of the data. Having filters and data sorters require also multiple row selections. The multiple row selection helps the user to save time for selecting multiple data. Selecting multiple field of a standard for specifying the coverage for a course would be an example of such a scenario. Finally, we plan to add import and export features for Excel files, to help the users to specify their programs without using the Tool.

4.3.2 Technology

We choose Sirius for creating our graphical editor. Sirius provides table views with filters and hide functions. The goal is to work with the import/export, the DSL and the graphical view. After specifying the grammar of the DSL the model, Xtext generates a model of the specification. This model is used as underlying-core library for the graphical editor and the textual editor. Sirius uses this model to display the table and its content. Performing changes in the DSL or in the content of the tables will causes an update on the instance of the Model, so that the textual and graphical editor will be updated automatically. The model-based approach improves the maintainability of a system. The software maintenance team has only to update the model, which will update the editors too.

4.4 Documentation Generation

In this section, we will describe the documentation generation. We will talk about the realisation in a first step and about the different types of documents. We will briefly describe the content of the generated documents.

4.4.1 Generator

The Generator is implemented in the Xtend language. Xtend is based on Java and has several advantages over Java. It is very user-friendly and helps the user writing dynamic texts. In these text, data needs to be inserted, which is specified by the user. In our case, these data may be the program or course attributes. Xtend makes it very easy to create tables and views, using while and for-loops. Data can be so easily be inserted in the different cells of the tables.

In our generator, we generate for all existing instance of the model class types a LaTeX file, containing the description of the institution, program, or course. In Figure 4.7, we illustrate the Explorer of the Documentation Generator.

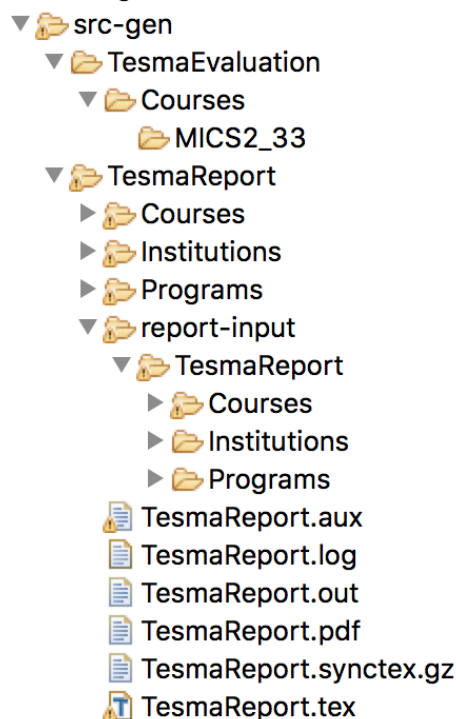


Figure 4.7: Documentation Generation - Report Structure

The core report is created in the "TesmaReport.tex" file. The LaTeX-Reports of the different instances are generated in separate files and grouped in folders depending on their class-type. They are organised by their class-type, which means by Institution, Programs and Courses. Each instance has his own Latex file, which is imported to the main File. The specifier has the possibility to add manually information to the LaTeX file. Therefore, we a folder called report-input is generated containing empty latex files. The specifier inserts the additional information in the corresponding empty LaTeX file. This file, containing the manual input, is imported to the original generated LaTeX file of the report. Finally, Texlipse converts the main LaTeX report to a PDF report, which is printable by the institution. The full generator is available at chapter 9.3.

4.4.2 Document types

We want to generate several documents, which can be used as information sheets, teaching material, and administration material. We want to generate the following documents :

- Description of the whole institution with programs and courses
- Information sheets about programs and courses for students
- Course Runtime information sheets for the students
- Introductory slides for the first lectures of a course
- Evaluation Excel Sheets for the different lectures
- Complete Grading Excel Table of a course
- Course Certification Sheets
- A formulary for registering a course at the institution's office

We want to generate mainly PDF files and Excel files. The goal is to create a homogeneous report of the whole institution with all this programs and courses. This document can be printed out and distributed for advertisements. If many universities use TESMA for specifying their programs, students can better compare the different institutions and chose the one, which fits the best to their needs.

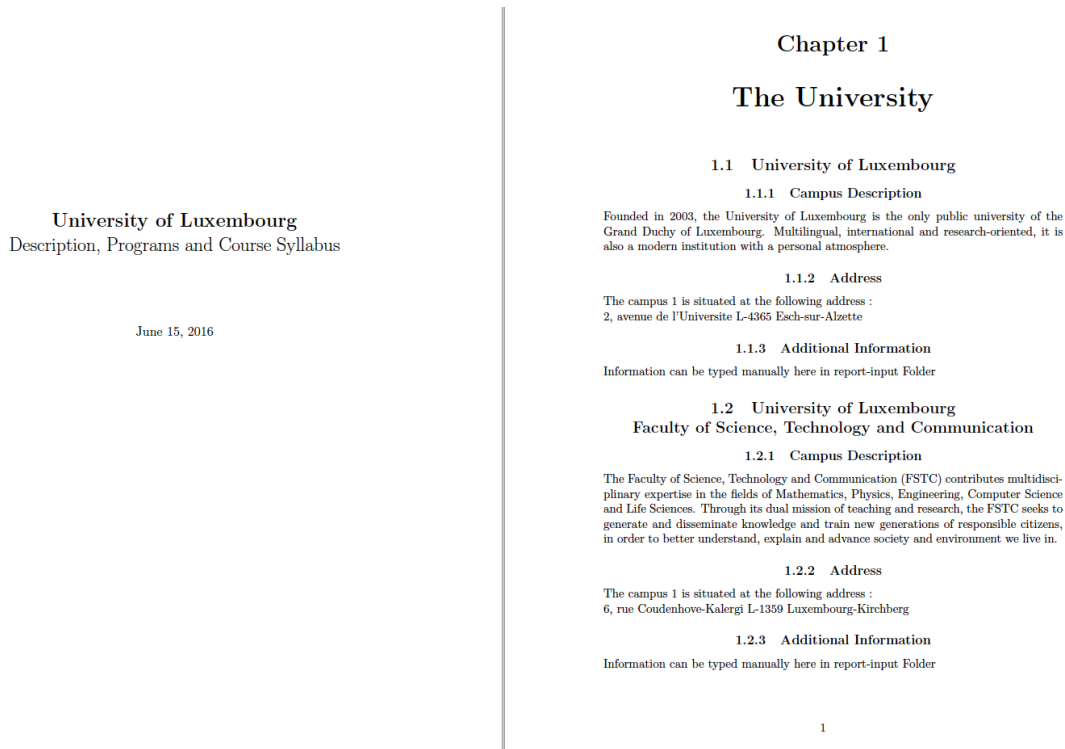


Figure 4.8: Documentation Generation - Report

We want to automatise the process for preparing introductory lecture presentations. The instructor does not have to loose time for preparing introductory presentations, grading sheets and grading tables. These are done by the system after having specified the

course correctly. Once the instructor feel comfortable with the DSL, graphical editor or the Excel templates for specifying a course, he may easily generate his teaching material by a simple right click on the project explorer and clicking on generate Document.

A course certification sheet should be generated to illustrate the coverage of a learning standard of a program or a course. These standards can be extended internationally. Thus, we would be able to compare internationally different courses. Professors would easily present their courses in different countries or universities. People, who are using TESMA and familiar with our report, would have no doubts to understand the descriptions.

We want to generate the grading sheets and grading tables, which often takes a lot of time to construct by the instructors. After having specified the evaluation criteria and grading criteria correctly, the instructor may generate grading sheets for the students. These grading sheets are created for each test and for each student following course. An example of such a grading sheet can be found in chapter 9.6. The user has only to fill out the evaluation of the different criteria and the grade will be automatically calculated for the test of that student. After completing the grading, the instructor has to fill out the complete grading table. In that table, the final grade is calculated automatically after filling out the intermediate grades of the tests. Then, the instructor may submit the final grades to the program's committee for creating the student's grading report.

After having specified a course, the tool should generate a formulary for the institution's office. Often the institution's course registration office have their own naming conventions and needs, so that a formulary is needed for registering the course. These task should also be done by TESMA, so that the user save time and concentrate more on specifying his program. In Figure 4.8, we show a small part of generated report of the University of Luxembourg.

4.5 Quality, Dependability, and Maintainability

Our goal is to develop a DSL of high quality. We need to satisfy our requirements in order to improve the quality of the domain-specific language. We designed the TESMA DSL to be intuitive, customisable, and loosely coupled. We have chosen to design of the grammar using mainly keywords in natural language. Additionally, the educational institutions may use different naming conventions for their concepts than used in our approach. This forced us to design the grammar of our DSL to be customisable according to the user's needs. The institutions may choose their own naming conventions. Lastly, the rules of the grammar are loosely coupled, i.e. optional cross-references are mostly used instead of containment relations. Starting with the quality requirements of the domain-specific language:

4.5.1 Quality

- Readability

The readability requirement is very important for having an overview of the specified program. We designed the domain-specific language to be intuitive and comprehensible. The DSL uses mostly natural language for the specification. Thus, the language is very easy to read, since it uses words from the daily life. The specification can be read like a sentence. An example of a specification would be "hide Course required MICS2_33 belongs MICS". In natural language this means : A hidden required course with the reference MICS2.33 belongs to the MICS program. We tried to use keywords in natural language for declaring programs and

course, as well as their attributes. Since, the DSL is very intuitive with this approach, it improves the quality of the DSL. The DSL is very easy to read and to be understood.

- Longevity

Our domain-specific language is designed for a long life cycle. Institution's do not often change in their internal structure. They often keep their architecture for 10 to 20 years. Our domain-specific language is designed to be generic and adaptable to the institution's needs, so we can fit the needs for a long period. We plan to implement adaptable naming conventions of the DSL, so that the user may change the keywords used for specifying a course. The DSL is designed to be modular, which allows updates on the specification. The users may add or remove attributes, which they do not need. There is no restriction on the usage of attributes except of some few exceptions, which are necessary for declaring a course e.g. description, credits, course instructor. These exceptions are set up by the course registration office, for registering the course at the institution. The Longevity requirements is so guaranteed, since it depends on the usability and maintainability of the DSL. These requirements, we will discuss later on in this section.

- Simplicity

The Simplicity requirement is guaranteed, since we are using natural language and structure the grammar in a implicit way. The user only has to declare a class type, and specify the attributes by putting the keywords and its corresponding value to it. These values are in general other class types, pure text or natural numbers. The DSL is so designed to be very simple and usable by almost every person. Additionally, since the courses can be specified in Excel, the user may fill out the cells in an Excel table and import it to the textual editor. The Excel sheet will be then translated to the domain-specific language. The user can than easily follow his specification in the textual editor.

- Usability

The usability is guaranteed, if the domain-specific language is effective, efficient, accessible, and satisfactory. The TESMA grammar contains all information, which are needed to create a program specification. The model generated out of the specification is the bases for the generation and the graphical editor. If the program director specifies correctly his program, he may easily illustrate the specification in a graphical editor or generate the document. The user is encouraged to finish the specification in order to generate the documents and teaching material, which helps him to teach the course. Having a Domain-specific language, which is simple and readable, ensure the effectiveness even more, since the users feel comfortable with the DSL. Thus, this makes our domain-specific language very effective.

The DSL is designed , which differs from a general purpose language like JAVA. We do not have logical operators or loops, which needs mathematical knowledge to be understood. Our domain-specific language is based on a natural language description of programs. We simply use natural language, fill out the templates and the attributes we need by following the syntax. This makes the language very efficient, since the user only has to understand the syntax for specifying the course.

We want to guarantee accessibility. If we look the design of the textual editor, the user has a project explorer, where he groups the instances according to their class

type (institution, program, course). The specified of an class type instance may so easily find his specified instance by following the folder hierarchy. The same hierarchy is kept for the report. The user may find the generated LaTeX files in the same way as for the DSL files. The grammar is also designed to be readable. Attributes are grouped in categories. Usually, they start with default attributes for a general description and continue with more specific information like lecture contents. Again, the DSL uses natural language, which makes it easy to find data.

We did not measure the satisfaction attribute, however due to the high demand on such educational program specification tools, we think that our DSL will be used. It is kept simple, readable, maintainable, modular and usable, so that satisfactory has do be guaranteed. A completed specification of a programs and course allows the user to generate numerous documents such as teaching material, introductory slides, grading tables, and so on. Before, these documents have been created manually. The instructor saves time, and follows directly the methodology of the institution for specifying a course.

Since, all 4 requirements are fulfilled our domain-specific language is usable. The DSL is designed to motivate people from the computer science domain and other people to use the tool for specifying their programs. It improves their specification, generates a lot of documents at once, and the users save a lot of time. Additionally, it improves the communication between the course registration office and the instructors, since they have to follow the institution's rules for specifying the course. These rules can be configured and applied to TESMA. Additionally, using Excel and importing it to the textual editor, makes the DSL even more usable, since people who are not familiar with Eclipse, may use Excel to specify their courses. Thus the DSL recognises these tables and the specification can be done.

- Genericity

To guarantee the Genericity requirement, we propose several features. The institution may configure the TESMA grammar according to their own naming conventions. We do not want to restrict the users to follow our naming conventions, since it would not improve the quality of our DSL. We designed our model to user a lot of reflexive relations. Institutions can create sub institutions, as well as programs with sub programs. This help the institutions to specify their architecture and be more clear in their specification. The readers of the specification will directly know to which program they are inscribed in, and in which institution it is taught. Additionally, we tried to not use containment relations, which would make the DSL unreadable and overloaded. It helps to separate the specification and allows multiple users to use it. Without containments a program director may specify the program, while the instructors are specifying their course. The instructor only have to submit their course files to the program director, who imports them to the project. This modularity ensures the genericity.

In general satisfying these attributes improves the quality of the domain-specific language. We see that the domain-specific language allows multiple user to use it, it allows the usage of Excel, and it is usable by a large group of people. Having a simple, intuitive and modular DSL guarantees a high quality of the DSL.

4.5.2 Dependability

In this section, we will talk about the Dependability attributes, which we want to follow. These requirements are for the moment not all implemented, since we developed only a first beta version of the tool. We think that following these requirements will produce a dependable tool.

- Integrity

Using a Model based approach helps to ensure the absence of incorrect system modification. Since, the graphical, and textual editor are based on a underlying-core library, a modification on the library, modifies the DSL and the graphical editor. Thus, since we guarantee the genericity requirement, Integrity needs to be ensured. Allowing the institution's to adapt the DSL to their naming conventions using a configuration ensure the integrity. The user are guided by changing the attributes in the config file. These changes are then applied to the system. The textual editor uses a validator for performing checks on the DSL. Thus, the modifications are checked by the system and the user only has to follow the rules or corrected the erroneous lines.

- Availability

We are using Xtext, Sirius, and Xtend for our tool. These 3 frameworks are stable Eclipse plugin. The user can use at any time the Eclipse Environment for specifying his course. It requires a machine with a Java Runtime Environment installed on it, to run Eclipse. In case of problems, the instructor can use Excel to specify his program and import it to Eclipse later on. The instructor could also submit the Excel file to the program director, who can import it inside TESMA. Thus, these design decisions ensures the availability.

- Reliability

In case of syntax errors, the user is notified by error and warning messages. Not all validator rules are currently implemented. We will extend the validator in a future version in order to guarantee the reliability. Additionally, functions needs to be optimised in order to reduce the execution time of the generation. High CPU usage may cause problems on some machine, which we want to be usable too.

- Safety

Since every actors specify his own instance (program, course), the system is kept safe. The program director may collect all specifications and import them to his Environment. Using backups of their specification ensures no data loss. This would be the worst case scenario. In general, the data is stored inside the workspace. Except of a hard-drive failure, we do not see another way of loosing the data out of the workspace. The workspace of the Eclipse Environment contains the whole specification done in the editor. Thus, we ensure the safety requirement.

- Confidentiality

The Confidentiality requirement is not considered for the moment. We plan to assign logins and user roles to the actors by using a web application. The web application will be discussed in the Future Work section.

4.5.3 Maintainability

In the last section, we will discuss the maintainability requirements. The maintainability requirements is a fundamental requirement for having a DSL of high quality. We talked a lot already about the quality of our DSL. We split the maintainability in several sub-requirements to show why TESMA is maintainable.

- Modularity

The Modularity requirement is very important for us, since we want to ensure the genericity. We avoid the usage of containment relations, which allows the user to specify the courses in different files. The users can specify the courses in parallel and send the different modules to the program director who can import them to the system. Those users, who are using Excel, specify the course in these tables and send it to the user. TESMA converts the Excel files to the DSL file and inserts them to the system. Thus, modularity is ensured.

- Functionality

Having the appropriate functions, for importing, exporting, validator and generator guarantees the functionality requirements. It is very easy to follow the implementation and the way how the validator and generator is organised. These functions are very useful to structure the code. The validator is split in different functions for creating the reports of the different instances. These functions are all executed in a "mother function", which covers the main body of the documents.

- Scalability

We structure the specification of the courses in different files and folders. The user has an overview of the files using the project explorer. The system is scalable, since the specifier may decide in which files he wants to specify his courses. For example, the specifier of a course does not necessarily need to see the specification of other instructor's. The specifier usable to specify the course in its own environment. Only the program director needs the full specification of the program with all the courses. Here the amount of files can pass to 70-80 files for a program. These files depend on the number of courses. It is not very hard to group all the data in the project explorer since it is very intuitive. Thus, TESMA guarantees the scalability requirement.

- Flexibility

As already said before, the flexibility requirement is ensured. We want to allow the institutions to introduce their own naming conventions to the system. Using Excel, own Eclipse Environment and importing and exporting files, makes the system very flexible and usable. TESMA is in deed designed to be flexible.

In this chapter, we presented the technologies used for developing TESMA. We presented the tool, which is composed of a textual editor, graphical editor and the documentation generator. We described the realisation of the concept and the implementation by defining the class types realised with the help of the grammar of the TESMA DSL. We finally talked about the quality of our DSL according to the realisation. We motivated our DSL according to the requirements, we have defined before. In the next chapter, we will describe the domain-specific language with an sample specification of a master program at the University of Luxembourg.

Chapter 5

Case Study

In this chapter, we present an example of a fully specified course inside a program at the University of Luxembourg. The course, which is named Software Engineering Environment, is part of the Master in Information and Computer Sciences. We will describe the procedure of specifying an institution, program, and course using our tool. We will show you screenshots of the tool by mainly using textual editor with the domain-specific language. In addition, we will talk about the quality of our domain-specific language on this concrete example.

5.1 TESMA Program Specification tool

We will specify a course of the University of Luxembourg using the TESMA tool suite. The course will be specified using the textual editor, since the graphical editor is not ready in the current version of TESMA. Nevertheless, we are able to show the quality of our domain-specific language, by specifying a complete example. The Software Engineering Environment course is taught inside the Master in Information and Computer Sciences program proposed at the University of Luxembourg. In order to demonstrate the usage of our DSL, we will specify that institution, that program, and that course. With the help of the specification, we will generate a sample document containing the descriptions. We will now start by the specification of the University of Luxembourg.

5.2 University of Luxembourg

In order to cover the genericity, and modularity requirements, we do not have any order for specifying a program. We will start by specifying the institution in the view of an institution director. In this section, we will describe the institutions, and continue with the specification of the institutions using our domain-specific language.

5.2.1 Description

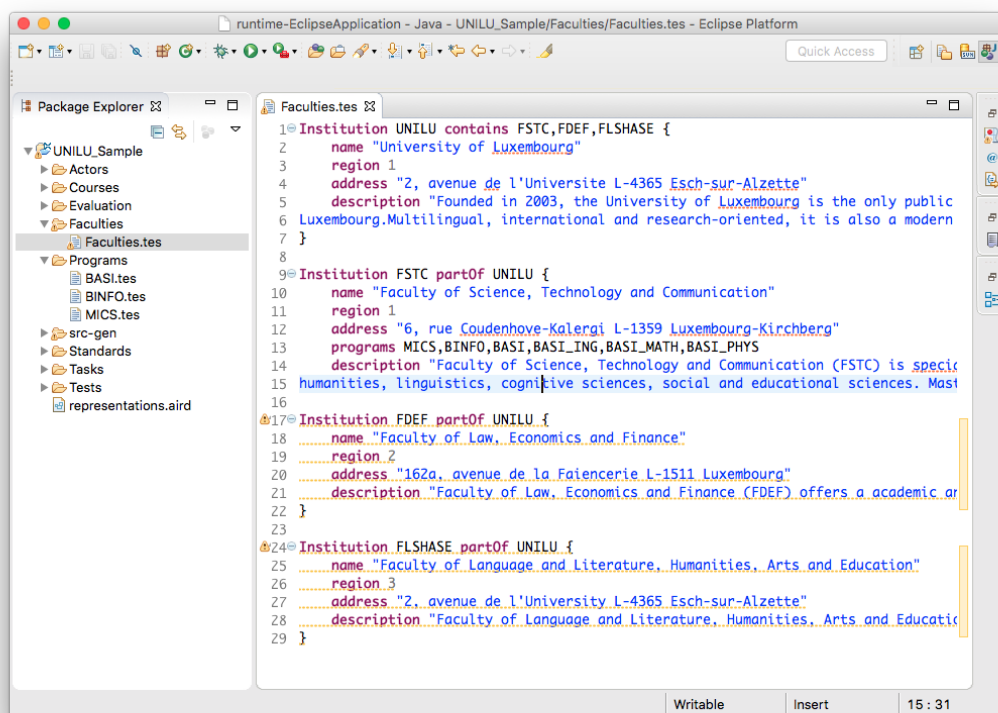
The University of Luxembourg (UNILU) is an educational institution founded in 2003. It is the only public university of the Grand Duchy of Luxembourg. The University of Luxembourg is a modern, research-oriented, and multilingual institutions. Courses are taught in different languages. It consists of three sub-institutions called faculties. The faculties are the following ones :

- *Faculty of Law, Economics, and Finance* (FDEF) offers an academic, and research environment. The FDEF has about 2,700 students, who are enrolled in three Bach-

elor programs, eleven Master programs, and three professional programmes, and two Doctoral Schools. The FDEF organises undergraduate courses in law, economics, management,, and IT management. Their Masters specialise in European Law, Financial Economics, Banking, and Finance, Security Management of Information Systems, Accounting, and Audit, Wealth Management,, and a Master in Entrepreneurship, and Innovation.

- *Faculty of Science, Technology, and Communication (FSTC)* is specialised in the fields of Mathematics, Physics, Engineering, Computer Science, and Life Sciences. The FSTC has two mission : teaching, and research, where they seek to generate knowledge, and educate the students to better understand, explain, and advance society, and environment. The FSTC offers bachelor's majors in sciences, engineering, computer science, and life sciences with a choice of bachelor,, and master's degrees in Information, and Computer Sciences, Integrated Systems Biology, Sustainable development, Engineering Sciences, Mathematics, Condensed Matter Physics, Master en management of security of information systems,, and a European Master in Small Animal Veterinary Science. The faculty also offers training in general medicine.
- *Faculty of Language, and Literature, Humanities, Arts, and Education* is specialised in humanities, linguistics, cognitive sciences, social, and educational sciences. Master studies include Contemporary European History, Psychology, Mediation, Gerontology, Cross-border Communication, and Co-operation, Philosophy, Geography, and Spatial Planning, European Governance, a Master in Learning, and Communication in Multilingual, and Multicultural Contexts,, and Luxembourgish studies.

After having explained the different institution, we want to specify the institutions. In Figure 5.1, we show the specification of the institution in the TESMA tool using the domain-specific language.



```
1 Institution UNILU contains FSTC,FDEF,FLSHASE {
2   name "University of Luxembourg"
3   region 1
4   address "2, avenue de l'Universite L-4365 Esch-sur-Alzette"
5   description "Founded in 2003, the University of Luxembourg is the only public
6 Luxembourg.Multilingual, international and research-oriented, it is also a modern
7 }
8
9 Institution FSTC partOf UNILU {
10  name "Faculty of Science, Technology and Communication"
11  region 1
12  address "6, rue Coudenhove-Kalergi L-1359 Luxembourg-Kirchberg"
13  programs MICS,BINFO,BASI,BASI_ING,BASI_MATH,BASI_PHYS
14  description "Faculty of Science, Technology and Communication (FSTC) is specia
15 humanities, linguistics, cognitive sciences, social and educational sciences. Mast
16
17 Institution FDEF partOf UNILU {
18  name "Faculty of Law, Economics and Finance"
19  region 2
20  address "162a, avenue de la Faiencerie L-1511 Luxembourg"
21  description "Faculty of Law, Economics and Finance (FDEF) offers a academic ar
22 }
23
24 Institution FLSHASE partOf UNILU {
25  name "Faculty of Language and Literature, Humanities, Arts and Education"
26  region 3
27  address "2, avenue de l'Universite L-4365 Esch-sur-Alzette"
28  description "Faculty of Language and Literature, Humanities, Arts and Educatio
29 }
```

Figure 5.1: TESMA specification of an institution

5.2.2 Specification

We will now specify our institution, the University of Luxembourg. The University of Luxembourg is the mother-institution containing 3 sub-institutions. These institutions are related together via a reflexive relation. A mother-institution may contain several sub-institution. The institution are specified in a similar way. The user has to declare an institution, and fill out the attributes. In the following example, we specify the University of Luxembourg containing three sub-institution:

```
Institution UNILU contains FSTC,FDEF,FLSHASE {
  name "University of Luxembourg"
  region 1
  address "2, avenue de l'Universite L-4365 Esch-sur-Alzette"
  description "Faculty of Science, Technology, and Communication} (FSTC) is
    specialised in the fields of Mathematics, Physics, Engineering,
    Computer Science, and Life Sciences. The FSTC has two mission :
    teaching, and research, where they seek to generate knowledge, and
    educate the students to better understand, explain, and advance
    society, and environment. "
}
```

The relations to the sub-institutions are represented in an enumerated list after writing the keyword "contains". We simplify the description, and specify only the general attributes. These attributes are intuitive, and can be followed easily by inspecting the source code. In the following example, we will specify the sub-institution FSTC, which is part of the University of Luxembourg, and is specified in the same way as the mother institution. The only difference is that we use the keyword "partOf" to relate a sub-institution to a mother institution. It is important to know that a sub-institution may also have a few sub-institutions.

```
Institution FSTC partOf UNILU {
  name "Faculty of Science, Technology, and Communication"
  region 1
  address "6, rue Coudenhove-Kalergi L-1359 Luxembourg-Kirchberg"
  programs MICS,BINFO,BASI,BASI_ING,BASI_MATH,BASI_PHYS
  description "Faculty of Science, Technology, and Communication (FSTC) is
    specialised in the fields of Mathematics, Physics, Engineering, Computer
    Science, and Life Sciences."}
```

The specifier has simply to declare the sub-institution as an institution, and relate it to the mother-institution. The programs attribute describes the relation of the program. An institution may contain several program. In our case, the FSTC contains different programs, where we will discuss the specification of the Master in Information and Computer Sciences (MICS) program. In Figure 5.1, we illustrate the 4 institutions with the relations, and the attributes. The validator checks if a program is specified in a sub-institution. In case, the specifier did not relate the institution to a program a warning highlight the class type in order to notify the user that no program is linked to the institution.

5.2.3 Institution director's role

The only actors, who are specifying the institution are the institution director or his secretary. They are specifying the current structure of the institution using the DSL. The institution director looks then for program directors for specifying a program.

5.3 Master in Information and Computer Sciences

After having specified the different institutions, the course director contacts his program director's to define some programs. The University of Luxembourg has many Bachelor, Master, and PHD programs e.g. Bachelor of Engineering Science (BASI), Bachelor in Informatics (BINFO), Master in Information, and Computer Sciences (MICS), and so on. In this section, we will specify the program called MICS, which is part of the FSTC institution. We will first describe the Master, and specify it with our domain-specific language.

5.3.1 Description

The Master in Information and Computer Sciences program is a continuation of the Bachelor studies. Students have the possibility to continue to a PHD program after finishing this Master. This master begins with an orientation meeting during the first week, where all new students meet their professors, and their new colleagues. After completing the week, the students start their first semester, which is mandatory for all. The semester is dedicated to the fundamentals of computer science. By the end of the first semester, the students have to select courses based on one or more profiles to continue the Master. These profiles are specialisations in different areas of computer science like security, software, intelligent systems, and so on. We will specify a list of modules sorted per term, which can be shown to the students. These modules contains then the courses proposed by the program. It is possible to divide the terms into sub-terms containing all the courses of the different profiles.

5.3.2 Specification

We will now start specifying the master program, MICS. In Figure 5.2, we illustrate the specification of the MICS program in the TESMA tool.

Declaring a program means specifying his ID, and assigning the program to an institution. In our case, we assign the MICS program to the FSTC sub-institution. Remember the FSTC is a sub-institution of the UNILU mother-institution. Then, the program director starts specifying the general attributes like name, isced level, description, costs, and so on. These attributes are either natural numbers or string, resp. enumerates lists of these types.

```

Program MICS in FSTC /*rnctInstitution relation*/ {
  name "Master in Information, and Computer Science"
  isced 7,4,8
  description "This master is a continuation of the Bachelor studies as a
    first step towards the PhD. MICS starts with an orientation meeting
    where all new students get to know the professors, and other students.
    The first semester is then mandatory for all: it is dedicated to the
    fundamentals of computer science..."
  programdirector guni /*rnctInstructor relation*/
  costs "200"
  language "english","german"
  email "mics@uni.lu"
  weblink "https://www.uni.lu/MICS"

```

After completing the general attributes, and assigning the program director to the program, the specifier starts specifying the different terms. The master is composed of

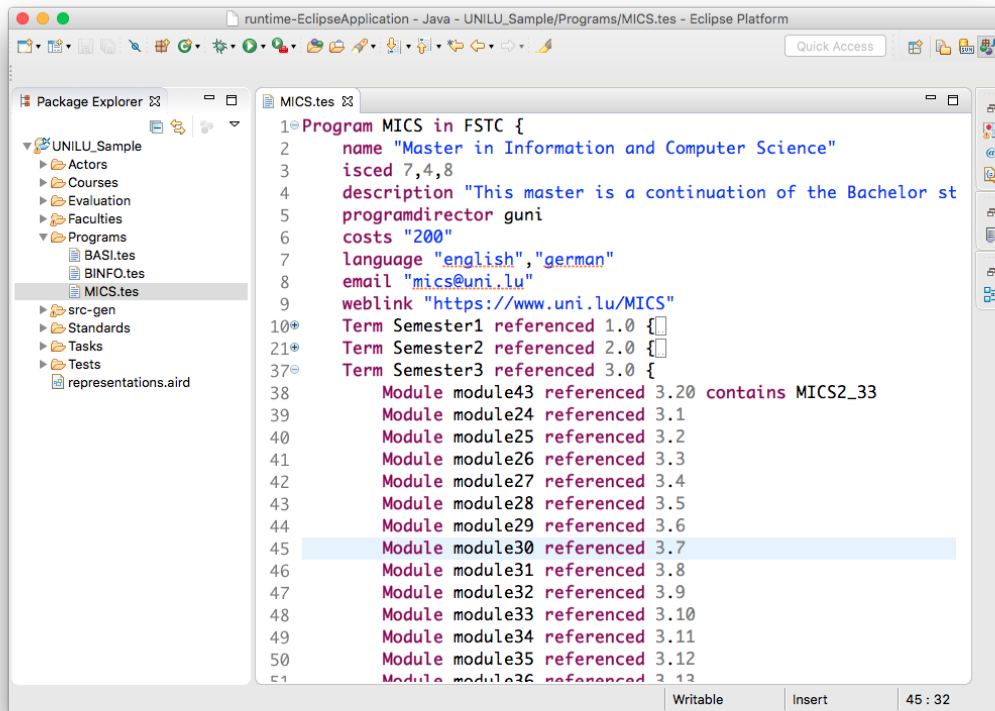


Figure 5.2: TESMA Program Specification - MICS

4 semester, with plenty of courses. We will show you a part of the specification in order to illustrate the hierarchy.

```

Program MICS in FSTC /*rnectInstitution relation*/ {
  /*... general attributes...*/
  Term Semester1 referenced 1.0 {
    Module module1 referenced 1.1
    Module module2 referenced 1.2
    ...
  }
  Term Semester2 referenced 2.0 {
    Module module10 referenced 2.1
    ...
  }
  Term Semester3 referenced 3.0 {
    ...
    Module module43 referenced 3.20 contains MICS2_33 /*rnectCourse relation*/
    ...
  }
}

```

A term, and a module are declared by specifying its ID, and giving a reference to it. The specifier, an instructor, may choose between the following data types for defining the programs contents, term, module, course. The specifier of a course is not forced to use terms, and modules for assigning courses to his program, he may directly assign the courses in a attributes called courses containing a list of courses. The instructor has also the possibility to specify only the modules containing the programs. In all

cases, the instructor has to chose one of the 3 possibilities, which is his only restriction. After completing the specification of the terms, modules or courses, the specification of a program is finished.

After finishing the specification of a program the specifier may assign a standard to his program. The specifier needs to select a fields of the standard to define the field coverage. We will talk about this process in the certification section of a program.

5.3.3 Certification

For specifying the certification, an appropriate standard needs to be chosen by the institution. We will chose an international learning standard in software engineering called SWEBOK. SWEBOK summarises a big part of the knowledge areas in software engineering. We first need to import the SWEBOK standard inside the textual editor. The result is the following specification :

```
Standard SWEBOK {
  Field s_1 (1,"Software Requirements")
  Field s_1_1 (1,1,"Software Requirements Fundamentals")
  Field s_1_1_1 (1,1,1,"Definition of a Software Requirement")
  Field s_1_1_2 (1,1,2,"Product, and Process Requirements")
  Field s_1_1_3 (1,1,3,"Functional, and Nonfunctional Requirements")
  Field s_1_1_4 (1,1,4,"Emergent Properties")
  Field s_1_1_5 (1,1,5,"Quantifiable Requirements")
  Field s_1_1_6 (1,1,6,"System Requirements, and Software Requirements")
  Field s_1_2_1 (1,2,1,"Process Models")
  Field s_1_2_2 (1,2,2,"Process Actors")
  Field s_1_2_3 (1,2,3,"Process Support, and Management")
  Field s_1_2_4 (1,2,4,"Process Quality, and Improvement")
  Field s_1_3_1 (1,3,1,"Requirements Sources")
  Field s_1_3_2 (1,3,2,"Elicitation Techniques")
  Field s_1_4_1 (1,4,1,"Requirements Classification")
  Field s_1_4_2 (1,4,2,"Conceptual Modeling")
  Field s_1_4_3 (1,4,3,"Architectural Design, and Requirements Allocation")
  Field s_1_4_4 (1,4,4,"Requirements Negotiation")
  Field s_1_4_5 (1,4,5,"Formal Analysis")
  Field s_1_5_1 (1,5,1,"System Definition Document")
  Field s_1_5_2 (1,5,2,"System Requirements Specification")
  Field s_1_5_3 (1,5,3,"Software Requirements Specification")
  ...
}
```

The SWEBOK standard contains more than 400 fields spread up in different knowledge areas. These standards are specified in separate files in the Standards Folder. After having specified them, the user has to define the FieldCoverage of the standard, and assigning the Bloom taxonomy to the fields. We illustrate now the FieldCoverage specification, which is added to the program after specifying the terms.

```
FieldCoverage SeeSWEBOK (0/0/0) {
  SWEBOK.s_1 (3/0/3),
  SWEBOK.s_1_1 (3/0/3),
  SWEBOK.s_1_1_1 (3/0/3),
  SWEBOK.s_1_1_2 (3/0/3),
  SWEBOK.s_1_1_3 (3/0/1),
  SWEBOK.s_1_1_4 (0/0/0),
  SWEBOK.s_1_1_5 (0/0/0),
}
```

```

    SWEBOOK.s_1_1_6 (1/0/2),
    SWEBOOK.s_1_2_1 (3/0/2),
    SWEBOOK.s_1_2_2 (2/0/2),
    SWEBOOK.s_1_2_3 (1/0/2),
    SWEBOOK.s_1_2_4 (0/0/0),
    ...
}

```

This procedure will also be used for assigning an international standard to a course. A program coverage usually contains the union of Field-Coverages of all course specified in the program.

5.3.4 Actors

In this sub-section, we will describe the roles of the actors during the program specification. The following actors are concerned by the program specification:

- The *Institution director's role* is the dean of the University of Luxembourg. He is specifying the institution, and validating the program description.
- The *Program director's role* is the study director of the program, and specifies the program. He is collecting all the course specification, and assigns them to the terms. He is preparing the certification phase, and assigns a field coverage to the program.
- The *Instructor's role* is the teacher of a course. After completing the course specification, they submit the specification to the program director, who assigns it to a term or module.
- The *Student's role* is the participant of a program. They want to be informed about the program, and inscribe for a program to follow the courses.
- The *Quality manager's role* is the validation of the certification done by the program director. He is certifying the program according to an internal standard or in case of an international standard he is contacting the international committee of the learning standard to get a certification.

All these actors are important for the specification of a program, and needs to cooperate together in order to complete the specification. We will now continue with the specification of a course using the TESMA DSL.

5.4 Software Engineering Environment

In this section, we will present to you the specification of the Software Engineering Environment course. We will start with a small description of the course and its content and specify these information using our domain-specific language.

5.4.1 Description

The Software Engineering Environments SEE is a course taught at the University of Luxembourg in the FSTC sub-institution. Software engineers need means for quality engineering of the software and systems. SEE concentrates on improving the specification, design and implementation of the means necessary for supporting software engineering

when joining a development team. The SEE course is organised in the following way: The lecture starts with a general overview of existing tools and describes the main technologies on which those tools are build. These tools are mostly based on the Eclipse framework with rich textual editing for domain specific languages using Xtext and advanced graphical editing using Sirius. There are low level technologies used in this lecture like Java, EMF, SVN, Jira. In the second part of this lecture, the students are working individually or in group on the definition and design of a software engineering environment during practical sessions. The goal is to set up the student to a context of a concrete engineering project. The course starts with an introduction on concepts and state of the art tools and practices. After that a directed and practical sessions will be held weekly in which students will have to engineer a specific software engineering environment and present their progress.

The students are evaluated based on oral presentations of their progress and a final presentation done at the end of the semester.

```

1 Course required MICS2_33 belongs MICS {
2   name "Software Engineering Environments"
3   reference 2.33
4   academicyear 2016/2017
5   term MICS.Semester3
6   module MICS.Semester3.module43
7   hoursPerWeek 2
8   totalHours 120
9   description "Software engineers need means for quality engineering..."
10  credits 5
11  languages "french", "english"
12  weblink "https://www.uni.lu/MICS/SEE"
13  coursemoderator guni
14  //Teaching and Students
15  boards SEEBoard
16  promotions MICS_SEE_CLASS
17  organisation org1 typeof lecture {
18    instructor guni : hours 30,weight 1,language "english, french"
19    instructor beri : hours 30,weight 1,language "english, french"
20    instructor alca : hours 30,weight 1,language "english, french"
21  }
22  Period (Semester,1,0) start 17.09.2015 end 17.12.2015 {
23    Period (Lecture,2,1) start 17.09.2015 end 08.10.2015 {
24      Period (Lecture1,3,1) start 17.09.2015 {tasks t1a}
25      Period (Lecture2,3,2) start 24.09.2015 {tasks t2a,t2b}
26    }
27    Period (CheckPoint3,2,2) start 17.12.2014 {tests oralCheckpoint3}
28    Period (FinalExam,3,1) start 15.01.2015{tests finalExam}
29  }
30  FieldCoveraae SeeSWEBOOK (0/0/0) f

```

Figure 5.3: TESMA Course Specification - SEE

5.4.2 Specification

In Figure 5.3, we illustrate the specification of the SEE course in the TESMA textual editor. The instructor starts by declaring the course. The course declaration consists of specifying the course id, the course-type and the program relation. The course-type defines whether a course is required or not. The program relation illustrated to which program the courses is belonging to. In our case, SEE is a required course and belongs to the MICS program. Now the specifier (instructor) starts by specifying the general

attributes like name, reference, academic-year, and so on. The instructor assigns the course to a module or a term. An example of such a specification would be :

```
Course required MICS2_33 belongs MICS {
  name "Software Engineering Environments"
  reference 2.33
  academicyear 2016/2017
  term MICS.Semester3
  module MICS.Semester3.module43
  hoursPerWeek 2
  totalHours 120
  description "Software engineers need means for quality engineering..."
  credits 5
  languages "french","english"
  weblink "https://www.uni.lu/MICS/SEE"
  coursemoderator guni
}
```

After specifying the general course description, the specifier starts to describe the general lecture organisations. He assigns students and teachers to the lecture as well as the teaching team. In the case of SEE, 3 instructors are teaching the course with equal evaluation weights.

```
Course required MICS2_33 belongs MICS {
  ...
  /*Teaching and Students*/
  boards SEEBoard
  promotions MICS_SEE_CLASS
  organisation org1 typeof lecture {
    instructor guni : hours 30,weight 1,language "english, french"
    instructor beri : hours 30,weight 1,language "english, french"
    instructor alca : hours 30,weight 1,language "english, french"
  }
  ...
}
```

The instructor starts specifying the lecture organisation by specifying the term into a period from the begin of the semester to the end of the semester. Then, he starts specifying sub-periods inside the semester. The lecture is divided into 3 timing parts, lecture, practical1, practical2.

```
Course required MICS2_33 belongs MICS {
  ...
  Period (Semester,1,0) start 17.09.2015 end 17.12.2015 {
    Period (Lecture,1,1) start 17.09.2015 end 08.10.2015 {
      Period (Lecture1,1,2) start 17.09.2015 {tasks t1a}
      Period (Lecture2,1,3) start 24.09.2015 {tasks t2a,t2b}
      ...
    }
    Period (CheckPoint1,2,1) start 07.10.2015 {tests oralCheckpoint3}
    Period (Practical1,2,1) start 08.10.2015 end 08.11.2015 {...}
    .../*Checkpoint 2 & 3 and Period Practical 2*/...
    Period (FinalExam,3,1) start 15.01.2015{tests finalExam}
  }
  ...
}
```

The specifier is free to choose how precise he wants to specify the lecture organisation. In our case, we were very precise, since we specify each lecture individually by organising the semester according to the lecture timing slots. The specifier may only assign a list of tasks to the lecture without specifying any periods. Tests may be included also in periods or specified with a list of tests given in the course. These tests and courses are also specified by the specifier. We will now present some examples for tasks and tests. Tasks and tests are specified in the Tasks resp. Tests folder of the Package Explorer. Tasks are declared with its name ID. A task may be divided into subtasks. An example would be that Lecture 1 covers task "t1a". Looking at the specification of this task, we get the following source code :

```
//LECTURE 1 - Instructor's task 1
Task lecture t1a contains t1a1,t1a2,t1a3,t1a4 concerns instructor {
  description "Project session"
  instructors guni:2
  Artefact artefact1 (input,"Introductory presentations part 1 - slides
    0-199")
}

//LECTURE 1 - Instructor's subtasks
Task lecture t1a1 concerns instructor {
  description "Student/Teachers presentation"
}
Task lecture t1a2 concerns instructor {
  description "Lecture general information"
}
Task lecture t1a3 concerns instructor {
  description "projects information"
}
Task lecture t1a4 concerns instructor {
  description "icrash requirements"
}
```

From the source code above, Task t1a is a lecture task, which concerns the instructors. It describes, which topics are covered during the corresponding lecture. The tasks can be part of three different groups: course tasks, general tasks, and shared tasks. The shared tasks can be specific tasks (like course tasks) which are assigned to many courses. The general tasks are tasks, which describe for example the course participation rules can be specified in that folder. Figure 5.4 illustrates the textual editor, while specifying the tasks.

The instructor may also define some tests for his course. These tests can be specified with the same approach as for tasks. The instructor declares a test by specifying its ID, weight and task coverage. The tasks covered by a test are examined in the test. After that, the instructor specifies the description, the rationales and the grading categories and assigns them to the test. Here is a small example of a oral-checkpoints in SEE:

```
Test oralCheckpoint1 weight 1 covers t1a,t2a,t3a,t4a {
  description "Project Check Points - Oral presentation"
  rationale "10 minutes presentation - 5 minutes discussion"
  grading categories features,generalContent,nonFunc
}
Test finalExam weight 1 covers t5b_3,t6b_3,t7b_3,t8b_3,t9b_3,t10b_3,t11b_3 {
```

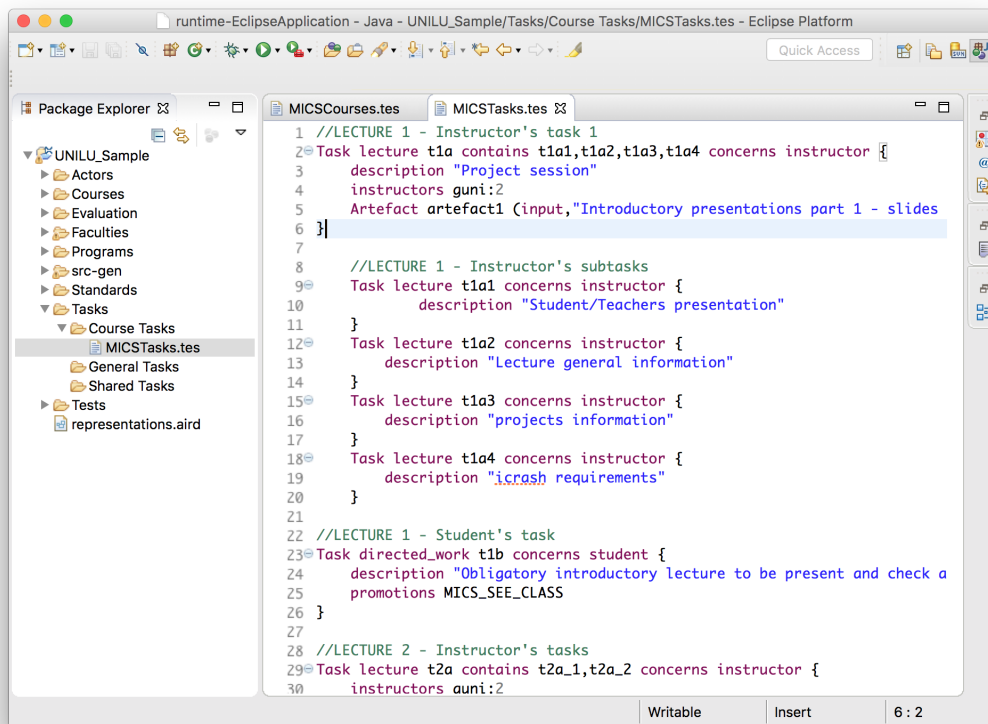


Figure 5.4: TESMA Task Specification - SEE tasks

```

Test oralexam weight 3 {
  //add weights of the different instructors
  description "Oral examen of 30 minutes, presenting the paper content"
  grading categories features,generalContent,nonFunc
}
Test completeproject weight 2 {
  description "The final version of the project with a README File
  containing all information needed for installing the plugin"
  grading categories generalContent,generalContentProd
}
Test paper weight 2 {
  description "The final version of the paper you have written while
  analysing your project with the SWEBOK criteria chosen at the
  beginning of the lecture"
  grading categories paper
}
}

```

The students are evaluated in 3 oral-checkpoints on the progress of the project and oral presentation of the complete project, the project, and a research paper written by students.

The instructors needs to specify the grading criteria of his course. These grading criteria are specified in the Evaluation Folder. Instructors may use criteria of other courses, if they cover the same idea. In our case, students are evaluated on their oral

presentation, project and paper. Therefore, we often define criteria about the content, structure and expression. Features are often describing the features of a tool. We show you now a part of the categories and criteria specified with the DSL.

```

/*LIST OF EVALUATION CATEGORIES */
GradingCategory generalContent weight 1 contains gradingCriteria
  expression,structure
  description "General content of the talk"
GradingCategory features weight 1 contains gradingCriteria
  presProdUser,funcCharProd
  description "Features"

/*LIST OF EVALUATION Grading Criteria */
//generalContentprod
GradingCriteria PresProdChar weight 1 points [0,4] with step 1
  description "Presentation of production characteristics"
GradingCriteria KeyDecMadeImpl weight 1 points [0,4] with step 1
  description "key decision made at implementation level"
GradingCriteria PresIntSof weight 1 points [0,4] with step 1
  description "Presentation of the internal software interactions and
    mechanisms"
//GENERALCONTENT
GradingCriteria expression weight 1 points [0,4] with step 1
  description "Expression"
GradingCriteria structure weight 1 points [0,4] with step 1
  description "Structure"

```

The Grading Categories contain the grading criteria. A weight is assigned to the categories and to the criteria. In our case, a criteria is evaluated with the following grades 0, 1, 2, 3, 4. These categories are then assigned to the corresponding tests.

After having specified the whole course, the instructor may certify his course according to a standard. Therefore, he uses exactly the same procedure as for the program certification and assigns a FieldCoverage to the course.

5.4.3 Actors

- The *Program director's role* is to validate and collect all the course specifications and relate them to the corresponding program.
- The *Instructor's role* is to specify their courses, they want to teach inside the program and send the specification to the program director. Additionally, he send the registration formulary to the administration office for registering the course. The instructor may also certify the course with the quality officer according to some standard.
- The *Student's role* is to follow the lectures of the course and complete the tasks. He wants to be informed about the course content and their evaluation.
- The *Quality Officer's role* is to certify the course according to some standard chosen by the institution.
- The *Secretary's role* is to register the course to the institution's database and adding the course description to the institution's website.

5.5 Document Generation

After completing the full specification of the institution, programs and courses, the institution may generate a full report. The documents generated are all in one PDF file. The report can be generated and printed out. Currently, the report only contains the description of the specification in natural language. There are three chapters with the course description : institutions, programs, and courses. These 3 chapters contains the whole description done in the domain-specific language. Figure 5.5 illustrates the textual editor with the report generated by the textual editor.

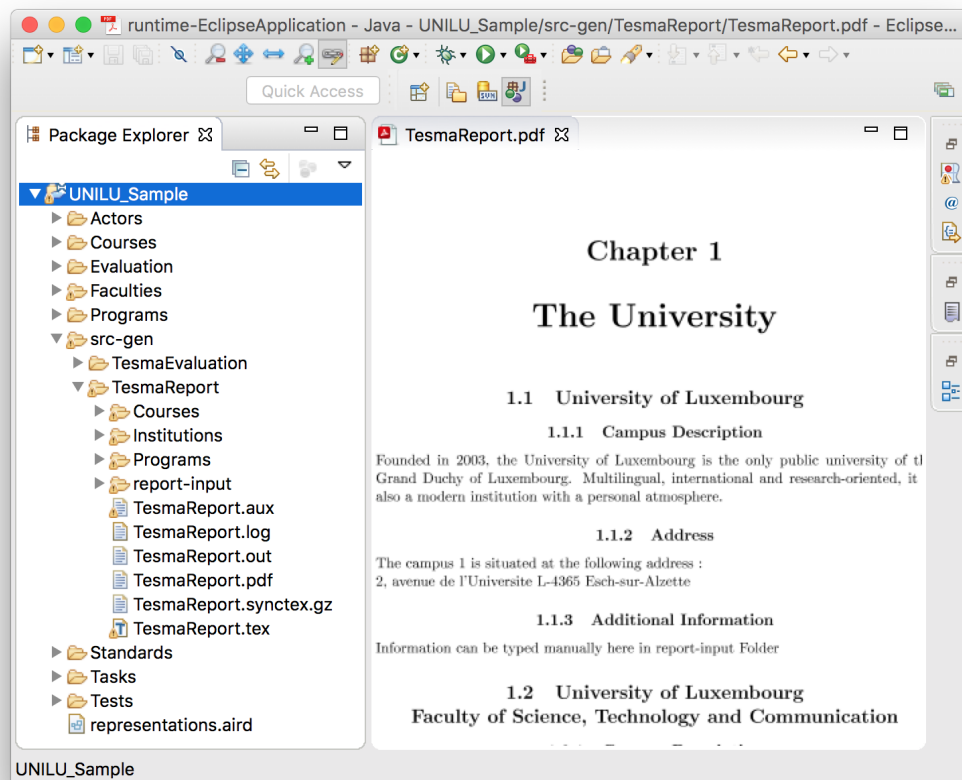


Figure 5.5: TESMA Documentation Generator

We did not implement the full generator, but we plan to generate registration sheets, introductory presentations, Grading sheets, and Grading tables in Excel format. The implementation of the Excel tables is already ready but not included in side the tool. In the Appendix we added some examples, which illustrate the design of the tables and grading sheets.

In order to launch the Generator, the specifier only has to do a simple right-click on the project and press Generator Report button. The Generator can be found in the Appendix at section 9.3.

5.6 Analysis of the quality of the DSL

In this section, we will talk about the quality of the domain-specific language. We will motivate the usage of our domain-specific language and show to you why we think that our DSL is of high quality.

The DSL is essential for the specification, since the textual editor generates a TESMA model, which can be graphically illustrated in the graphical editor. That is the reason why we need a DSL of high quality. The DSL does not only have to be usable by the users, but also a design of high quality in order to be represented graphically. If we have a look at our specification, we specified a part of the institutions, programs and courses in about 5-6 hours. Specifying a course will not take much longer using a DSL. Planning the course content and looking for information takes the most time. Once all information are collected and the course content is moreover stable, the DSL can be used to specify the course. The DSL may so be used as a guide for retrieving the information. The specifier may look at the attributes to find the information needed for creating a new course. In general, it is less time consuming to use the domain-specific language, since the specifier may effectively write the specification and generate documents needed for the course out of the specification. These documents may be course information sheets, course registration formulary, some grading sheets, grading tables, introductory slides, and so on.

Using the domain-specific language is very simple and intuitive. Declaring the different class types is very easy, since they are all similar to each other. The user has to know only the keywords. In a future version of TESMA templates will help the user to specify the courses even faster. Then the user needs only to fill out the declaration attributes (name, ID, and relation) and add the needed attributes to the specification. In general, it consists of four process : 1) Declaring; 2) General Specification; 3) Specific specification; 4) Certification. First the specifier declares the instance, then he describes generally the instance. Finally, the specifier continues specifying the specific information like the courses in the program or the lecture organisation. After completing these 3 steps, the specifier may certify his instance with the quality officer.

Concerning the syntax and structure of the DSL, we designed the DSL to be accessible, readable and intuitive. The DSL is designed using natural language, which facilitates the usage for non-computer scientist. There are no operator to use like loops or if clauses, which are mainly used in general purpose languages. The specifier has only the fill out the template and eventually add some attributes. This guarantees the simplicity. Due to our Package Explorer and structure of the attributes inside an instance, we guarantee the readiness an accessibility of information. General attributes are specified first, then specific attributes and then the certification. Due to our hierarchical structure of some attributes, the specifier may easily read his specification. Looking at the periods of a course is very intuitive and logically ordered in time. These design decisions guarantees our accessibility and readability requirements. In case a user does not want to specify a course in very deep detail, he's able to provide simple lists of task, which will be done during the semester. These lists are usually unordered. This scenario forces us to provide a flexible DSL, which can be used by different specifiers. Due to the simplicity of our DSL and the motivation to generate all the needed document, the DSL is designed to be usable. It motivates the user to finish the specification.

Additionally, we need a maintainable domain-specific language. We see due to our design the domain-specific language is modular and very flexible. The specifier has several possibilities to design his course. Since, the DSL is modular the specifier may work in parallel on their specification. The small parts can later on be grouped in one project.

In order to reduce the amount of data, the specification can be split in several folders and files, which allows the user to access information pretty quickly. The future import function will help the user to specify the course without even using the domain-specific language.

Our motivation was to provide a DSL, which can be used by an large group of people, who are not computer scientists. We created a DSL with high quality. We have seen so far some positive points, which improved the DSL. We showed the requirements, which need to be followed in order to provide a DSL for education institutions, where people who are not from the computer scientist domain, should use the system. These institutions were interested in contributing for such a tool. This led us to write a paper, which we submitted to the ICAIT 2016 in Japan. The paper can be found in the appendix at section 9.5. The paper is accepted for the ICAIT 2016 conference at the University of Aizu. The conference will be held in October 2016.

5.7 Results

If we look at our analysis of our domain-specific language, we conclude that the design was chosen to obtain an effective and efficient domain-specific language. It is very constructive and intuitive. By looking at our example the DSL satisfies already our quality requirements, a large part of the maintainability requirements and a part of the dependability requirements. The future version of TESMA will cover more requirements, which will improve the quality of the DSL even more, without complicating the syntax. In general, we see that a educational domain-specific language of high quality means that we have a syntax, which is not complicated and can be used by a large group of people and having a number of functionalities. Concerning the dependability attributes, we already considered some of them like, integrity, availability and reliability by using the validator, templates and model outline. We will focus more on these requirements in a future version of TESMA to propose more validation rules.

To conclude our example from the University of Luxembourg, we think that our DSL is applicable on even more institutions. We saw that the quality requirements are mostly satisfied in the current Beta version and plans have been proposed for the future version. These requirements helps to design a DSL of high quality and to obtain a modular and generic DSL, which can be used by a large group of people.

We presented in this chapter an sample specification of a course at the University of Luxembourg. We showed the usability of our domain-specific language, and demonstrated how easy a program can be specified. We analysed the quality of our DSL according to the quality requirements. We will continue with the limitations of our tool.

Chapter 6

Limitations

In this chapter, we will present the limitations of the TESMA tool. These limitations are not only based on our tool but also on the environment. We will present now some of these limitations, and eventually propose some solutions or improvements.

We will first start discussing the limitation of the environment, where TESMA will be used. We see some problems, which we need to solve in order to get a high usage rate of TESMA. Since, we are focusing on a large group of users, who will use the domain-specific language for specifying the course, we have different many different point of views in how to specify a program. Some people will simply refuse to use such a tool or methodology for specifying a course. People often do not want to change their habits and stick on their own procedures. Since, we work with a large group of people, the instructors come from different domains and some of them have difficulties in using computer systems. People who do not have experience at all with computers will not be able to use the tool. The minimal requirement for using the tool is the knowledge of Excel. In the worst case, when an user never has used a computer, the excel templates can be printed out and filled by hand. The secretary has done to enter the information inside Excel or directly using the DSL. For sure having less knowledge with software tools, will be a bottleneck an produce some problems. In summary, these information can than be imported inside tool without using the DSL.

We figured out after talking to some program directors, that there are often communication issues between program directors and instructors. Specifications often needs to be redone because information is incomplete or missing totally. TESMA helps to avoid such misunderstandings, Since, the user is guided by following the attributes of an instance.

The institutions need to define their own standard or choose an appropriate international standard. The international standards often do not cover all the topics proposed by a course, which forces institutions to extends the standards. Currently, TESMA allows the user to extension the international standard, but it does not propose any functionality to specify the coverage of the international standard. If a standard is extended, the tool will show to the user the coverage of the extended version, which could lead to misunderstanding. A redesign of the certification model, would be necessary to cover this limitation.

In general, we want to demonstrate the usability of the tool and motivate the people to use TESMA for specifying their programs. It should help the specifier to complete faster their tasks. The institution could force the staff to use TESMA for their specification, by proposing some trainings for using TESMA. However, our goal is not to force people using it, but to motivate them and to be satisfied with the tool.

We did not test if the tool is applicable at other institutions expect of the University of Luxembourg. We analysed the program structures at some other universities, and

developed a model, which could fit all of these. However, some scenarios needs to be specified with our tool. However, we talked to some institutions, who are interested in contributing for the development for such a tool.

Looking at our tool, we also see some technical limitation. We are working in an Eclipse Environment, which is not used by all the universities. Some people do not feel comfortable with Eclipse. Installations needs to be done and TESMA needs to be set up by the IT team, which is costly in terms of time.

The DSL is very intuitive and simple to use, however the user could write errors, which are not retrieved by the validator. We suggest that work needs be done on the validator rules. Writing a validator is a common problem in computer science, Since, it is a very difficult task. It is pretty impossible to see and cover all possible errors, which can occur while writing the specification, however we could cover a larger set.

In order to reduce the amount of big texts in the specification, more attributes needs to be proposed, which helps the user to specify the course even faster. We would optimise the specification time.

The graphical editor based on Sirius will not be usable by people without experience with Eclipse. We do not see problem for people from the computer science domain, Since, they are usually working with Eclipse, however for non-computer-scientist it will cause problems, Since, it is not very user friendly. It makes sense to propose a web application or use the Excel templates rather than the Eclipse Framework.

Finally, we need to examine the execution time of the generator, which currently needs about 1-2 seconds to generate the report. However, generating the full documentation of the specification with teach material, will take much more time. Therefore, we would need to optimise the execution time in order to satisfy the user. These execution times could pose a problem for slow machines. This limitation can be tested in a future versions of TESMA.

We see some work needs to be done in optimising the source code and extending the model in order to obtain a useful tool. The domain-specific language itself has already a good quality, but we could improve it in some points to avoid misunderstandings, like the definition of tasks, tests, and periods.

In the next chapter, we will continue with the future work, where we will present our ideas for updating and optimising TESMA.

Chapter 7

Future Work

In this chapter, we will talk about the future versions of TESMA. We planned some extensions, and further developments in order to provide a usable tool for the institutions to specify their programs. Additionally, we want to optimise the domain-specific language in the next versions. We will present the different ideas, and improvements, which we want to implement in the near future.

Currently, we developed a Beta version of the tool, where the TESMA domain-specific language can be used for specifying a course. However the future version should have a lot more functionalities and improvements.

We plan to iterate the process to stabilise the requirements, the tool design and the implementation.

The grammar of the domain-specific language needs to be finalised. Some attributes could be added for obtaining a complete description of the different instances. Additionally, the certification needs to be extended in order to be able to distinguish the coverage of an international standard between an extended standard. Optimisations on the task, test and period definition will be done in the future, for easily specifying the grading sheets and grading tables.

We also want to add more functionalities to the tool. The graphical editor will be implemented for a first version using Sirius. The graphical view will help the users, who are not familiar with DSL to specify their programs. Additionally, the import and export functionality of Excel files, will be implemented in the near future for covering a larger group of people, who will use TESMA. For the moment, the generator generates only a report of the specification, we need to implement the generation of the other documents. Meanwhile, the code needs to be optimised in order to obtain a fast generator.

We need to invest some time on working on the validator for covering more possible errors, which could occur while specifying a program. We will then improve the reliability of the system.

We also plan to develop an automatised generation of a web application for improving the maintenance and the usability of our tool. This web application should be used by the users for specifying the course. It is mapped to our textual language grammar. The usage of a web application will ease the specification of the programs. Additionally, a smartphone application would be interesting for informing the students about course contents or course descriptions. Students may download the application for getting information about the institution. The goal is to provide a user-friendly front-end, which is mapped to TESMA grammar.

Finally, we want to do a test phase to analyse the satisfaction of the usage of the tool. The testing phase should give us feedback on the quality of our tool. In a first period, the testing phase will be done at the University of Luxembourg, with professors

of different domains. After succeeding at the University of Luxembourg, we plan to test the tool at other institution. We plan to test the tool via a web application, where an institution may register themselves. In the web application, the institution obtains an account, where they may create accounts for the instructors, program directors, and so on for specifying their own programs.

Chapter 8

Conclusion

In this thesis, we presented the development of our tool, for educational program and course specification based on an textual domain-specific language. We presented how the quality of a domain-specific language can be improved for a large group of people. Our tool has been successfully used on a small example. We think that our tool will help the instructor to better specify and organise their courses. It will not only improve the internal communication for registering a new course or program at the institution, but also the communication between instructor and student. The student will have all the necessary information needed from the generated documents. Our textual editor is very intuitive and easy-to-use. It contains all information needed to specify a new course. Due to the design, the specifier uses natural language for specifying his course. Additionally, a number of universities are interested in such tools, which motivate us to work further on the tool, extend the model and develop a web application for it. After several meetings with the members of the faculty committee of the University of Luxembourg, we got a full support to develop such a educational program and course specification tool for the faculty. We motivate us to work even further and propose the tool for the whole university and to other institutions. In order to inform the other institutions and reasearch about our tool, we submitted a paper for the 2nd International Conference on Applications in Information Technology (ICAIT-2016) held in October 2016. The paper can be found at Chapter 9.5. We got the confirmation that the paper has been accepted for the conference at the University of Aizu in Japan, where we will present our work to the participants.

Bibliography

- [1] Arie van Deursen , Paul Klint , Joost Visser, *Domain-specific languages: an annotated bibliography*, ACM SIGPLAN Notices, v.35 n.6, p.26-36, June 2000 [doi>10.1145/352029.352035]
- [2] Beck, K., Gamma, E. 2003. *Contributing to eclipse ? Principles, Patterns, and Plugins*. Addison-Wesley Professional.
- [3] Bettini, L. 2013. *Implementing Domain-Specific Languages with Xtext and Xtend*. Packt Publishing Ltd.,Birmingham, UK.
- [4] Steinberg, D., Budinsky, F., Paternostro, M., Merks, E. 2008. *EMF: Eclipse Modeling Framework, 2nd Edition*. Addison-Wesley Professional.
- [5] Northover, S., Wilson, M. 2004. *SWT: The Standard Widget Toolkit, Volume 1*.Addison Wesley Pub Co Inc.
- [6] Lajmi, A., Martinez, J., Ziadi, T. 2014. *DSLFORGE: Textual Modeling on the Web*. MODELS Demonstrations 2014.
- [7] Hupponen T, Karlsson K, Laitinen J, Ojala O, Pirinen A, Seuranen E, Takkinen L. *TeXlipse*. Online, retrieved on April 2016 <http://texlipse.sourceforge.net/>
- [8] Oliver, A., Stampoultzis, G., Sengupta, A., Klute, R., Fisher, D. 2009. *Apache POI - the Java API for Microsoft Documents*. The Apache Software Foundation.
- [9] Fairley, R., Bourque P. 2014. *Guide to the Software Engineering Body of Knowledge - SWEBOOK*. Version 3.0. IEEE Computer Society Press Los Alamitos.
- [10] ACM/IEEE-CS Joint Task Force on Computing Curricula. 2013. *Computer Science Curricula 2013 - Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM Press and IEEE Computer Society Press. Online, retrieved on April 2016. <http://dx.doi.org/10.1145/2534860>
- [11] Kolovos, D. Paige, R., Kelly, T., Polack, F. 2006. *Requirements for domains-specific languages*. Department of Computer Science, University of York.Proc. 1st ECOOP Workshop on Domain-Specific Program Development (DSPD 2006).
- [12] Van Deursen, A. 1998. *Little Languages : Little Maintenance?, Volume 10*. Journal of Software Maintenance.
- [13] Barisic, A. Amaral, V. Goulao, M., Barroca, B. 2011. *How to reach a usable DSL? Moving toward a Systematic Evaluation*. Electronic Communications of the EASST, Volume 50.

- [14] Barisic, A. Amaral, V. Goulao, M., Barroca, B. 2011. *Quality in Use of Domain-Specific Languages : a Case Study*. PLATEAU 2011.
- [15] Barisic, A. Amaral, V. Goulao, M. 2011. *Usability Evaluation of domain-specific languages*. Quality of Information and Communications Technology (QUATIC).
- [16] Barisic, A. 2013. *Evaluating the quality in use of domain-specific languages in an agile way*. In Proceedings of the doctoral symposium at 16th international conference on model driven engineering languages and systems (MODELS), September, 2013, CEUR-WS.
- [17] Joeckel, G., Longhurst, M. *PDF syllabus builder: open source tool for online instructors, course developers and instructional designers*. Online, retrieved on July 2016. http://olc.onlinelearningconsortium.org/effective_practices/pdf-syllabus-builder-open-source-tool-online-instructorscourse-developers-and-i.
- [18] School Software Group. *Build Your Own Curriculum(BYOC): Evaluating a K-12 online curriculum management system*. ProQuest. Online, retrieved July 2016. <http://www.schoolsoftwaregroup.com/>.
- [19] Olsen, J., Meyer, J. Curriculum Design Tool. *Jump Rope*. Online, retrieved on July 2016. <https://www.jumpro.pe/features/curriculum-design-tool/>.
- [20] "Integrated School Management Software." Student Information System. Administrator's Plus, n.d. Web. 16 June 2016.
- [21] "Administration for Adult Education Organizations." PowerVista RollCall. N.p., n.d. Web. 16 June 2016.
- [22] Cornell University Center for Teaching Excellence. *Writing a Syllabus*. Online, retrieved on July 2016. <http://www.cte.cornell.edu/teaching-ideas/designing-yourcourse/writing-a-syllabus.html>.
- [23] University of Washington. *Course and Syllabus Design*. Online, retrieved on July 2016. <http://www.washington.edu/teaching/teaching-resources/preparing-to-teach/designing-your-course-and-syllabus/>.
- [24] Slattery, J. M. and Carlson, J. F. 2005. *Preparing an effective syllabus: Current best practices*. *College Teaching*, 53(4), 159-164.
- [25] University of Texas at Austin. *Develop Syllabus*. Online, retrieved July 2016. <https://facultyinnovate.utexas.edu/teaching/professional-development/preparing/plan/syllabus>
- [26] Altman, Howard B., and William E. Cashin. 2009. *Writing a Syllabus*. Idea Paper.
- [27] Hoycross, Jennifer. 2006. *Curriculum Mapping? An Essential Tool for Curriculum Development*. *Technology and Education* 4th ser. 17: 61-64.
- [28] University of Wisconsin-Madison. *Handbook for Creating Course Portfolios*. Wisconsin: Spring, 1997. Print.
- [29] Davis, B. G. 2009. *Tools for teaching (2nd ed)*. San Francisco, CA: Jossey-Bass.
- [30] Parkes, Jay. *The Purposes of a Syllabus*. EBSCO 2nd ser. 50: 55-61.

- [31] Keller, Carl E. 2014. *A National Survey On The Perceived Importance Of Syllabi Components: Differences And Agreements Between Students And Instructors In The Principles Of Accounting Course*. Academy Of Educational Leadership Journal 33rd. 18: 23-34.
- [32] Wikipedia . *K-12*. Wikimedia Foundation. Online, retrieved on June 2016. <https://en.wikipedia.org/wiki/K?12>
- [33] Forehand M.. *Bloom's Taxonomy - Emerging Perspectives on Learning, Teaching and Technology*. University of Georgia.2010.Print.

Chapter 9

Appendix

9.1 Appendix A : TESMA Grammar

9.2 Model

```
grammar lu.uni.lassy.messep.tesma.Tesma with org.eclipse.xtext.common.Terminals
generate tesma "http://www.uni.lu/lassy/messep/tesma/Tesma"
/* TESMA Model Starting Point */
Model:
  elements+=(ctInstitution | ctProgram | ctCourse |
             ctStandard | ctActor | ctTask | ctTest |
             ctCategory | ctCriteria
             )*;
```

9.2.1 ctInsitution

```
/*CLASS TYPES */
/**
 * Institution = Faculties, Universities, SubUniversities etc.
 */
ctInstitution:
  (hide=HIDDEN)? 'Institution' name=ID
  ('partOf' rncMotherInstitution=[ctInstitution|FQN])?
  ('contains' rncSubInstitutions+=[ctInstitution|FQN]
   ('rncSubInstitutions+=[ctInstitution|FQN])*)? '{'
  ('name' label=STRING)
  ('region' sepRegion=INT)
  ('address' address=STRING)
  ('programs' programs+=[ctProgram|ID] ('rncProgram+=[ctProgram|FQN])*)?
  ('description' description=STRING)?
  ('currentSituation' currentSituation=STRING)?
  ('gastronomie' gastronomie=STRING)?
  ('contact' contact=STRING)?
  '}',
;
```

9.2.2 ctProgram

```

/**
 * Program given in a institution
 */
ctProgram :
  (hide=HIDDEN)? 'Program' name=ID
    ('partOf' rncMotherProgram=[ctProgram|FQN])?
    ('contains' rncSubPrograms+=[ctProgram|FQN]
      (',' rncSubPrograms+=[ctProgram|FQN])*)?
    ('in' institution=[ctInstitution|FQN])? '{'
    ('name' label=STRING)
    ('iscd' iscdLevel=INT (',' iscdCategory=INT (','
      iscdSubcategory=INT)?)?)
    ('description' description=STRING)
    ('programdirector' rncProgramDirector=[ctInstructor|FQN])
    ('prerequisites' prerequisites=STRING)?
    ('requisites' requisites=STRING)?
    ('costs' cost=STRING)?
    ('language' languages+=LANGUAGE(',' languages+=LANGUAGE)*)
    ('email' email=STRING)
    ('weblink' weblink=STRING)
    ('courses' courses+=[ctCourse|FQN](',' courses+=[ctCourse|FQN])*)?
    (terms+=ctTerm)*
    (modules+=ctModule)*
  '}'
;

```

9.2.3 ctCourse

```

/**
 * Definition of a course
 */
ctCourse:
  (hide=HIDDEN)? 'Course' (nature=NATUREOFCOURSE) name=ID
    ((programtype='belongs' | programtype='in')
      rncMotherProgram=[ctProgram|FQN])? '{'
  //General Attributes
    ('name' label=STRING)
    ('reference' reference=REFERENCE)
    ('corecourse' corecourse=[ctCourse|FQN])?
  //TIMING
    ('academicyear' beginyear=INT '/' endyear=INT)
    ('term' rncTerm=[ctTerm|FQN])
    ('module' rncModule=[ctModule|FQN])?
    ('hoursPerWeek' hoursperweek=INT)
    ('totalHours' hoursetotal=INT)

  //COURSE INFORMATION
    ('description' description=STRING)
    ('credits' credits=INT)
    ('languages' languages+=LANGUAGE(',' languages+=LANGUAGE)*)?
    ('weblink' weblink=STRING)?

```

```

//TEACHING
('coursemoderator' coursemoderator=[ctInstructor|FQN])?

//People linked to the course
('groups' rnctGroup+=[ctGroup|FQN] (','rnctGroup+=[ctGroup|FQN])*)?
('boards' rnctBoard+=[ctBoard|FQN] (','rnctBoard+=[ctBoard|FQN])*)?
('promotions'
  rnctPromotion+=[ctPromotion|FQN] (','rnctPromotion+=[ctPromotion|FQN])*)?
('students'
  rnctStudent+=[ctStudent|FQN] (','rnctStudent+=[ctStudent|FQN])*)?

rnctCourseOrganisaton+=ctCourseOrganisaton*

//Period
rnctPeriod+=ctPeriod*
('tasks' (rnctTasks+=[ctTask|FQN]) (','rnctTasks+=[ctTask|FQN])*)?
('tests' (rnctTest+=[ctTest|FQN]) (','rnctTest+=[ctTest|FQN])*)?
//SWEBOK - CS2013 criteria
rnctFieldCoverage+=ctFieldCoverage*
}','
;

```

9.2.4 ctCourseOrganisation

```

/**
 * Class type for showing the course organisation and the hours given by each
 * teacher
 */
ctCourseOrganisaton:
  'organisation' name=ID 'typeof' coursetype=SUBCOURSETYPE ('called'
    other=STRING)? '{'
    ('instructor' rnctInstructor+=[ctInstructor|FQN] ':' 'hours'
      totalhours+=INT ',' 'weight' weight+=INT ',' ('language'
        language+=LANGUAGE))*
  '}',
;

```

9.2.5 ctPeriod

```

/**
 * Definiton of a period inside a semester
 */
ctPeriod:
  'Period' (('name=ID(',' level=INT)(',' number=INT)')) (('start'
    startDate=ctDate ('end' endDate=ctDate)?) ('from' startTime=ctTime ('to'
    endTime=ctTime)?)?)? '{'
    ('tasks' (rnctTasks+=[ctTask|FQN] (','rnctTasks+=[ctTask|FQN])*)?)?
    ('tests' rnctTest+=[ctTest|FQN] (','rnctTest+=[ctTest|FQN])*)?
    subPeriod+=ctPeriod*
  '}',
;

```

9.2.6 ctTask

```

//DEFINITION OF A TASK
//remove concerns instructor add attribute concerns
ctTask:
  'Task'(tasktype=TASKTYPE) name=ID ('contains'
    rnctSubTasks+=[ctTask|FQN](','rnctSubTasks+=[ctTask|FQN])*)? 'concerns'
    (concerned=TASKCONCERNED)? '{'
  //Attributes
    ('description' description=STRING)?
  //Relations - Actors
    ('instructors'
      rnctInstructors+=[ctInstructor|FQN](':instructorVolume+=INT)?(','rnctInstructors+=[ctIns
    ('students'
      rnctStudents+=[ctStudent|FQN](':studentVolume+=INT)?(','rnctStudents+=[ctStudent|FQN](':
    ('promotions'
      rnctPromotions+=[ctPromotion|FQN](':promotionVolume+=INT)?(','rnctPromotions+=[ctPromoti
    ('groups'
      rnctGroups+=[ctGroup|FQN](':groupVolume+=INT)?(','rnctGroups+=[ctGroup|FQN](':groupVolu
    ('boards'
      rnctBoards+=[ctBoard|FQN](':boardVolume+=INT)?(','rnctBoards+=[ctBoard|FQN](':boardVolu

  //SWEBOOK - CS2013 criteria /*REDESIGN*/
  //rnctFieldCoverage+=ctFieldCoverage*

  //Artefacts
    (rnctArtefacts+=ctArtefact)*
  '}'
;

```

9.2.7 ctArtefact

```

ctArtefact:'Artefact' name=ID '(' artefacttype=ARTEFACTTYPE','
  description=STRING)';

```

9.2.8 ctTest

```

//Definition of a Test
ctTest:
  'Test' name=ID 'weight' value=INT ('covers'
    rnctTask+=[ctTask|FQN](','rnctTask+=[ctTask|FQN])*)? '{'
    ('description' description=STRING)?
    ('rationale' rationale=STRING)?
    ('grading'categories'
      rnctCategory+=[ctCategory|FQN](','rnctCategory+=[ctCategory|FQN])*)?
    ('correctors'
      rnctInstructors+=[ctInstructor|FQN](':weight+=INT)?(','rnctInstructors+=[ctInstructor|FQ
    ('result' result=ctMeasuredVariable)?
    rnLowerTest+=ctTest*
  '}'
;

```

9.2.9 ctCategory

```
ctCategory:
  'GradingCategory' name=ID 'weight' weight=INT ('contains'
    'gradingCriteria'
    rnctCriteria+=[ctCriteria|FQN](','rnctCriteria+=[ctCriteria|FQN])*)?
  ('points' evaluation=ctMeasuredVariable)?
  ('description' description=STRING)?
;
```

9.2.10 ctCriteria

```
ctCriteria :
  'GradingCriteria' name=ID 'weight' weight=INT ('points'
    evaluation=ctMeasuredVariable )? 'description' description=STRING
;
```

9.2.11 ctMeasuredVariable

```
ctMeasuredVariable:
  result=(ctNumericalVariable | ctNumericalEnum | ctNominalOrdinalVariable |
    ctOrdinalVariable)
;
```

```
ctNumericalVariable:  '['lowerbound=INT','upperbound=INT']' 'with' 'step'
  step=INT;
ctNumericalEnum:      '{'numericalenum+=INT(',' numericalenum+=INT)*'}';
ctNominalOrdinalVariable:
  '{'nominalvariable+=STRING':'description+=STRING(','nominalvariable+=STRING':'description+=S
ctOrdinalVariable:
  '['(ordinalVariable+=STRING':'description+=STRING('-'ordinalVariable+=STRING':'description+=
```

9.2.12 ctGrades

```
//Grading scale
ctGrades :
  'Grades' name=ID '{'
    ('Course' rnctCourse+=[ctCourse|FQN] '{'
      ('Test' rnctTest+=[ctTest|FQN] '{'
        ('gradingCategory' rnctCategory+=[ctCategory|FQN] '{'
          ('grades' (rnctCatIns+=[ctInstructor|FQN] ('('weight+=INT')) ':'
            categoryGrade+=INT)((','rnctCatIns+=[ctInstructor|FQN] ('('weight+=INT'))')
            ':' categoryGrade+=INT)*)?
          ('gradingCriteria' rnctCriteria+=[ctCriteria|FQN] '{'
            ('grades' (rnctCritIns+=[ctInstructor|FQN] ('('weight+=INT'))')
              ':'
              criteriaGrade+=INT)((','rnctCritIns+=[ctInstructor|FQN] ('('weight+=INT'))')
              ':' criteriaGrade+=INT)*)?
            '})*)
          '})*)
        '})*)
      '})*)
    '})*)
```

```

    '}',
;

```

9.2.13 ctActor

```

//DEFINITION OF THE abstract class Actor and the subactors => Instructors,
    Groups and Students

```

```

ctActor :
    val=(ctInstructor | ctBoard | ctGroup | ctStudent | ctPromotion)
;

```

9.2.14 ctBoard

```

ctBoard:
    'Board' name=ID ('in'
        rnctCourses+=[ctCourse|FQN](','rnctCourse+=[ctCourse|FQN])*)? ':'
        ('instructors'
            rnctInstructor+=[ctInstructor|FQN](':totalHours+=INT'h')('moderator'))(','
            rnctInstructor+=[ctInstructor|FQN](':totalHours+=INT'h'))*)?
;

```

9.2.15 ctGroup

```

ctGroup:
    'Group' name=ID ('in'
        rnctCourses+=[ctCourse|FQN]?(','rnctCourse+=[ctCourse|FQN])*)? ':'
        ('instructors' rnctInstructor+=[ctInstructor|FQN](','
            rnctInstructor+=[ctInstructor|FQN])*)?
        ('students' rnctStudent+=[ctStudent|FQN]?
           (','rnctStudent+=[ctStudent|FQN])*)?
;

```

9.2.16 ctInstructor

```

ctInstructor:
    'Instructor' name=ID ('in'
        rnctCourses+=[ctCourse|FQN]?(','rnctCourse+=[ctCourse|FQN])*)? ':'
        ('name' firstname=STRING(','lastname=STRING)?)?
        ('email' email=STRING)?
        ('groups' rnctGroup+=[ctGroup|FQN]?(','rnctGroup+=[ctGroup|FQN])*)?
        ('board' board+=[ctBoard|FQN](','board+=[ctBoard|FQN])*)?
;

```

9.2.17 ctPromotion

```
ctPromotion:
  'Promotion' name=ID ('in'
    rnctCourses+=[ctCourse|FQN]?(','rnctCourse+=[ctCourse|FQN])*)? ':'
    ('groups' rnctGroup+=[ctGroup|FQN](','rnctGroup+=[ctGroup|FQN])*)?
;

```

9.2.18 ctStudent

```
ctStudent:
  'Student' name=ID ('in'
    rnctCourses+=[ctCourse|FQN]?(','rnctCourse+=[ctCourse|FQN])*)? ':'
    ('name' firstname=STRING(','lastname=STRING)?)?
    ('email' email=STRING)?
    ('groups' rnctGroup+=[ctGroup|FQN](','rnctGroup+=[ctGroup|FQN])*)?
    (rnctGrades+=ctGrades)?
;

```

9.2.19 ctFieldCoverage

```
/*STANDARD DEFINITION */
ctFieldCoverage :
  'FieldCoverage' name=ID '('focusLevel=INT'/'inputLevel=INT'/'
    outputLevel=INT')' '{'
    (rnctField+=[ctField|FQN] '('relativeWeight+=INT'/'bloomInLevel+=INT'/'bloomOutLevel+=INT')
    ('rnctField+=[ctField|FQN] '('relativeWeight+=INT'/'bloomInLevel+=INT'/'bloomOutLevel+=INT'
    ")")
;

```

9.2.20 ctStandard

```
ctStandard :
  'Standard' name=ID '{'
    (rnctFields+=ctField)*
  '}'
;

```

9.2.21 ctField

```
ctField:
  'Field' name=ID
    '('areaNumber=INT(','topicNumber=INT(','subtopicnumber=INT)?)?','description=STRING')'
;

```

9.2.22 Auxiliary class types

```
ctDate: date=DATE;
ctTime: time=TIME;

/* RELATION TYPES */
```

9.2.23 Auxiliary class types

```
/**
 * Class type Term for defining a quarter, semester etc
 */
ctTerm:
  'Term' name=ID 'referenced' reference=REFERENCE '{'
    (modules+=ctModule)*
  '}'
;

/**
 * Class type Module for defining a subdivision in groups of same course types
 * of a term (does not depend on a term)
 */
ctModule :
  'Module' name=ID 'referenced' reference=REFERENCE ('contains'
    courses+=[ctCourse|FQN] (','courses+=[ctCourse|FQN])*)?
;

```

9.2.24 Auxiliary data types

```
ctDate: date=DATE;
ctTime: time=TIME;

LANGUAGE: STRING ;
HIDDEN: "hide";
NATUREOFCOURSE: "required" | "elective";
TASKTYPE: "lecture" | "directed_work" | "written_exam" | "mid-term_exam" |
  "oral_exam" | "seminar_paper" | "project" | "presentation" | "exercises" |
  "other" | "no_assessment";
SUBCOURSETYPE: "lecture" | "practical" | "tutorial" | "other";
ARTEFACTTYPE: "output" | "input";
TASKCONCERNED : "instructor" | "student";
//Modifiable in config file
TIME: INT('.'INT);
DATE: INT('.'INT)('.'INT);
REFERENCE: INT('.'INT)*;
//WEEKDAY: "monday" | "tuesday" | "wednesday" | "thursday" | "friday" |
  "saturday" | "sunday";
FQN: ID ('.' ID)*;
```

9.3 Appendix B : TESMA - Generator

```

/*
 * generated by Xtext 2.9.2
 */
package lu.uni.lassy.messep.tesma.generator

import java.util.HashMap
import java.util.List
import lu.uni.lassy.messep.tesma.tesma.ctActor
import lu.uni.lassy.messep.tesma.tesma.ctCategory
import lu.uni.lassy.messep.tesma.tesma.ctCourse
import lu.uni.lassy.messep.tesma.tesma.ctCriteria
import lu.uni.lassy.messep.tesma.tesma.ctInstitution
import lu.uni.lassy.messep.tesma.tesma.ctProgram
import lu.uni.lassy.messep.tesma.tesma.ctStandard
import lu.uni.lassy.messep.tesma.tesma.ctTask
import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.xtext.generator.IFileSystemAccess2
import org.eclipse.xtext.generator.IGeneratorContext
import lu.uni.lassy.messep.tesma.handler.CsvHandler
import java.util.ArrayList
import java.util.Map

/**
 * Generates code from your model files on save.
 *
 * See
 * https://www.eclipse.org/Xtext/documentation/303\_runtime\_concepts.html#code-generation
 */
class TesmaGenerator implements ITesmaGenerator {

    var Iterable<ctInstitution> institutions
    var Iterable<ctProgram> programs
    var Iterable<ctCourse> courses
    var Iterable<ctStandard> standards
    var Iterable<ctTask> tasks
    var Iterable<ctActor> actors
    var Iterable<ctCategory> categories
    var Iterable<ctCriteria> criterias
    var ArrayList<String[]> isced
    var Map<String,String> iscedMap

    /**
     * Initializer function
     */
    def init(){
        iscedMap=new HashMap<String,String>()
        for(line:isced){
            var level=line.get(0)
            var category=line.get(1)
            var subcategory=line.get(2)

```

```

        if(category.equals("*")){
            category="0"
        }
        if(subcategory.equals("*")){
            subcategory="0"
        }
        iscedMap.put(level+category+subcategory.line.get(3))
    }
}

override doGenerate(List<Resource> input, IFileSystemAccess2 fsa){
    isced=CsvHandler.importData("data/isced.csv")
    if(iscid!=null){
        init()
    }

    //CALL OF ALL DATA NEEDED FOR CREATING THE DESCRIPTION
    institutions =
        input.map(r|r.allContents.toIterable.filter(typeof(ctInstitution))).flatten
    programs =
        input.map(r|r.allContents.toIterable.filter(typeof(ctProgram))).flatten
    courses =
        input.map(r|r.allContents.toIterable.filter(typeof(ctCourse))).flatten
    standards =
        input.map(r|r.allContents.toIterable.filter(typeof(ctStandard))).flatten
    tasks =
        input.map(r|r.allContents.toIterable.filter(typeof(ctTask))).flatten
    actors =
        input.map(r|r.allContents.toIterable.filter(typeof(ctActor))).flatten
    categories =
        input.map(r|r.allContents.toIterable.filter(typeof(ctCategory))).flatten
    criterias =
        input.map(r|r.allContents.toIterable.filter(typeof(ctCriteria))).flatten

    //All Institutions
    if (institutions!=null) generateInstitutions(fsa.institutions)

    //All Programs
    if (programs != null) generatePrograms(fsa.programs)

    //AllCourses
    if (courses!=null) generateCourses(fsa.courses)

    //AllStandards
    //if(standards!=null) generateStandards(fsa.standards)

    //Generating Final report
    fsa.generateFile(''TesmaReport/TesmaReport.tex'',generateReport)
}

//FILE GENERATION
def generateStandards(IFileSystemAccess2 fsa, Iterable<ctStandard>
standards) {
    for(s:standards){
        try{

```

```

        fsa.generateFile(''TesmaReport/Standard/s.name.tex''.(s as
            ctStandard).generateStandardContent)
    }
    catch(Exception e){
        println(''Error while generating Standard : s.name'')
    }
}
}
.
def generateCourses(IFileSystemAccess2 fsa. Iterable<ctCourse> courses) {
    for(c:courses)
        if(c.hide.nullOrEmpty){
            try{
                fsa.generateFile(''TesmaReport/Courses/c.name.tex''.(c as
                    ctCourse).generateCourseContent)
                //XLSGenerator.generateGradingTable(c as
                    ctCourse.'''src-gen/TesmaEvaluation/Courses/c.name/GlobalTable_c.name.xls''')
            }
            catch(Exception e) {
                println(''Error while generating Course : c.name'')
            }
        }
    }
}

def generatePrograms(IFileSystemAccess2 fsa. Iterable<ctProgram> programs){
    for(p:programs){
        if(p.hide.nullOrEmpty){
            try{
                fsa.generateFile(''TesmaReport/Programs/p.name.tex''.(p as
                    ctProgram).generateProgramContent)

            }
            catch(Exception e){
                println(''Error while generating Program : p.name'')
            }
        }
    }
}

def generateInstitutions(IFileSystemAccess2 fsa. Iterable<ctInstitution>
    institutions) {
    for(i:institutions){
        if(i.hide.nullOrEmpty){
            try{
                fsa.generateFile(''TesmaReport/Institutions/i.name.tex'''.i.generateInstitutionCo
            }
            catch(Exception e){
                println(''Error while generating Institution : i.name'')
            }
        }
    }
}

//REPORT CONTENT GENERATION
def generateStandardContent(ctStandard s)''''''
/**

```

```

* Generating the content of a course
*/
def generateCourseContent(ctCourse c)'''
  IF c.hide==null && c.corecourse==null
    \subsection{c.label - c.reference}
    \subsubsection{Core Course Information}
    \paragraph{Description}
    \par c.description
    \begin{center}
    \setlength\mylength{\dimexpr\textwidth-5\arrayrulewidth-8\tabcolsep}
    \begin{longtable}{|p{0.4\mylength}|p{0.3\mylength}|p{0.3\mylength}|}
    \caption{Course Information} \label{tab:long} \\

    \hline \multicolumn{3}{|p{1\mylength}|}{\textbf{c.label - c.reference}} \\
    \hline
    \endfirsthead

    \multicolumn{3}{p{1\mylength}}%
    {\bfseries \tablename\ \thetable{} -- continued from previous page} \\
    \hline \multicolumn{3}{|p{1\mylength}|}{\textbf{c.label}} \\
    \endhead

    \hline \multicolumn{3}{|p{1\mylength}|}{Continued on next page} \\
    \hline
    \endfoot

    \hline \multicolumn{3}{|p{1\mylength}|}{c.label} \\
    \endlastfoot

    \hline \multicolumn{2}{|p{0.5\mylength}|}{Academic Year} &
    c.beginyear/c.endyear \\
    \hline

    IF c.rnctMotherProgram!=null
    \multicolumn{3}{|p{1\mylength}|}{\textbf{\Large{Core Program}}} \\
    \hline
    \multicolumn{2}{|p{0.5\mylength}|}{Name} & c.rnctMotherProgram.label \\
    \hline
    \multicolumn{2}{|p{0.5\mylength}|}{Term} & c.rnctTerm.reference \\
    \hline
    IF c.rnctModule!=null
    \multicolumn{2}{|p{0.5\mylength}|}{Module} & c.rnctModule.reference \\
    \hline
    ENDIF
    ENDIF
    \multicolumn{2}{|p{0.5\mylength}|}{Credits} & c.credits \\
    \hline
    \multicolumn{2}{|p{0.5\mylength}|}{Hourse/Week} & c.hoursperweek \\
    \hline
    \multicolumn{2}{|p{0.5\mylength}|}{Total Hours} & c.hoursetotal \\
    \hline\hline
    \multicolumn{2}{|p{0.5\mylength}|}{Languages} & FOR l : c.languagesl
    ENDFOR \\
    \hline
    \multicolumn{2}{|p{0.5\mylength}|}{Weblink} & c.weblink \\
    \hline
    \multicolumn{2}{|p{0.5\mylength}|}{Course Moderator} &
    c.coursemoderator.firstname c.coursemoderator.lastname \\
    \hline

```

```

\hline
\end{longtable}
\end{center}

FOR course:courses
  IF c==course.corecourse
    \subsubsection{Related Course Information in other programs}
    \paragraph{Description}
    \par course.description
    \begin{center}
      \setlength\mylength{\dimexpr\textwidth-5\arrayrulewidth-8\tabcolsep}
      \begin{longtable}{|p{0.4\mylength}|p{0.3\mylength}|p{0.3\mylength}|}
        \caption{Course Information} \label{tab:long} \\

        \hline \multicolumn{3}{|p{1\mylength}|}{\textbf{course.label}} \\
          \hline
        \endfirsthead

        \multicolumn{3}{|p{1\mylength}|}%
        {\bfseries \tablename\ \thetable} -- continued from previous
          page} \\
        \hline \multicolumn{3}{|p{1\mylength}|}{\textbf{course.label}} \\
          \hline
        \endhead

        \hline \multicolumn{3}{|p{1\mylength}|}{Continued on next page}
          \\ \hline
        \endfoot

        \hline \multicolumn{3}{|p{1\mylength}|}{course.label} \\ \hline
        \endlastfoot

        \hline \multicolumn{2}{|p{0.5\mylength}|}{Academic Year} &
          course.beginyear/course.endyear \\ \hline
        \hline
        \multicolumn{2}{|p{0.5\mylength}|}{Name} &
          course.rnctMotherProgram.label \\ \hline
        \multicolumn{2}{|p{0.5\mylength}|}{Term} &
          course.rnctTerm.reference \\ \hline
        IF course.rnctModule!=null
          \multicolumn{2}{|p{0.5\mylength}|}{Module} &
            course.rnctModule.reference \\ \hline
        ENDIF
        \multicolumn{2}{|p{0.5\mylength}|}{Hourse/Week} &
          course.hoursperweek \\ \hline
        \multicolumn{2}{|p{0.5\mylength}|}{Total Hours} &
          course.hoursetotal \\ \hline\hline
        \multicolumn{2}{|p{0.5\mylength}|}{Languages} & FOR 1 :
          c.languagesl ENDFOR \\ \hline
        \multicolumn{2}{|p{0.5\mylength}|}{Weblink} & c.weblink \\ \hline
        \multicolumn{2}{|p{0.5\mylength}|}{Course Moderator} &
          c.coursemoderator.firstname c.coursemoderator.lastname \\ \hline

```

```

        \hline
        \end{longtable}
        \end{center}
    ENDIF
ENDFOR
\input{report-input/TesmaReport/Courses/c.name}
ENDIF
''',

/**
 * Generating Content of a program
 */
def generateProgramContent(ctProgram p)'''
    IF p.hide==null
        IF p.rnctMotherProgram==null
            \subsection{p.label}
        ELSE
            \subsection{p.rnctMotherProgram.label \ \ p.label}
        ENDIF
        \par \textbf{International Standard Classification of Education : }\\
        IF
            iscedMap.get(p.iscedLevel.toString+p.iscedCategory.toString+p.iscedSubcategory.toString)
            iscedMap.get(p.iscedLevel.toString+p.iscedCategory.toString+p.iscedSubcategory.toString)
        ELSE
            ISCED NOT CORRECTLY DEFINED
        ENDIF
        IF p.description!=null
            \subsubsection{Description}
            p.description
        ENDIF
        IF p.requisites!=null || p.prerequisites!=null || p.cost!=null
            \subsubsection{Inscription Details}
            IF p.prerequisites!=null
                \par p.prerequisites
            ENDIF
            IF p.requisites!=null
                \par p.requisites
            ENDIF
            IF p.cost!=null
                \par The semester fees are set to p.cost euros
            ENDIF
        ENDIF
        \subsubsection{Teaching Language}
        \par The program is taught mainly in FOR 1:p.languages1 ENDFOR.
        \subsubsection{Contacts}
        \begin{description}
        \item[Program Director] p.rnctprogramdirector.lastname
            p.rnctprogramdirector.firstname - p.rnctprogramdirector.email
        \item[Secretary] p.email
        \item[Weblink] p.weblink.
        \end{description}
        IF !p.modules.nullOrEmpty
            \subsubsection{All courses}
            \begin{center}
            \setlength\mylength{\dimexpr\textwidth-5\arrayrulewidth-8\tabcolsep}
            \begin{longtable}{|p{.5\mylength}|p{.05\mylength}|p{.45\mylength}|}

```

```

\caption{All Courses per Module} \label{tab:long} \\\
\hline \multicolumn{3}{|p{1\mylength}|}{\textbf{Module
  Organisation}} \\\ \hline
\endfirsthead

\multicolumn{3}{p{1\mylength}}%
{\bfseries \tablename\ \thetable{} -- continued from previous
  page}} \\\
\hline \multicolumn{3}{|p{1\mylength}|}{\textbf{Module
  Organisation}} \\\ \hline
\endhead

\hline \multicolumn{3}{|p{1\mylength}|}{Continued on next page}}
  \\\ \hline
\endfoot

\hline \multicolumn{3}{|p{1\mylength}|}{Module Organisation}} \\\
  \hline
\endlastfoot

FOR m:p.modules
  \hline \multicolumn{3}{|p{1\mylength}|}{\textit{Module
    m.reference}} \\\ \hline
  FOR c:m.courses
    c.label & c.credits & c.weblink \\\ \hline
  ENDFOR
ENDFOR
\hline
\end{longtable}
\end{center}
ENDIF
IF !p.courses.nullOrEmpty
  \subsubsection{All courses}
  \begin{center}
  \setlength\mylength{\dimexpr\textwidth-5\arrayrulewidth-8\tabcolsep}
  \begin{longtable}{|p{.5\mylength}|p{.05\mylength}|p{.45\mylength}|}
  \caption{All Courses} \label{tab:long} \\\
  \hline \multicolumn{3}{|p{1\mylength}|}{\textbf{Course List}} \\\
    \hline
  \endfirsthead

  \multicolumn{3}{p{1\mylength}}%
  {\bfseries \tablename\ \thetable{} -- continued from previous
    page}} \\\
  \hline \multicolumn{3}{|p{1\mylength}|}{\textbf{Course List}} \\\
    \hline
  \endhead

  \hline \multicolumn{3}{|p{1\mylength}|}{Continued on next page}}
    \\\ \hline
  \endfoot

  \hline \multicolumn{3}{|p{1\mylength}|}{Course List}} \\\ \hline
  \endlastfoot
  FOR c:p.courses
    c.label & c.credits & c.weblink \\\ \hline

```



```

        ENDFOR
        \hline
        \end{longtable}
        \end{center}
    ENDIF
    IF !p.terms.nullOrEmpty
        \subsubsection{All courses}
        FOR t:p.terms
            \begin{center}
                \setlength\mylength{\dimexpr\textwidth-5\arrayrulewidth-8\tabcolsep}
                \begin{longtable}{|p{.5\mylength}|p{.05\mylength}|p{.45\mylength}|}
                \caption{All Courses for term t.reference} \label{tab:long} \\\
                \hline \multicolumn{3}{|p{1\mylength}|}{\textbf{Term t.reference
                    Organisation}} \\\ \hline
                \endfirsthead

                \multicolumn{3}{p{1\mylength}}%
                {\bfseries \tablename\ \thetable{} -- continued from previous
                    page}} \\\
                \hline \multicolumn{3}{|p{1\mylength}|}{\textbf{Term t.reference
                    Organisation}} \\\ \hline
                \endhead

                \hline \multicolumn{3}{|p{1\mylength}|}{\textbf{Continued on next page}}
                    \\\ \hline
                \endfoot

                \hline \multicolumn{3}{|p{1\mylength}|}{\textbf{Term t.reference
                    Organisation}} \\\ \hline
                \endlastfoot

                \hline \multicolumn{3}{|c|}{\textbf{Term t.reference}} \\\ \hline
                FOR m:t.modules
                    \hline \multicolumn{3}{|p{1\mylength}|}{\textit{Module
                        m.reference}} \\\ \hline
                    FOR c:m.courses
                        c.label & c.credits & c.weblink \\\ \hline
                    ENDFOR
                ENDFOR
                \hline
                \end{longtable}
                \end{center}
            ENDFOR
        ENDIF
        \input{report-input/TesmaReport/Programs/p.name}
    ENDIF
''',

/**
 * Generating Content of a Institution
 */
def generateInstitutionContent(ctInstitution i)'''
    IF i.hide==null
        IF i.rnctMotherInstitution==null
            \section{i.label}
        ELSE

```

```

        \section{i.rnctMotherInstitution.label \\ i.label}
    ENDF
    IF i.description!=null
        \subsection{Campus Description}
        \par{i.description}
    ENDF
    IF i.address!=null
        \subsection{Address}
        \par The campus i.sepRegion is situated at the following address :
            \\ i.address
    ENDF
    IF i.currentSituation!=null
        \subsection{Current Situation}
        \par The campus i.sepRegion is situated at the following address :
            \\ i.address
    ENDF
    IF i.gastronomie!=null
        \subsection{Gastronomie}
        \par The campus i.sepRegion is situated at the following address :
            \\ i.address
    ENDF
    IF i.contact!=null
        \subsection{Contacts}
        \par i.contact
    ENDF
    \input{report-input/TesmaReport/Institutions/i.name}
ENDIF
'''

```

//REPORT GENERATION

```

def generateReport()'''
    generateReportIntroduction()
    generateInstitutionSection(institutions)
    generateProgramSection(programs)
    generateCourseSection(courses)
    generateReportConclusion()
'''

```

//SECTION GENERATION

```

def generateCourseSection(Iterable<ctCourse> courses)'''
    IF !courses.nullOrEmpty
        \chapter{\textbf{Courses}}
        FOR c:courses
            IF c.hide==null && c.corecourse==null
                \section{c.rnctMotherProgram.name - c.label}
                \input{Courses/c.name}
            ENDF
        ENDFOR
    ENDF
'''
/**
 * Generating Program Section

```

```

*/
def generateProgramSection(Iterable<ctProgram> programs)'''
  IF !programs.nullOrEmpty
    \chapter{\textbf{Programs}}
    FOR i:institutions
      IF !i.programs.nullOrEmpty
        \section{i.label}
        FOR p:programs
          IF p.institution==i
            \input{Programs/p.name}
          ENDIF
        ENDFOR
      ENDIF
    ENDFOR
    FOR p:programs
      IF p.institution==null
        \section{p.label}
        \input{Programs/p.name}
      ENDIF
    ENDFOR
  ENDIF
'''

/**
 * Generating Institution section
 * @param Iterable<ctInstitution> institution
 */
def generateInstitutionSection(Iterable<ctInstitution> institution)'''
  IF !institution.nullOrEmpty
    \chapter{\textbf{The University}}
    FOR i:institutions
      \input{Institutions/i.name}
    ENDFOR
  ENDIF
'''

/**
 * Generating Report Conclusion
 */
def generateReportConclusion()'''
  \end{document}
'''

/**
 * Generating Report Introduction
 */
def generateReportIntroduction() '''
  \documentclass[11pt]{report}
  \usepackage[a4paper. top=3cm. bottom=3cm]{geometry}
  \usepackage[latin1]{inputenc}
  \usepackage{setspace}
  \usepackage{fancyhdr}
  \usepackage{sectsty}
  \usepackage{tocloft}
  \usepackage{amsmath.amssymb}
  \usepackage{hyperref}
  \usepackage{graphicx}

```

```

\usepackage{array}
\usepackage{tikz}
\usepackage{longtable}
\usepackage{booktabs}
\allsectionsfont{\centering}
\title{\textbf{University of Luxembourg} \ \ Description. Programs and
      Course Syllabus}
\begin{document}
\maketitle
\newpage
\newpage
% Define Page style for all chapters
\pagestyle{fancy}
% Delete the current section for header and footer
\fancyhf{}
% Set custom header
\lhead[]{\thepage}
\rhead[\thepage]{}
% Set arabic (1.2.3...) page numbering
\pagenumbering{arabic}
\newlength\mylength
\textwidth = 400pt
'''

override afterGenerate(Resource input. IFileSystemAccess2 fsa.
    IGeneratorContext context) {
    throw new UnsupportedOperationException("TODO: auto-generated method
        stub")
}

override beforeGenerate(Resource input. IFileSystemAccess2 fsa.
    IGeneratorContext context) {
    throw new UnsupportedOperationException("TODO: auto-generated method
        stub")
}

override doGenerate(Resource input. IFileSystemAccess2 fsa.
    IGeneratorContext context) {
    throw new UnsupportedOperationException("TODO: auto-generated method
        stub")
}
}

```

9.4 Appendix C : TESMA - Validator

9.4.1 Validator

```

/*
 * generated by Xtext 2.9.2
 */
package lu.uni.lassy.messep.tesma.validation

```

```

import lu.uni.lassy.messep.tesma.tesma.ctStudent
import lu.uni.lassy.messep.tesma.tesma.ctBoard
import lu.uni.lassy.messep.tesma.tesma.ctGroup
import lu.uni.lassy.messep.tesma.tesma.ctPromotion
import lu.uni.lassy.messep.tesma.tesma.ctInstructor

import lu.uni.lassy.messep.tesma.tesma.ctCourse
import lu.uni.lassy.messep.tesma.tesma.ctInstitution
import lu.uni.lassy.messep.tesma.tesma.ctProgram

import lu.uni.lassy.messep.tesma.tesma.TesmaPackage

import java.util.Collections
import java.util.ArrayList
import java.util.Collection
import java.util.List

import org.eclipse.xtext.validation.Check

import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.core.runtime.Path
import org.eclipse.core.resources.ResourcesPlugin

/**
 * This class contains custom validation rules.
 *
 * See
 * https://www.eclipse.org/Xtext/documentation/303\_runtime\_concepts.html#validation
 */
class TesmaValidator extends AbstractTesmaValidator {

    public static val INVALID_NAME = 'invalidName'
    public static val NOT_DEFINED = 'notDefined'
    public static val DUPLICATES_FOUND = 'duplicatedFound'

    //CHECKS ON PROGRAMS
    /**
     * Uniqueness of Programs
     */
    @Check
    def checkIfProgramsAreUnique(ctProgram p){
        val platformString = p.eResource.URI.toPlatformString(true)
        val myFile = ResourcesPlugin.getWorkspace().getRoot().getFile(new
            Path(platformString))
        val project = myFile.getProject()
        val List<Resource> resources= FileHandler.getProjectResources(project)
        var Iterable<ctProgram> programs =
            resources.map(r|r.allContents.toIterable.filter(typeof(ctProgram))).flatten

        var List<ctProgram> duplicates = new ArrayList<ctProgram>()
        for(prog:programs){
            if(prog.name.equals(p.name)){

```

```

        duplicates.add(prog)
    }
}
if(duplicates.size>1){
    error('Duplicated programs
        found', TesmaPackage.Literals.CT_PROGRAM__NAME, DUPLICATES_FOUND)
}
}

/**
 * Checks if all the data in the programs are complete
 */
@Check
def checkProgramsCompleteness(ctProgram p){

    if(p.label==null){
        error('Attribute label not
            defined', TesmaPackage.Literals.CT_PROGRAM__LABEL, NOT_DEFINED)
    }
    if(p.description==null){
        error('Attribute description not
            defined', TesmaPackage.Literals.CT_PROGRAM__DESCRIPTION, NOT_DEFINED)
    }
    if(p.email==null){
        error('Attribute email not
            defined', TesmaPackage.Literals.CT_PROGRAM__EMAIL, NOT_DEFINED)
    }
    if(p.weblink==null){
        error('Attribute weblink not
            defined', TesmaPackage.Literals.CT_PROGRAM__WEBLINK, NOT_DEFINED)
    }
    if(p.rnctprogramdirector==null){
        error('Attribute programdirector not
            defined', TesmaPackage.Literals.CT_PROGRAM__RNCTPROGRAMDIRECTOR, NOT_DEFINED)
    }
    if(p.languages.nullOrEmpty){
        error('Attribute languages not
            defined', TesmaPackage.Literals.CT_PROGRAM__LANGUAGES, NOT_DEFINED)
    }
}

/**
 * Checks the relations are correct inside the programs
 */
@Check
def checkProgramsCorrectness(ctProgram p){
    var count=0;
    if(!p.courses.nullOrEmpty){
        count++
    }

    if(!p.terms.nullOrEmpty){
        count++
    }
    if(!p.modules.nullOrEmpty){
        count++
    }
}

```

```

    }

    if(count>1){
        error('Duplicated information user either term, module or
              courses',TesmaPackage.Literals.CT_PROGRAM__TERMS,NOT_DEFINED)
        error('Duplicated information user either term, module or
              courses',TesmaPackage.Literals.CT_PROGRAM__MODULES,NOT_DEFINED)
        error('Duplicated information user either term, module or
              courses',TesmaPackage.Literals.CT_PROGRAM__COURSES,NOT_DEFINED)
    }

}

//CHECKS FOR CLASSTYPE : institution
/**
 * check if the institutions definition is complete
 */
@Check
def checkInstitutionsCompleteness(ctInstitution i) {
    if(i.label==null){
        error('Attribute label not
              defined',TesmaPackage.Literals.CT_INSTITUTION__LABEL,NOT_DEFINED)
    }
    if(i.sepRegion==0){
        error('Attribute sepRegion not defined correctly
              (>0)',TesmaPackage.Literals.CT_INSTITUTION__SEP_REGION,NOT_DEFINED)
    }
    if(i.address==null){
        error('Attribute address not
              defined',TesmaPackage.Literals.CT_INSTITUTION__ADDRESS,NOT_DEFINED)
    }
}

/**
 * check if the reference to a program is correct
 */
@Check
def checkInstitutionsReferencedToPrograms(ctInstitution i) {
    if (i.programs.nullOrEmpty && i.rnctMotherInstitution!=null) {
        warning('No link between institution and programs',
                TesmaPackage.Literals.CT_INSTITUTION__PROGRAMS,
                NOT_DEFINED)
    }
    var index=0
    while(index<i.programs.length){
        var Collection<ctProgram> progs =i.programs
        if(Collections.frequency(progs,i.programs.get(index))>1){
            error('Duplicated elements in programs
                  attribute',TesmaPackage.Literals.CT_INSTITUTION__PROGRAMS,DUPLICATES_FOUND)
        }
        index++
    }
}

/**

```

```

    * Uniqueness of Institutions
    */
    @Check
    def checkIfInstitutionsAreUnique(ctInstitution i){
        val platformString = i.eResource.URI.toPlatformString(true)
        val myFile = ResourcesPlugin.getWorkspace().getRoot().getFile(new
            Path(platformString))
        val project = myFile.getProject()
        val List<Resource> resources= FileHandler.getProjectResources(project)
        var Iterable<ctInstitution> institutions =
            resources.map(r|r.allContents.toIterable.filter(typeof(ctInstitution))).flatten

        var List<ctInstitution> duplicates = new ArrayList<ctInstitution>()
        for(inst:institutions){
            if(inst.name.equals(i.name)){
                duplicates.add(inst)
            }
        }
        if(duplicates.size>1){
            error('Duplicated programs
                found', TesmaPackage.Literals.CT_INSTITUTION__NAME, DUPLICATES_FOUND)
        }
    }
}

//CHECKS FOR COURSES
/**
 * Uniqueness of Courses
 */
@Check
def checkIfCoursesAreUnique(ctCourse c){
    val platformString = c.eResource.URI.toPlatformString(true)
    val myFile = ResourcesPlugin.getWorkspace().getRoot().getFile(new
        Path(platformString))
    val project = myFile.getProject()
    val List<Resource> resources= FileHandler.getProjectResources(project)
    var Iterable<ctCourse> courses =
        resources.map(r|r.allContents.toIterable.filter(typeof(ctCourse))).flatten

    var List<ctCourse> duplicates = new ArrayList<ctCourse>()
    for(course:courses){
        if(course.name.equals(c.name)){
            duplicates.add(course)
        }
    }
    if(duplicates.size>1){
        error('Duplicated programs
            found', TesmaPackage.Literals.CT_COURSE__NAME, DUPLICATES_FOUND)
    }
}

//CHECKS FOR ACTORS
/**
 * Check if students are Unique
 */

```



```

@Check
def checkIfStudentsAreUnique(ctStudent student){
    val platformString = student.eResource.URI.toPlatformString(true)
    val myFile = ResourcesPlugin.getWorkspace().getRoot().getFile(new
        Path(platformString))
    val project = myFile.getProject()
    val List<Resource> resources= FileHandler.getProjectResources(project)

    var Iterable<ctStudent> students =
        resources.map(r|r.allContents.toIterable.filter(typeof(ctStudent))).flatten
    var List<ctStudent> duplicates = new ArrayList<ctStudent>()

    for(s:students){
        if(s.name.equals(student.name)){
            duplicates.add(s)
        }
    }
    if(duplicates.size>1){
        error('Duplicated students found, change the
            name',TesdaPackage.Literals.CT_STUDENT__NAME,DUPLICATES_FOUND)
    }
}
/**
 * Check if promotions are Unique
 */
@Check
def checkIfPromotionsAreUnique(ctPromotion p){
    val platformString = p.eResource.URI.toPlatformString(true)
    val myFile = ResourcesPlugin.getWorkspace().getRoot().getFile(new
        Path(platformString))
    val project = myFile.getProject()
    val List<Resource> resources= FileHandler.getProjectResources(project)

    var Iterable<ctPromotion> promotions =
        resources.map(r|r.allContents.toIterable.filter(typeof(ctPromotion))).flatten
    var List<ctPromotion> duplicates = new ArrayList<ctPromotion>()

    for(promo:promotions){
        if(promo.name.equals(p.name)){
            duplicates.add(promo)
        }
    }
    if(duplicates.size>1){
        error('Duplicated students found, change the
            name',TesdaPackage.Literals.CT_PROMOTION__NAME,DUPLICATES_FOUND)
    }
}
/**
 * Check if groups are Unique
 */
@Check
def checkIfGroupsAreUnique(ctGroup g){
    val platformString = g.eResource.URI.toPlatformString(true)
    val myFile = ResourcesPlugin.getWorkspace().getRoot().getFile(new
        Path(platformString))
    val project = myFile.getProject()

```

```

val List<Resource> resources= FileHandler.getProjectResources(project)

var Iterable<ctGroup> groups =
    resources.map(r|r.allContents.toIterable.filter(typeof(ctGroup))).flatten
var List<ctGroup> duplicates = new ArrayList<ctGroup>()

for(group:groups){
    if(group.name.equals(g.name)){
        duplicates.add(group)
    }
}
if(duplicates.size>1){
    error('Duplicated students found, change the
        name',TesdaPackage.Literals.CT_GROUP__NAME,DUPLICATES_FOUND)
}
}
/**
 * Check if instructors are Unique
 */
@Check
def checkIfInstructorsAreUnique(ctInstructor i){
    val platformString = i.eResource.URI.toPlatformString(true)
    val myFile = ResourcesPlugin.getWorkspace().getRoot().getFile(new
        Path(platformString))
    val project = myFile.getProject()
    val List<Resource> resources= FileHandler.getProjectResources(project)

    var Iterable<ctInstructor> instructors =
        resources.map(r|r.allContents.toIterable.filter(typeof(ctInstructor))).flatten
    var List<ctInstructor> duplicates = new ArrayList<ctInstructor>()

    for(ins:instructors){
        if(ins.name.equals(i.name)){
            duplicates.add(ins)
        }
    }
    if(duplicates.size>1){
        error('Duplicated instructors found, change the
            name',TesdaPackage.Literals.CT_INSTRUCTOR__NAME,DUPLICATES_FOUND)
    }
}
/**
 * Check if boards are Unique
 */
@Check
def checkIfBoardsAreUnique(ctBoard b){
    val platformString = b.eResource.URI.toPlatformString(true)
    val myFile = ResourcesPlugin.getWorkspace().getRoot().getFile(new
        Path(platformString))
    val project = myFile.getProject()
    val List<Resource> resources= FileHandler.getProjectResources(project)

    var Iterable<ctBoard> boards =
        resources.map(r|r.allContents.toIterable.filter(typeof(ctBoard))).flatten
    var List<ctBoard> duplicates = new ArrayList<ctBoard>()

```

```
for(board:boards){
    if(board.name.equals(b.name)){
        duplicates.add(board)
    }
}
if(duplicates.size>1){
    error('Duplicated students found, change the
        name',TesmaPackage.Literals.CT_BOARD__NAME,DUPLICATES_FOUND)
}
}
}
```

9.5 Appendix D : TESMA - ICAIT 2016 Paper submission

TESMA: Towards the Development of a Tool for Specification, Management and Assessment of Teaching Programs

Nicolas Guelfi

University of Luxembourg
6 rue Coudenhove Kalergi
Luxembourg
+352.46 66 44 5251
nicolas.guelfi@uni.lu

Benjamin Jahić

University of Luxembourg
6 rue Coudenhove Kalergi
Luxembourg
+352. 691 30 74 38
benjamin.jahic.001@student.uni.lu

Benoît Ries

University of Luxembourg
6 rue Coudenhove Kalergi
Luxembourg
+352.46 66 44 5267
benoit.ries@uni.lu

ABSTRACT

Defining and managing teaching programs at university or other institutions is a complex task for which there is not much support in terms of methods and tools. This task becomes even more critical when comes the time to obtain certifications w.r.t. official standards. In this paper, we present an on-going project called TESMA whose objective is to provide an open-source tool dedicated to the specification and management (including certification) of teaching programs. This tool has been engineered using a development method called Messir for its requirements elicitation and introduces a domain-specific language dedicated to the teaching domain. This paper presents the current status of this project and the future activities planned.

Categories and Subject Descriptors

K.3.2 [Computers and Education]: Computer and Information Science Education – *accreditation, curriculum, self-assessment*.

General Terms

Documentation, Design, Languages.

Keywords

Teaching Programs Development, Tool-support, Software Engineering, Domain-Specific Languages, Automatic Generation.

INTRODUCTION

The University of Luxembourg is a young university (created in 2003). In this “start-up” context, we have been setting up new programs at bachelor, master and doctorate levels providing different education certificates. All those programs are offered to our students by three faculties. The need for a tool to support the task to define and manage (including certification) the education program came rapidly. The market analysis for this category of tools showed that no tool was available. A project has been started to engineer a method and a tool to support those needs. This project has been conducted following a software engineering process that had the following main steps:

- Requirements analysis: to provide the initial requirements for the TESMA tool, a requirement

specification document has been produced using the Messir method [1].

- Design: to state the main choices concerning the TESMA architecture and interfaces.
- Implementation: to reach an operational system usable for validation w.r.t. the requirements.
- Those steps have been performed iteratively to produce the TESMA tool in an incremental way.

The content of this paper provides details on the requirements analysis, design and implementation of the TESMA tools. It also ends with a short related work section which summarizes what we found when we did our market analysis.

1. TESMA REQUIREMENTS

Following part of the Messir method [1], we have elicited the actors that are concerned by teaching programs. They are:

- *The institution director* who represents the institution and validates new programs, courses and course modifications (e.g. the dean of a faculty, the head of a teaching unit, ...).
- *The program director* who specifies his programs and validates course modifications made by instructors.
- *The instructor*, who specifies, manages and maintains the courses he gives.
- *The student*, who tunes his curriculum (elective courses, ...) and receives information about his curriculum.
- *The secretary*, who is a delegate of any of the institutional actors (*institution director, program director or instructor*) and also ensures interoperability with other institution information systems.
- *The quality officer* who evaluates and validates the programs with respect to the universities internal laws. He's also responsible of program certification processes.

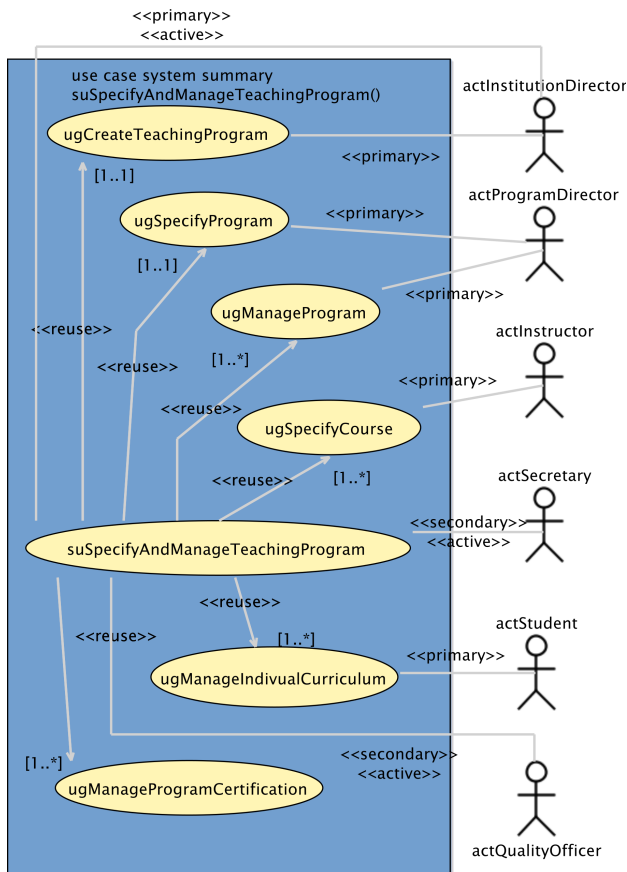


Figure 1. TESMA summary use-case.

You can find above a use-case model made in the context of the Messir method that displays the actors contributing to the high-level summary use-case dedicated to managing a teaching program.

The concepts managed by the TESMA actors are analysed and specified in the Messir concept model which is a UML class diagram. Among all the concepts that are necessary to specify the operations executed by the actors we have:

- concepts related to the actors and for which TESMA has to handle an internal representation: students, instructors, ...
- concepts related to the programs: course details, teaching periods, course evaluation, ...
- concepts related to program certification: standard description, standard coverage by an existing program, ...

In Messir those concepts are specified using an UML class diagram. Figure 2 provides a partial diagram view that models the concepts related to a program (institution, program, courses).

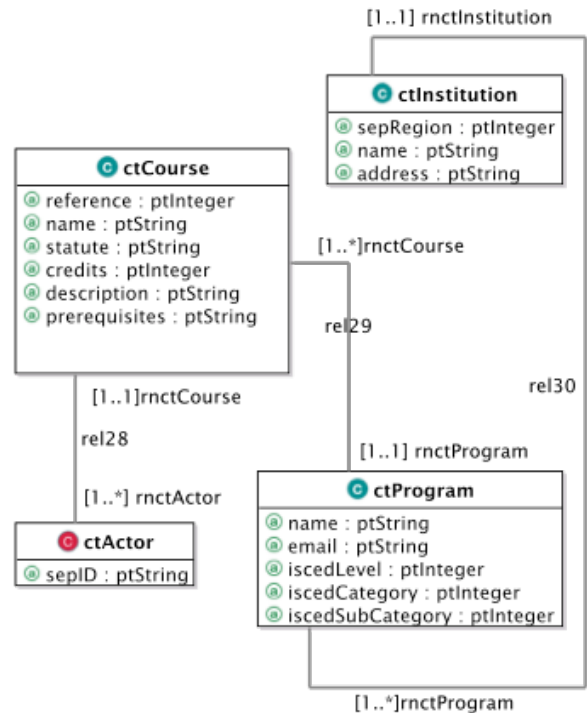


Figure 2. TESMA concepts.

The requirements analysis phase has allowed to determine a first version of the functionalities and data that should be handled in the first increment. The next section presents the design and implementation of this first version.

2. TESMA DESIGN AND IMPLEMENTATION

After having analysed the TESMA actors, concepts and functionalities. We have started to design a first version of the tool. A major design choice made is to allow the specification of programs using a domain-specific language (DSL) defined using the Xtext [2] framework. Thus, we designed TESMA as a plugin to the Eclipse workbench [3]. It is composed of three main architectural components illustrated in Figure 3 at the top of the diagram. Our components are based on stable Eclipse plugins themselves based on the Eclipse Modelling Framework (EMF) [4].

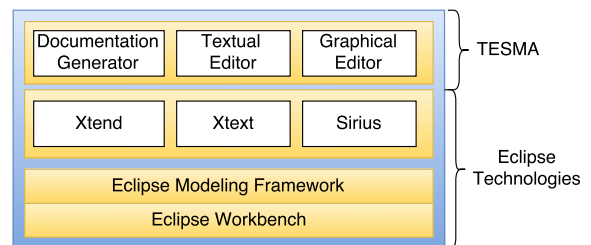


Figure 3. Architectural components overview.

2.1 Textual Editor

The main feature of the *Textual Editor* is to allow the specification of the teaching programs (this specification is called TESMA model in the remaining part of this paper) with the TESMA DSL. It also offers other supporting features, as for instance: syntactical validation rules, syntax highlighting, templates proposal, etc.

Xtext is an open-source framework that eases the development of domain-specific languages and offers features to provide a textual editor to the TESMA DSL. Xtext is based on EMF, which is the underlying-core library handling the TESMA model.

The TESMA DSL is designed to be intuitive, customizable and loosely coupled. In order to have an intuitive DSL, we have chosen to design its grammar using mainly keywords in natural language. Institutions may use different terminologies for the concepts used in our approach, this is why we designed the grammar of our DSL to be customizable. The institutions have the possibility to choose their own naming conventions. Lastly, the rules of the grammar are loosely coupled, i.e. optional cross-references are mostly used instead of containment relations.

2.2 Graphical Editor

The Graphical Editor provides a representation of the TESMA Model in a tabular view and offers the possibility to modify the TESMA model. The graphical editor provides typical table handling features like data sort, import/export from/to Excel sheets, hide/show columns, multiple rows selections.

The technology used to develop our graphical editor is Sirius [5], an open-source software Eclipse project that eases the creation of custom graphical modelling workbenches. Both Xtext and Sirius are based on EMF, which allows the TESMA tool-support to interact between Xtext and Sirius using EMF as underlying-core library for the TESMA model as represented in Figure 4.

Thanks to our tabular format, the graphical editor is intuitive and usable by non-computer experts. All the program's attributes are easy to access and modify. The modifications can be performed directly inside the graphical editor view.

2.3 Documentation Generator

The main feature of the *Documentation Generator* is to generate documents of different types, like Excel sheets, CSV files and PDF files. The *Documentation Generator* may be configured to produce a customized PDF file, e.g. by not generating some of the sections inside the pdf files.

The technologies used to develop the documentation generator are Xtend [2], Latex and the apache.poi library, for handling Excel sheets. Xtend is a programming language based on Java. It provides a compact syntax and eases the generation of natural language text. Latex is a document preparation system, which uses libraries, keywords and plaintext for writing scientific documents in pdf format. Finally, the apache.poi library provides the necessary tools for generating Excel sheets, which are used as teaching material.

The Documentation Generator has been designed to ease information retrieval in the generated Latex files. Additionally, it is designed to automatically update the final report, when the

user manually adds data into the reserved appropriate folders. Finally, the different Latex files are imported inside one Latex file, which is compiled into a pdf file containing the program description.

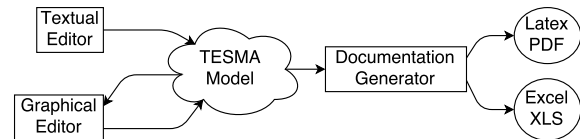


Figure 4. TESMA process overview.

3. ILLUSTRATION

We illustrate the TESMA approach with a course of a Master program named “Software Engineering Environment” (SEE) at the University of Luxembourg. Figure 5 is a screenshot of the TESMA tool-support in the Eclipse environment.

The TESMA model describing the SEE course have been specified using the approach described in this paper. The teaching program description of this example includes a number of textual files using the TESMA DSL syntax. The course run information is illustrated in Figure 5 by specifying the course teaching team organisation and dividing the teaching term into small periods and defining the tasks, tests for each period. The tasks and tests are defined in separate folders and referenced to the teaching period. At that point, TESMA is able to generate *a part of the Teaching Material*, like evaluation criteria, task lists and course information.

```
1 Course required MICS2_33 belongs to MICS {
2   name "Software Engineering Environments"
3   reference 2.33
4   academyyear 2016/2017
5   term MICS.Semester3
6   module MICS.Semester3.module43
7   hoursPerWeek 2
8   totalHours 120
9   description "Software engineers need means for quality engineering..."
10  credits 5
11  languages "french","english"
12  weblink "https://www.uni.lu/MICS/SEE"
13  coursemoderator guni
14  //Teaching and Students
15  boards SEBoard
16  promotions MICS_SEE_CLASS
17  organisation org1 typeof lecture {
18    instructor guni : hours 30,weight 1,language "english, french"
19    instructor beri : hours 30,weight 1,language "english, french"
20    instructor alca : hours 30,weight 1,language "english, french"
21  }
22  Period (Semester,1,0) start 17.09.2015 end 17.12.2015 {
23    Period (Lecture2,2,1) start 17.09.2015 end 08.10.2015 {
24      Period (Lecture1,3,1) start 17.09.2015 {tasks t1a}
25      Period (Lecture2,3,2) start 24.09.2015 {tasks t2a,t2b}
26    }
27    Period (Checkpoint3,2,2) start 17.12.2014 {tests oralCheckpoint3}
28    Period (FinalExam,3,1) start 15.01.2015 {tests finalExam}
29  }
```

Figure 5. SEE specification in TESMA tool.

All other concepts can be specified using the TESMA DSL including the coverage of an education standard by an education program.

In this case study for the SEE course, we created for each category of TESMA model element a file containing all information related. In this case, one institution, one program and one course have specified, which represent about 10 textual files. We defined 10 instructors and 7 students for this example case, which are grouped in a single file. The total description in our case needs about 500 lines of specification text (>1000 in

case of certification). The specification text size mostly depends on the preciseness of the specifier. If the specification is done in details, the number of lines increases quickly. In general, it could vary from 100 to 1500 lines.

4. RELATED WORK

A number of related works have been performed in the past in the field of education programs and course specifications, especially for K-12 classes in high schools. However, we could not find methods, which are supported by tools, which help to design detailed teaching programs. On the one hand, some university guidelines are available online, e.g. [6,7], without offering supporting software applications. On the other hand, a few software applications are available, that do not offer comprehensive and customizable guidelines. We present in the following, three of these tools.

PDF Syllabus builder [8] is an open-source tool, which only offers a template PDF form for writing course syllabi. TESMA covers many additional features, for instance it generates automatically reports in PDF format. The design of the reports is standardized and generated by the TESMA tool.

Jump-Rope [9] is a proprietary tool, based on a web application. It supports a curriculum design tool, standards based gradebook, accurate attendance, administrator tools. TESMA has similar features and is more flexible in terms of customization of its textual language. Moreover, the use of MDE techniques allows the automatic reconfiguration of the user interface and the generated documents.

Build-Your-Own-Curriculum [10] is a proprietary tool, based on web-application developed for K-12 classes in the United States. It supports the feature of defining standardized courses, classroom managements, evaluations and assignments. TESMA is similar to this kind of tools, but has two major advantages: firstly, it is based on MDE technique, which allows customizing the documentation generation process in a flexible way. Secondly TESMA is an open-source project, which allows adapting freely its code to the institution's taste.

5. CONCLUSION AND FUTURE WORK

In this paper, we presented how we developed our project of engineering a tool for educational program and course specification based on a textual domain-specific language with a graphical editor. It generates documents out of the specification, which can be used by the institution's staff. Our tool and method has been successfully used on a small, yet real, example. As a future work we plan to iterate the process to stabilize the requirements and the tool design and implementation. We also plan to study the automated generation of a web application from the language grammar in

order to provide a user-friendly front-end that is mapped to our textual language grammar.

6. REFERENCES

- [1] Guelfi, N. 2016. *The Messir Scientific Approach to Requirements Engineering*. Laboratory for Advanced Software Systems Technical Report, TR-LASSY-16-01. University of Luxembourg.
- [2] Bettini, L. 2013. *Implementing Domain-Specific Languages with Xtext and Xtend*. Packt Publishing Ltd., Birmingham, UK.
- [3] Beck, K., Gamma, E. 2003. *Contributing to eclipse – Principles, Patterns, and Plugins*. Addison-Wesley Professional.
- [4] Steinberg, D., Budinsky, F., Paternostro, M., Merks, E. 2008. *EMF: Eclipse Modeling Framework, 2nd Edition*. Addison-Wesley Professional.
- [5] Viyović, V., Maksimović M. and Perisić B. 2014. Sirius: A rapid development of DSM graphical editor. In *Proceedings of the IEEE 18th International Conference on Intelligent Engineering Systems* (Tihany, Hungary, July 3-5, 2014). INES 2014. IEEE, 233-238. DOI=<http://dx.doi.org/10.1109/INES.2014.6909375>.
- [6] University of Washington. Course and Syllabus Design. Online, retrieved on July 2016. <http://www.washington.edu/teaching/teaching-resources/preparing-to-teach/designing-your-course-and-syllabus/>.
- [7] Cornell University Center for Teaching Excellence. *Writing a Syllabus*. Online, retrieved on July 2016. <http://www.cte.cornell.edu/teaching-ideas/designing-your-course/writing-a-syllabus.html>.
- [8] Joeckel, G., Longhurst, M. *PDF syllabus builder: open source tool for online instructors, course developers and instructional designers*. Online, retrieved on July 2016. http://olc.onlinelearningconsortium.org/effective_practices/pdf-syllabus-builder-open-source-tool-online-instructors-course-developers-and-i.
- [9] Olsen, J., Meyer, J. *Curriculum Design Tool. Jump Rope*. Online, retrieved on July 2016. <https://www.jumpro.pe/features/curriculum-design-tool/>.
- [10] School Software Group. *Build Your Own Curriculum (BYOC): Evaluating a K–12 online curriculum management system*. ProQuest. Online, retrieved July 2016. <http://www.schoolsoftwaregroup.com/>.

9.6 Appendix E : TESMA Sample Grading Sheet

Start time:		End Time:		Groupe n°:		
Student Last/First Name:		Reviewer:				
		0	1	2	3	4
Category	criteria					
General content of the talk <i>(40,00%)</i>	Expression (elocution/rythm/language)					
	Structure (intro/conclusion/structure/duration)					
Features <i>(60,00%)</i>	Presentation of the product user categories					
	main functional characteristics of the product (completness/correctness)					
	main non-functional characteristics of the product (completness/correctness)					
		0	1	2	3	4
	Qty / Grade					
Max points	Total Points/grade :					
20	Total:					

Figure 9.1: Grading sheet sample

9.7 Appendix E : TESMA Sample Grading Table

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	Degree:	BINFO	S3																				
2	Session:	February	2016																				
3	Lecture:	Software Engineering Project																					
4	Professor:	GUELFI Nicolas																					
5																							
6	Last	First	ID	Type	reg.	Note finale	Note Théo	groupe	Doc.#1	NG	ASS	ASS	Exp #1	Doc #2	NG	ASS	ASS	Exp #2	MESSIR	Inc#1	Inc#2		
7									UM				UM										
8						0,0	4,43	7	15,00	26	18	20	8,23	0,00	41	26	37	0,00		0,00	0,00	0,00	
9						12,5	12,55	2	13,00	35	22	24	10,38	15,45	31	25	39	9,90		8,33	8,33	14,17	
10						14,0	14,04	1	13,33	32	26	30	11,28	16,82	30	28	39	10,10		13,33	13,33	15,00	
11						14,5	14,27	5	14,61	45	42	33	15,38	13,44	22	27	31	8,33		13,33	13,33	18,33	
12						11,5	11,26	8	8,00	39	29	28	12,31	15,00	46	30	41	12,19		7,50	7,50	13	
13						12,5	12,60	5	14,61	45	42	33	15,38	13,44	22	27	31	8,33		10,00	10,00	12	
14						13,5	13,35	6	13,13	37	33	29	12,69	14,40	26	31	22	8,23		11,67	11,67	16,67	
15						12,0	11,96	6	13,13	37	33	29	12,69	14,40	26	31	22	8,23		11,67	11,67	16,67	
16						12,5	12,45	3	12,39	18	16	19	6,79	13,64	21	21	33	7,81		10,00	10,00	18,33	
17						5,0	4,97	8	8,00	39	29	28	12,31	0,00	0	0	0	0,00		11,67	11,67	0	
18						14,0	14,04	6	13,13	37	33	29	12,69	14,40	26	31	22	8,23		15,00	15,00	17,50	
19						13,5	13,42	7	15,00	26	18	20	8,21	14,00	41	26	37	10,83		12,50	12,50	15,00	
20						10,0	9,88	9	11,00	36	31	28	12,18	12,00	46	31	42	12,40		4,17	4,17	8	
21						10,5	10,58	9	11,00	36	31	28	12,18	12,00	46	31	42	12,40		5,00	5,00	12	
22						11,0	10,99	8	8,00	39	29	28	12,31	15,00	46	30	41	12,19		8,33	8,33	11	
23						12,0	11,75	7	15,00	26	18	20	8,21	14,00	41	26	37	10,83		5,83	5,83	12	
24						14,0	13,63	4	14,45	41	30	29	12,82	15,44	31	38	37	11,04		10,00	10,00	15,00	
25						14,0	13,63	4	14,45	41	30	29	12,82	15,44	31	38	37	11,04		10,00	10,00	15,00	
26						13,0	12,93	1	13,33	32	26	30	11,28	16,82	30	28	39	10,10		6,67	6,67	15	
27						11,5	11,34	3	12,39	18	16	19	6,79	13,64	21	21	33	7,81		8,33	8,33	13,33	
28						12,0	11,89	7	15,00	26	18	20	8,21	14,00	41	26	37	10,83		6,67	6,67	12	
29						13,0	13,07	1	13,33	32	26	30	11,28	16,82	30	28	39	10,10		6,67	6,67	15,83	
30						11	10,85	9	11,00	36	31	28	12,18	12,00	46	31	42	12,40		11,67	11,67	7	
31																							

grades

individuel

+