A Web-Portal with Semantic Web Technologies

Ingrid Vangkilde

Kgs. Lyngby 2003
IMM-THESIS-2003-31

A Web-Portal with Semantic Web Technologies

Ingrid Vangkilde

Kgs. Lyngby 2003

IMM-THESIS-2003-31

ISSN 1601-233X

PREFACE

This report is the documentation of my Master Thesis, which is the culmination of my Informatics Engineering studies. This thesis has been developed under the Division for Computer Science and Engineering (CSE), in the department of Informatics and Mathematical Modeling (IMM), at the Technical University of Denmark (DTU). Professors Jørgen Fischer Nilsson and Hans Bruun have supervised this project.

The motivation for this thesis is my interest in developing systems, especially web-based systems, and Internet technologies.

I would like to thank some of my teachers at DTU, for being an inspiration source during my studies and for motivating and guiding me in finding my areas of interest. These are, in alphabetical order: Anne E. Haxthausen, Flemming Stassen, Hans Bruun, Hans Rischel, Jens Thyge Kristensen, Jørgen Fischer Nilsson, Michael R. Hansen, Morten P. Lindegaard, Rolf Nevald and Tom Østerby.
I would also like to thank the Internet Mobile division of Bording Data A/S, were I worked during my studies, for giving me the opportunity of realizing the importance and relevance of my education.

Finally, I would like to thank my beloved Søren Thrane for the emotional support during the writing of this project.

Kongens Lyngby, 30 April 2003

Ingrid Vangkilde

ABSTRACT

The Semantic Web is an extension of the current web, where its contents contain information about their own meaning. This is achieved by adding metadata (i.e. data about data) to web documents. The Semantic Web is supposed to allow machines to "understand" the data they retrieve from the web.

This thesis will illustrate the development of a web portal that makes use of the Semantic Web. The web portal will have as main task to assist DTU students to plan semester courses at IMM. The content of the portal will be data about IMM courses and students, and of course data about this data.

The web portal being developed in this thesis project is required to extract information about IMM courses from other web sites (e.g. the DTU Course Catalogue and ACM Computing Classification System) and allow the people responsible for these courses to add more information about them. In the same way, DTU students must be able to feed the system with information about their own profiles. The system must then contain information about the meaning of this course information and student profiles, and how they relate to each other. The portal must then use this information to assist students to plan which courses they should follow in a specific semester. These requirements must be solved taking advantage of Semantic Web technologies.

In this project, the current possibilities in the area of Semantic Web will be studied. It will be examined what kind of problems this technology is able to solve, what its limitations are, etc. Through the development of the portal, the advantages and disadvantages of this new technology will be shown in its present stage of development, and how it can be exploited.

Keywords: Semantic web, description logics, XML, RDF, DAML+OIL, web, portal, Internet, ontology, knowledge base, classification, data constraints.

TABLE OF CONTENTS

1 INTRODUCTION

This thesis project is about developing a system, more precisely a web-portal, using Semantic Web technologies. The project has two main points: The development of a system and the study of a new technology.

The first part of the project, consisting of the development of a system, has as purpose to put in practice some of the knowledge I acquired during my studies, in such areas as object-oriented methods, programming techniques and best-practices, logics, Internet technologies, etc. The second part of the project, consisting of the study of Semantic Web technologies, has as purpose to exploit my ability to understand and utilize the result of new research areas of computer science.

This chapter describes the problem to be solved, explains the contents of this document and suggests what previous knowledge is necessary to take full advantage of it.

1.1 Problem description

The aim of this project is to build a web-portal where DTU students can be assisted in planning his/her semester schedule, by checking that the chosen courses for a semester comply with DTU regulations and with the student's preferences. Even though this system can be developed using different approaches, it is a requirement of this project that Semantic Web technologies be used in solving the problem.

The system is to take advantage of course information already available on the Internet (for example the online version of the DTU course catalogue), and enable teachers and students to input some more information into the system, if necessary.

The data structure of the system, corresponding to the domain model, will probably change often, as DTU regulations change almost every year. For this reason, this data structure must be stored in the system as data[1], allowing it to be modified without alterations to the application.

For simplification purposes, the project will focus on courses taught at the division of Computer Science and Engineering (CSE). The ontology used in relation to course topics will therefore be derived from the most recent ACM Computing Classification System [ACM]. The ontology used to describe DTU courses will only use the DTU regulation for master studies.

No security issues will be addressed in this project. The amount of information stored about students and their course choices will for this matter be held to a minimum.

[1] This means that the data structure is data about data, also known as metadata. This metadata can be handled as the rest of the system's data.

1.2 How to read this paper

To be able to fully understand this thesis report, it is required that the reader have a basic knowledge of logics and XML. Besides, a previous knowledge of the notation used in this document would ease its reading (see Notation at the end of this chapter).

This document primarily describes the software engineering process of developing a software system for solving the above mentioned problem, and consists of the following chapters:

Semantic Web Technologies – accounts for the status of this new technology, what is available, how it is intended to be used, and so on.

Domain analysis – provides an extensive understanding of the system's domain, determines its terminology and formalizes it in a model.

Requirements specification – specifies in detail what is required of the system, in terms of what it is supposed to solve, what is expected from its performance and how it is intended to be used.

System design – gives details about the choices made on how to accomplish the requirements specified in the previous chapter, including what will be used to implement the solution.

Implementation – accounts for the process of implementing what was intended in the system design, any problems encountered and considerations taken.

Test – enumerates and describes the tests performed on the system and on one of the tools that is crucial for its functioning.

Conclusion – states the results of the project and describes what was learned in the process.

This document finalizes with a references chapter mentioning all the material used to make this thesis, and the appendix containing all code and results that are too extensive to include in any of the chapters.

1.2.1 Notation

The model in the domain analysis is expressed using ER-diagrams [ER], Hasse diagrams and $\mathcal{SHIQ}$ expressions [DL].

The entity-relationship model is ideal to show relations among concepts. Entity-relationship models can be visualized graphically by using ER-diagrams. Lattice theory is ideal to show concept hierarchies, or better said concept classifications. Lattices can be visualized graphically by using Hasse diagrams.

Description Logics is much more expressive, being able to convey information about both relations and hierarchies. Unfortunately, there is no graphical visualization for description logics. A combination of ER-diagrams and Hasse-diagrams can be used to convey most of the information contained in a description logics model.

The description language used in this report is the subset of the attributive language $\mathcal{AL}$ called $\mathcal{SHIQ}$, i.e. the description language extended with concept constructors negation, transitive roles, role hierarchies, inverse roles, and quantified number restrictions. $\mathcal{SHIQ}$ has a restriction that roles that have a transitive sub-role must not occur in number restrictions.

The requirements specification and system design are expressed using UML (Unified Modeling Language) [USDP] to describe the system, more precisely use cases, collaboration diagrams and a class diagram.

2 SEMANTIC WEB TECHNOLOGIES

This chapter has as purpose to give a background on Semantic Web technologies. This will allow the reader to better understand the development process described in this report.

The Semantic Web is an activity leaded by the World Wide Web Consortium (W3C)[2] that has as goal to add well-defined meaning to Internet data, thus enabling people and computers to work in cooperation. The Semantic Web allows both human users and machines to query the Internet as if it were a database.

The Semantic Web is primarily focused on RDF/XML. RDF (Resource Description Framework) is a mechanism for describing data in any domain, i.e. data about data (metadata). RDF separates the semantic data from the rest of the system's data. RDFS (RDF Schema) can be used to describe the ontology used in the system. But RDF and RDFS are not much more than a way to describe the content of web data as if it were a database.

To make our data yet more powerful, we can make use of description logics to describe it, so we can do some reasoning with it. In this way, we can find not only information that is explicitly given, but also reach some new conclusions about the data. For achieving this extra power, an extension of RDF, called DAML+OIL, can be used.

DAML+OIL is a description logic language disguised in an XML format. DAML+OIL is developed by the Defense Advanced Research Projects Agency (DARPA) under the DARPA Agent Markup Language (DAML[3]) Program.

2.1 Resource Description Framework (RDF)

The web contains information that can be located via URLs[4]. The amount of information in the web is huge and grows very fast. One of the biggest problems on handling this information is how to find what we are searching for.

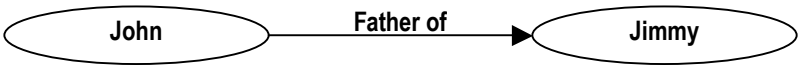
Web robots go through millions of URLs trying to find words in text that match our search. For example, if you are looking for fast food, and the page only mentions pizza and burgers, you won't find it by using a web robot.

Some people try to classify the web content others generated, by for example creating subject catalogues and site labels. This task is very ambitious, taking into account the amount of sites that exist currently, and will surely never be able to classify all the information. Besides, not everybody will agree on how the web content should be classified. The best people to classify the web content are the people creating the content.

[2] W3C Semantic Web Activity: http://www.w3.org/2001/sw/
[3] The DARPA Agent Markup Language Homepage: http://www.daml.org/
[4] URL – Uniform Resource Locator.

RDF is a framework for describing and interchanging metadata about web content. RDF is not tied to a specific syntax. The XML serialization syntax of RDF is suitable for the web and for applications, but is not very human-readable. For that purpose a graphical syntax is normally used. The graphical syntax represents RDF statements as triplets, and the whole model is a directed graph.

A RDF Statement consists of the combination of a subject (Resource), a predicate (Property), and an object (value). An example of a statement is "John is the father of Jimmy". Here John is the resource, Jimmy the value, and "father of" is the property. This statement can be viewed as the triple `(John, Father of, Jimmy)`, as a directed graph or as XML:




```
<rdf:Description about='http://foo/family.rdf#John'>
    <http://foo/family.rdf#Father_of
            rdf:resource='http://foo/family.rdf#Jimmy'/>
</rdf:Description>
```

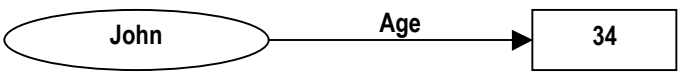
Notice that in the above example, both `John` and `Jimmy` are resources.

A Resource is anything that can have a URI[5] ; this includes all sites on the web (e.g. `http://foo/family.rdf`), as well as individual elements of an XML document (e.g. `http://foo/family.rdf#John`). Here the symbol # indicates that the element `John` is defined in the document `http://foo/family.rdf`.

URIs in XML tags and property names can be abbreviated by using XML-namespaces. For instance, we can define the substitution of the namespace prefix `fam` for `http://foo/family.rdf` and then write simply `fam:Father_of` instead of `http://foo/family.rdf#Father_of`:

```
<rdf:Description about='http://foo/family.rdf#John'>
    <fam:Father_of rdf:resource='http://foo/family.rdf#Jimmy'/>
</rdf:Description>
```

A Value or object can be a resource or a literal. A literal, which is just a string, can not be the subject of a statement, and therefore has no properties. An example of a statement where the value is a literal could be `(John, Age, 34)`:




```
<rdf:Description about='http://foo/family.rdf#John'>
    <fam:Age>34</fam:Age>
</rdf:Description>
```

[5] URI – Uniform Resource Identifier.

A Property or predicate relates subjects to objects. A Property is a resource because it may have properties of its own.

A Statement is also a resource, that again can have properties like who created the statement, when, etc.

2.2 Ontology

"An ontology is a set of concepts - such as things, events, and relations - that are specified in some way (such as specific natural language) in order to create an agreed-upon vocabulary for exchanging information."[6]

RDF enables us to create Vocabularies. Anybody can create a vocabulary, but some organizations or groups of users will probably agree on some vocabularies, and use them to classify their data.

RDF is simply a standard for creating ontologies. Different software products can use ontologies created by different people, as they will all be available on the web.

2.3 RDF Schema (RDFS)

RDFS extends RDF to include means to define property domains and ranges, class and subclass hierarchies, and property and subproperty hierarchies. RDFS also provides some additional modeling primitives as the `rdfs:label` that defines a human-readable name format, and the `rdfs:comment` that allow the developer to comment his work.

On the downside, RDFS is very weak, as it for example cannot describe simple constraints as cardinality constraints. An extension to RDFS is then necessary to be able to describe a complete ontology. More sophisticated languages, as Unified Modeling Language (UML) and Description Logics can be built on top of RDFS.

2.4 DAML+OIL

DAML+OIL, usually called just DAML, extends RDFS by adding description logics expressiveness to it. DAML allow to relate complex restrictions to class and property definitions. DAML extends RDFS in the following ways:

- Support of XML Schema Datatypes rather than just string literals, like dates, integers, decimals, etc.
- Additional restrictions on properties like cardinality constraints.
- Definition of classes by enumerations of their instances.
- Definition of classes by terms of other classes and properties (class expressions using unionOf, intersectionOf, complementOf, hasClass and hasValue).

[6] Whatis.com – http://whatis.techtarget.com/

- Ontology and instance mapping (sameClassAs, samePropertyAs, sameIndividualAs, differentIndividualFrom) permitting translation between ontologies.
- Additional hints to reasoners (disjointWith, inverseOf, TransitiveProperty, UnambiguousProperty).

The requirements specified for DAML prioritize rules and queries very much. Most of the DAML constructors have been added to support rules and querying. On the other hand, the support of rules has been a neglected research area and therefore there is not much information about the subject and it is very difficult to find tools that support it.

Some people consider DAML to still be in its infancy, others consider it to be half way of its development, but all agree that it is not completely developed yet. Even though it is actually the recommended ontology language by the World Wide Web Consortium, a new project called Ontology Web Language (OWL) is being developed to replace DAML. The OWL project has removed some of the requirements specified for DAML, as rules, queries and services. Furthermore, there are not many tools available supporting OWL yet.

2.4.1 Description Logics

Description Logics are knowledge representation languages adapted for expressing knowledge about concepts and concept hierarchies, and they are very suited for providing structure to information.

Description Logics is a subset of First Order Logic, which is function-free and does not allow explicit variables. Description Logics has a reduced expressivity in favor of having greater decidability in inference procedures, compared to First Order Logic. Besides it permits a better structuring of the knowledge.

DAML is a description logics language based on XML; more specifically it is a XML version of $\mathcal{SHIQ}$ (Attributive Language with transitive roles ($\mathcal{S}$), role hierarchy ($\mathcal{H}$), inverse role ($\mathcal{I}$) and qualified number restriction ($\mathcal{Q}$)).

The following table contains the set of constructs available in $\mathcal{SHIQ}$, DAML and their semantics. The semantics is defined by an interpretation $(\Delta^I, \cdot^I)$, where Δ^I is a nonempty set (domain of discourse) and $\cdot^I$ is a function that maps every concept to a subset of Δ^I and every role to a subset of $\Delta^I \times \Delta^I$.

Construct	$\mathcal{SHIQ}$	DAML+OIL	Semantics
concept	C	Class	$C^I \subseteq \Delta^I$
role name	R	Property	$R^I \subseteq \Delta^I \times \Delta^I$
top	$\top$	Thing	Δ^I
bottom	$\top$	Nothing	$\varnothing$
union	$C \sqcup D$	unionOf	$C^I \cup D^I$
intersection	$C \sqcap D$	intersectionOf	$C^I \cap D^I$
negation	$\neg C$	complementOf	$\Delta^I \setminus C^I$
value restriction	$\forall R.C$	toClass	$\{a \in \Delta^I \mid \forall b.(a,b) \in R^I \rightarrow b \in C^I\}$
existential quantification	$\exists R.C$ $\exists R.\{x\}$	hasClass hasValue	$\{a \in \Delta^I \mid \exists b.(a,b) \in R^I \wedge b \in C^I\}$ $\{a \in \Delta^I \mid (a,x) \in R^I\}$
collection of individuals	$\{a_1 \ldots a_2\}$	oneOf	$\{a_1 \ldots a_2\}$
number restrictions	$\geq n\ R$ $\leq n\ R$ $=n\ R$	minCardinality maxCardinality cardinality	$\{a \in \Delta^I \mid \lvert\{b \in \Delta^I \mid (a,b) \in R^I\}\rvert \geq n\}$ $\{a \in \Delta^I \mid \lvert\{b \in \Delta^I \mid (a,b) \in R^I\}\rvert \leq n\}$ $\{a \in \Delta^I \mid \lvert\{b \in \Delta^I \mid (a,b) \in R^I\}\rvert = n\}$
concept agreement	$C \doteq D$	sameClassAs	$C = D$
role agreement	$R \doteq S$	samePropertyAs	$R = S$
concept hierarchy	$C \sqsubseteq D$	subClassOf	$C \subseteq D$
role hierarchy	$R \sqsubseteq S$	subPropertyOf	$R \subseteq S$
inverse role	R^-	inverseOf	$\{(b,a) \in \Delta^I \times \Delta^I \mid (a,b) \in R^I\}$
qualified number restriction	$\geq n\ R.C$ $\leq n\ R.C$ $=n\ R.C$	minCardinalityQ maxCardinalityQ cardinalityQ	$\{a \in \Delta^I \mid \lvert\{b \in \Delta^I \mid (a,b) \in R^I \wedge b \in C^I\}\rvert \geq n\}$ $\{a \in \Delta^I \mid \lvert\{b \in \Delta^I \mid (a,b) \in R^I \wedge b \in C^I\}\rvert \leq n\}$ $\{a \in \Delta^I \mid \lvert\{b \in \Delta^I \mid (a,b) \in R^I \wedge b \in C^I\}\rvert = n\}$

3 DOMAIN ANALYSIS

This chapter has as purpose to provide a better understanding of the domain of DTU courses and semester planning, to initiate the development of the web portal. This is done by first identifying all terms used in the domain, then finding the relations among these terms and constraints to these relations.

The chapter contains a term dictionary containing all the terms found relevant for the domain. Some simplifications to the domain are stated for clarification purposes. The relations between those terms are illustrated by Entity-Relationship diagrams (ER-diagrams). Some of the concepts in the model can and must be classified. To express these classifications, Hasse-diagrams will be used.

Some of the domain model constraints are not easily illustrated through ER-diagrams or Hasse-diagrams. These are therefore expressed in natural language. The analysis is extended to include some questions the system must be able to handle (this is done for exemplification purposes only). Finally, the whole domain model is formalized through description logics, which can be easily translated to DAML+OIL in the implementation phase of the project.

3.1 Term Dictionary

The following list contains all relevant terms used in the domain of this project.

ACM Computing Classification System – taxonomy for describing topics in computer science. This will be used as a common terminology when describing the topics covered by courses and the students' topics of interest.

AMS course – a course in the area of work, environment and society (in Danish: Arbejde, Miljø og Samfund), which students following the complete master study are required to pass some of.

Any course – a course that is being taught at DTU (see Course) or any course that has ever been taught at DTU, even if it is not taught any more.

Complete master – the 5-year master study at DTU. Students following this study type may come directly from high school.

Core course – a course in the areas of mathematics, physics and chemistry, which students following the complete master study are required to pass (in Danish: Kernestof).

Course – corresponds to any course currently being taught at DTU, contained in the current online version of the Course Catalogue.

Course name – the English title of the course.

Course schedule – the periods of time a course is given at. The schedule must correspond to one or two seasons, and it may have several modules.

Exotic course – (also called a humanistic course) a course that is not in the area of engineering. Only a few courses at DTU fall under this category.

International mandatory course – a subset of specialization courses that international master students must pass some of.

International master – the 2-year master study at DTU, aimed at students that have obtained at least a bachelor degree from a non-Danish university.

International master course – the courses that students following the international master study are allowed to take.

International specialization course – the specialization courses that are allowed to international master students.

Language – at the present time, it can only be English or Danish, corresponding to the language that a course is taught at and the language a student speaks. It is assumed that students that speak Danish are also able to follow courses in English.

Line package – a group of courses, where all the courses are compulsory for complete master students (in Danish: Fagpakke). These courses are the first courses that must be passed, when taking a complete master study of 5 years.

Line package AMS course – an AMS course that is placed in a line package.

Line package core course – a core course that is placed in a line package.

Line package course – the compulsory courses contained in the line package.

Mandatory core course – a mandatory course that is also a core course. Students not following the complete master may not take courses that are core, but not mandatory core.

Mandatory course – a group of courses in the areas of mathematics and physics, from which students following the master study of 2 years must pass some of.

Mandatory line package course – a mandatory course that is also a line package course. Students not following the complete master may not take courses that are line package, but not mandatory line package.

Master – the 2-year master study at DTU (in Danish: Overbygning). Students must have obtained a title corresponding to a bachelor degree or equivalent degree prior to following this type of study.

Master course – all the courses that master students are allowed to follow. Courses not in this group are only allowed to students following the complete master study.

Module – a specific period of time in a week when courses are normally taught. A course may have a special module, e.g. courses taught at special periods of time, like evening courses, etc. Examples of modules are 1A, 1B, 2A, 2B, etc.

Overlap – two or more courses overlap when they cover mostly the same topics. If a student passes two or more courses that overlap with each other, he will only get credit points for one of them. It is therefore not recommended to take courses that overlap.

Prerequisite – topics that are considered known in the course stating the prerequisite (see Prerequisite group). Background knowledge assumed in a course.

Prerequisite group – a group of courses representing one prerequisite. The student must have passed at least one of the courses in the group in order to fulfill the prerequisite.

Profile – Information about the student, including the language he speaks, the courses he has passed, topics of interest, etc.

Season – either fall or spring.

Semester – the half of the year corresponding to a season, containing courses that the student plans to take.

Specialization course – a course that students may take in order to specialize themselves in a specific area.

Student – the main user of the system, person who follows a master study at DTU.

Topic – the content of the courses, or area in which students are interested, expressed in terms of the ACM classification system.

Type of study – either the whole engineering study (complete master, in Danish: Civilingeniørstudie), the master part of the engineering study (in Danish: Overbygning) or the international master in engineering.

3.2 Domain simplifications

This section contains some of the simplifications that have been assumed when developing this domain model.

Course schedule – Some courses are only offered in odd or even years, or maybe only in a specific year. This information has been ignored, as it changes very often and would require the system to be updated manually every time it happens. The system will then make no guarantee that the courses planned are actually offered in the year the student plans to take them.

Line package – In this project we are mostly concerned with the Informatics line package.

Prerequisite – DTU have different kinds of prerequisites (compulsory, recommended and desired). In this project it will be assumed that all prerequisites are only recommended.

Type of study – There are actually other types of study at DTU (Ph.D. study, Open University, Dairy and Food Science study and Bachelor study). This project will only deal with Master Studies.

3.3 ER-diagrams

The following ER-model has as goal to give a graphical overview of the domain concepts (or entities) and the relations among them. Every entity class and entity property in the following diagrams has already been described in detail in the term dictionary. Additional constraints to the model, not illustrated in the diagram are found in the Constraints section of this chapter.

The whole ER-model has been divided into 3 diagrams, for readability purposes. The first diagram shows the most important course relations. The second diagram relates to student profiles. The last diagram presents the structure of semester plans, which are what the system must help the student put together.

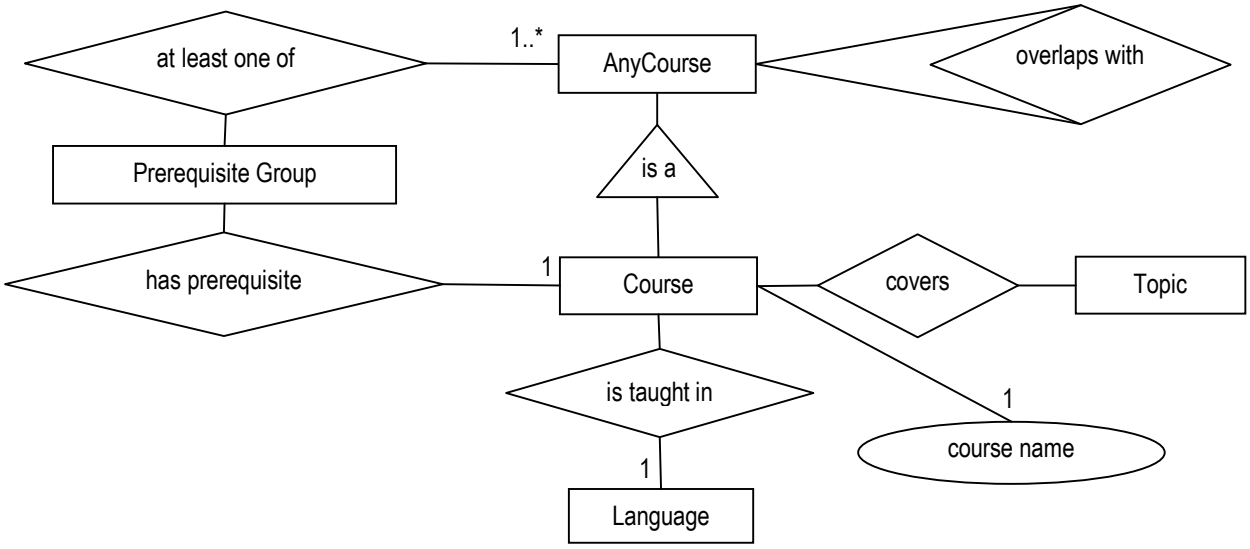


Figure 3-1: ER-diagram for Course.

Figure 3-1 contains the properties and relations between courses that are relevant for this project. The AnyCourse entity represents a course that is taught or has been taught at DTU, while the Course entity represents only the courses that are currently taught at DTU, and therefore also found in the newest version of the Course Catalogue.

Notice the solution found for course prerequisites. It is not enough to enumerate the prerequisites of a course, as some times only one prerequisite from a group must have been fulfilled in order to obtain the required knowledge for the course. Here is an example of this problem: The course 02110 Algorithms and Data Structures II, has as prerequisite 02105 Algorithms and Data Structures I, and one of the courses 02100, 02199 or 02115, which are all courses in Introductory Programming. This situation will be mapped to the course diagram in the following manner: 02110 has two prerequisite groups, the one containing only 02105, and the other containing 02100, 02199 and 02115.

Topics covered by courses follow the structure given in the ACM Computing Classification System [ACM 1998].

The convention used in this project about Course Topic and Student Interest classification is as follows: The course must be related to topics that are actually covered by the course. The student specifies the topics he is interested in. Only the courses that cover exactly the topics he selects or any of its subtopics will comply to the student wishes. An example of this is a course that covers Memory Structures, which is a subtopic of Hardware. A student that is interested in Hardware will of course be presented with this course. On the other hand, if the student has specified his interest to be Virtual Memories, which is a subtopic of Memory Structures, the student will not be presented with this course, even though it may cover this topic in some way.

The following diagram expresses the information enclosed in student profiles.

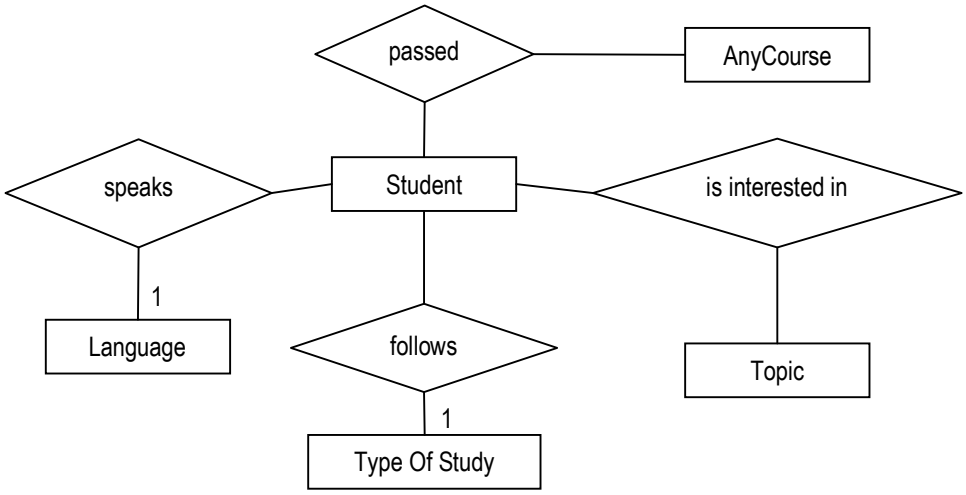


Figure 3-2: ER-diagram for student profiles, their relation to courses and ACM topics.

The next diagram represents the semester plan that the student is trying to put together, with the assistance of the system. Notice that a course can be taught at more than one schedule. For example a course that is taught both in the fall and spring season, and in modules 1A and 1B in each season would have the following schedule: Fall 1A, fall 1B, spring 1A, spring 1B. Some courses may lack the isTaughtAt information, causing them to have no schedules.

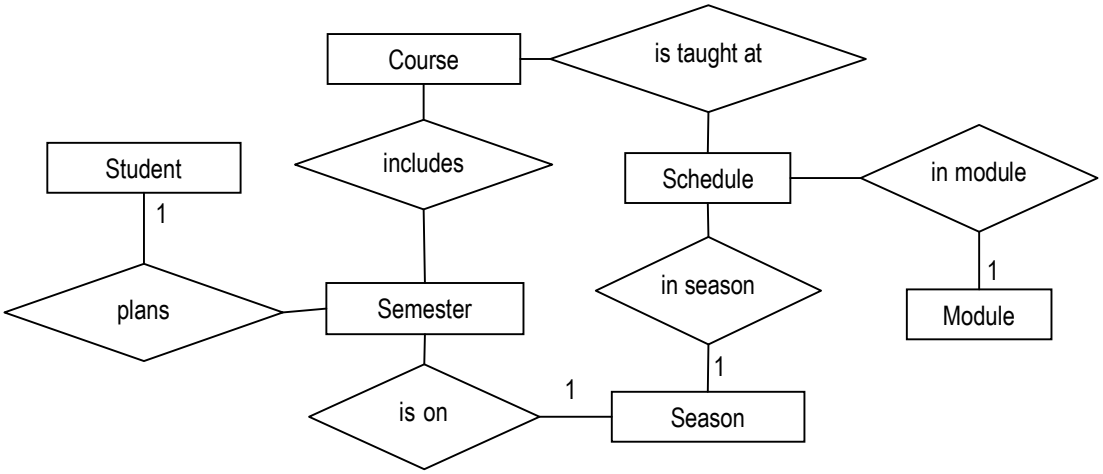


Figure 3-3: ER-diagram for semester plans, their relation to student profiles and courses.

3.4 Classifications

A Hasse diagram is a directed graph for a partially ordered set which does not have loops and arcs implied by the transitivity. A Hasse diagram is the ideal model to graphically visualize classification of concepts. Some of the concepts previously shown in the ER-diagrams have some classifications that will be expressed in the following. The first diagram is concerned with course classification. Each course at DTU has a well-defined type, which indicates which students must, may or may not follow it.

The Any Course type refers to all courses at DTU, even those that no longer exist.

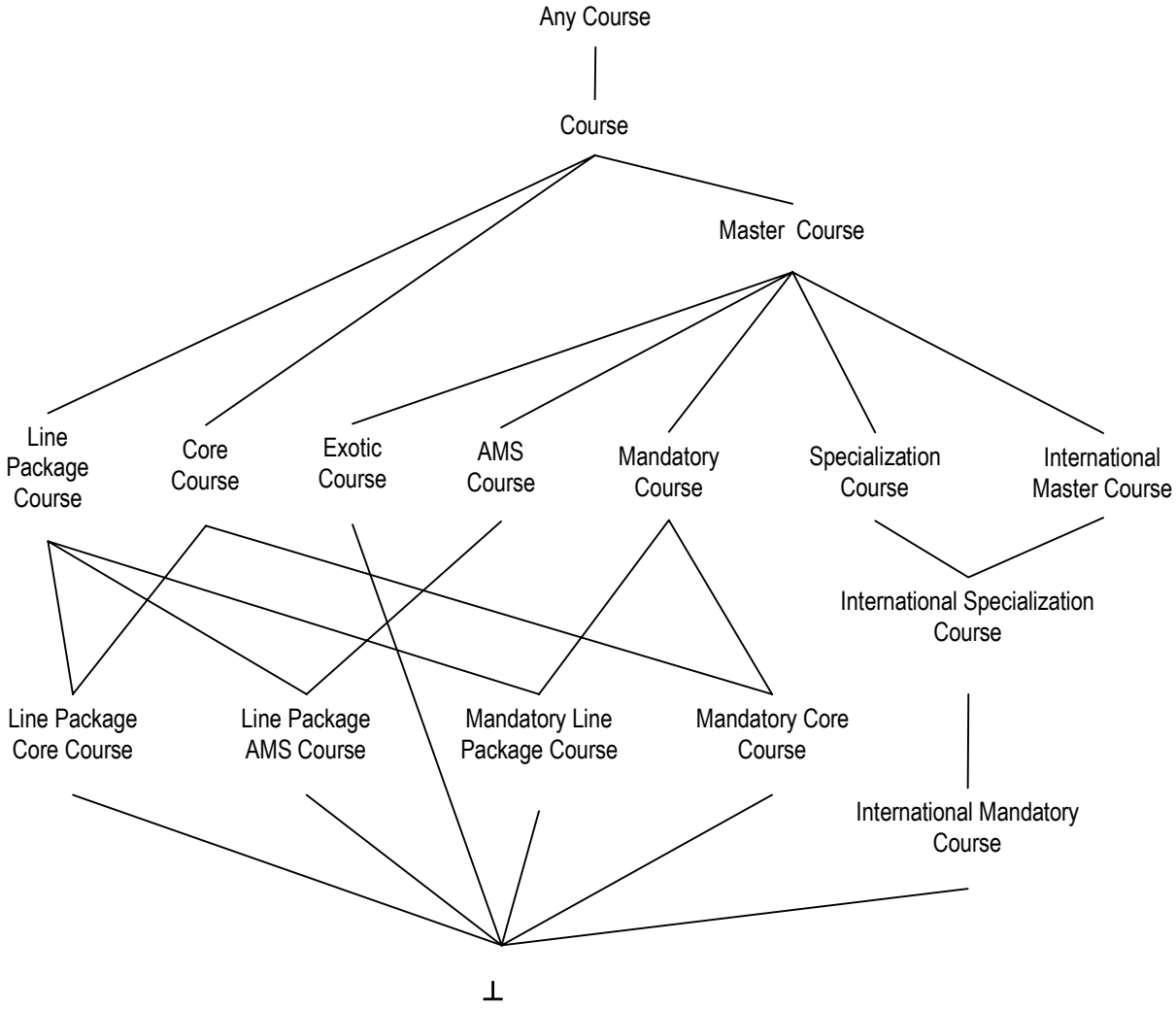


Figure 3-4: Hasse diagram for Course classification.

In the Course diagram, there are some course types that are in a sense a little artificial, as the case of Line Package Core Course. Notice that those types have multiple inheritances. The model assumes that if two course types are not in an inheritance line, they are disjoint. An example of this is that there exists no course that is both an AMS Course and a Specialization Course at the same time (therefore no entity class inherits from both these

classes), but there are many courses that are both Core Course and Line Package Course at the same time (namely the Line Package Core Course entities).

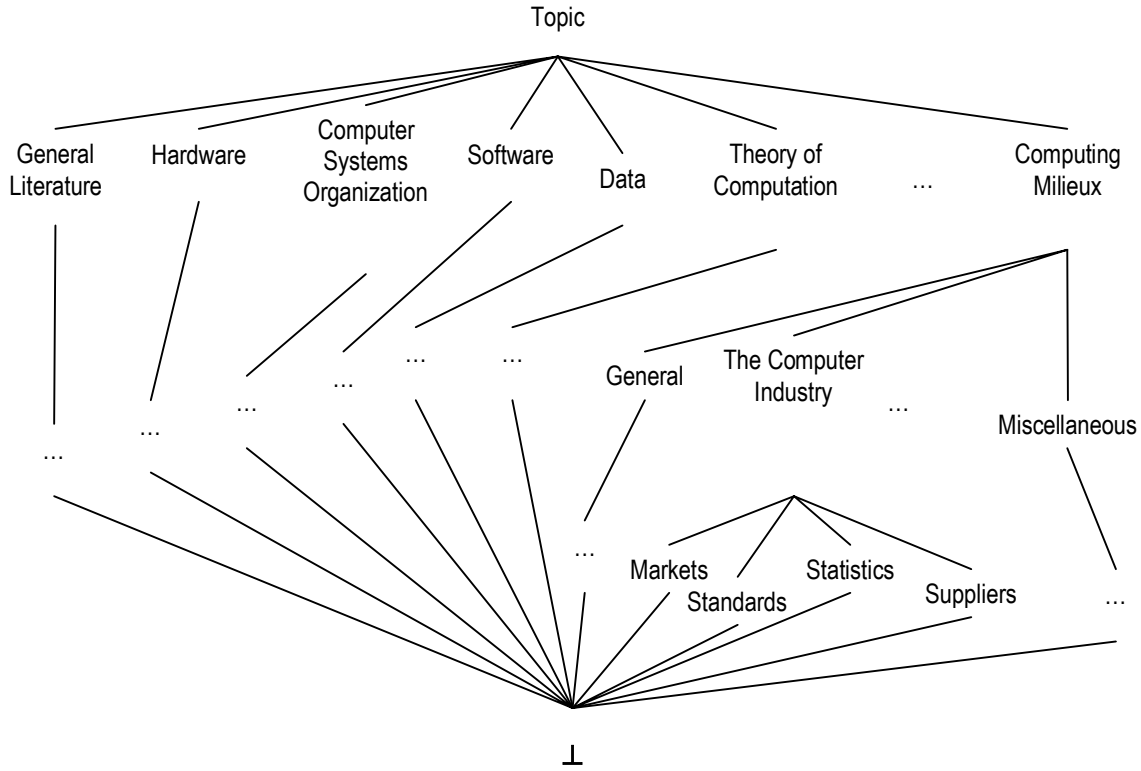


Figure 3-5: Part of the Hasse diagram for the ACM Computing Classification System[7].

The above classification is the ACM Computing Classification System. This classification is too large to be displayed graphically, and therefore only the first level and some examples of the other levels are illustrated here. The whole ACM tree can be found at the ACM homepage[8].

The ACM Computing Classification System contains many classification topics that are no longer used, but are still present for searching previously classified documents. As all courses will be classified by the new classification system, the retired topics will not be included in this project.

[7] Copyright 2002, by the Association for Computing Machinery, Inc. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permission to republish from: Publications Dept., ACM, Inc. Fax +1 (212) 869-0481 or E-mail permissions@acm.org.

[8] http://www.acm.org/class/1998/ccs98.html

The type of study entity can either be of type Complete Master, Master or International Master.

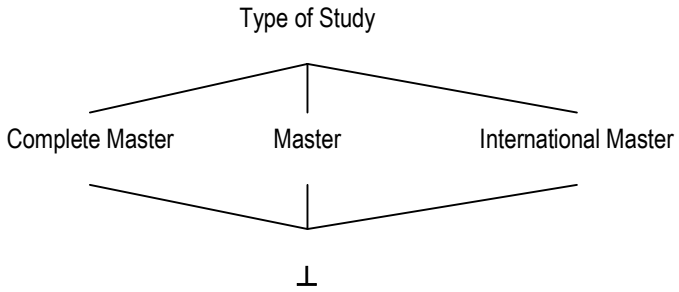


Figure 3-6: Hasse diagram for Type of Study classification.

The season entity can either be Fall or Spring.

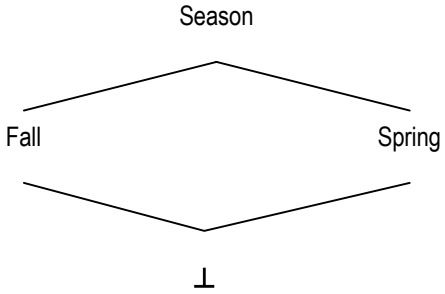


Figure 3-7: Hasse diagram for Season classification.

The language entity can either be English or Danish.

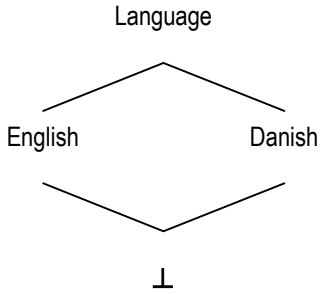


Figure 3-8: Hasse diagram for Language classification.

There are 12 different modules, each corresponding to either morning or afternoon of a given weekday, or a 3-week course. Some courses are placed at special schedules, as for example evening courses. These courses are classified as Other.

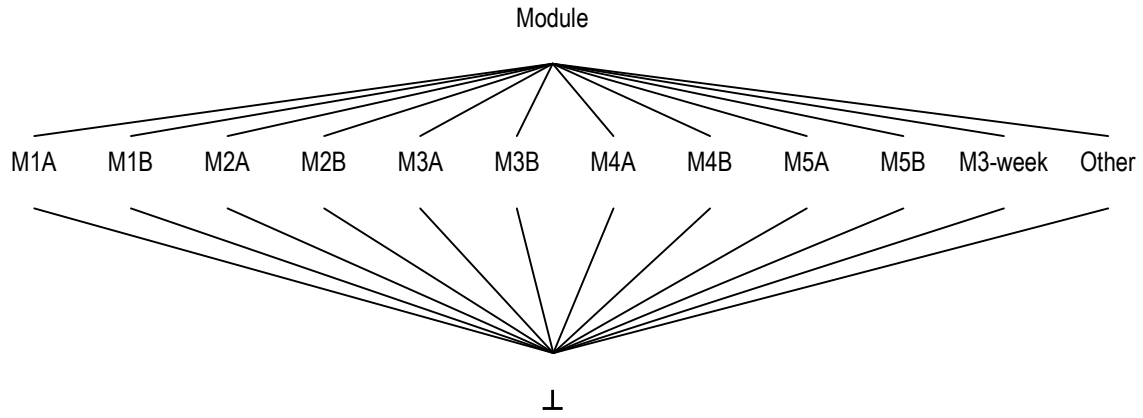


Figure 3-9: Hasse diagram for Module classification.

3.5 Description Logics Model

The following model is a formalization of the ER-diagrams and the Hasse-diagrams stated above. This model provides a very formal and precise description of the domain. In the following it is assumed that the intersection of any pair of concepts not listed in the model corresponds to the empty set BOTTOM.

The first part of the model simply enumerates all the concepts in the Domain:

```
⊤ ≐ AnyCourse ⊔ PrerequisiteGroup ⊔ Topic ⊔ Language
    ⊔ Student ⊔ TypeOfStudy ⊔ Semester ⊔ Schedule ⊔ Season
    ⊔ Module
```

The following part of the model corresponds to diagrams 3-1 to 3-3, i.e. the ER-diagrams:

```
AnyCourse ⊑ ∀overlapsWith.AnyCourse
          ⊔ ∀atLeastOneOf⁻.PrerequisiteGroup
          ⊔ ∀passed⁻.Student

Course ⊑ (= 1 courseName)
         ⊓ ∀hasPrerequisite.PrerequisiteGroup
         ⊓ ∀covers.Topic
         ⊓ (=1 isTaughtIn.Language)
         ⊓ ∀includes⁻.Semester ⊓ ∀isTaughtAt.Schedule
```

```
PrerequisiteGroup ⊑ (= 1 hasPrerequisite⁻.Course)
                    ⊓ (≥ 1 atLeastOne.AnyCourse)

Topic ⊑ ∀covers⁻.Course ⊔ ∀isInterestedIn⁻.Student

Language ⊑ ∀isTaughtIn⁻.Course ⊔ ∀speaks⁻.Student

TypeOfStudy ⊑ ∀follows⁻.Student

Student ⊑ (= 1 follows.TypeOfStudy)
          ⊓ (= 1 speaks.Language)
          ⊓ ∀isInterestedIn.Topic
          ⊓ ∀passed.AnyCourse
          ⊓ ∀plans.Semester

Semester ⊑ (= 1 isOn.Season)
           ⊓ (= 1 plans⁻.Student)
           ⊓ ∀includes.Course

Schedule ⊑ (= 1 inSeason.Season) ⊓ (= 1 inModule.Module)

Season ⊑ ∀isOn⁻.Semester ⊔ ∀inSeason⁻.Schedule

Module ⊑ ∀inModule⁻.Schedule
```

The next part of the model corresponds to diagram 3-4, i.e. the Hasse-diagram for Course classification:

```
AnyCourse ≐ AnyCourse ⊔ Course

Course ≐ LinePackageCourse ⊔ CoreCourse ⊔ MasterCourse

LinePackageCourse ≐ LinePackageCoreCourse ⊔ LinePackageAMSCourse
                    ⊔ MandatoryLinePackageCourse
                    ⊔ LinePackageCourse

CoreCourse ≐ LinePackageCoreCourse ⊔ MandatoryCoreCourse
             ⊔ CoreCourse

MasterCourse ≐ ExoticCourse ⊔ AMSCourse ⊔ MandatoryCourse
```

```
                ⊔ SpecializationCourse ⊔ InternationalMasterCourse
                ⊔ MasterCourse

LinePackageCoreCourse ≐ LinePackageCourse ⊓ CoreCourse

LinePackageAMSCourse ≐ LinePackageCourse ⊓ AMSCourse

MandatoryLinePackageCourse ≐ LinePackageCourse ⊓ MandatoryCourse

MandatoryCoreCourse ≐ CoreCourse ⊓ MandatoryCourse

InternationalSpecializationCourse ≐ SpecializationCourse
                                  ⊓ InternationalMasterCourse

InternationalMandatoryCourse ⊑ InternationalSpecializationCourse
```

The next part of the model corresponds to diagram 3-5, i.e. the Hasse-diagram for the ACM Computing Classification System. Once again, the whole model is too large to be displayed in this document, and therefore only a part of the model is shown in the following:

```
⊤ ≐ GeneralLiterature ⊔ Hardware ⊔ ComputerSystemsOrganization
    ⊔ Software ⊔ Data ⊔ TheoryOfComputation
    ⊔ MathematicsOfComputing ⊔ InformationSystems
    ⊔ ComputingMethodologies ⊔ ComputerApplications
    ⊔ ComputingMilieux

GeneralLiterature ≐ GeneralLiterature ⊔ AGeneral
                  ⊔ IntroductoryAndSurvey ⊔ Reference
                  ⊔ AMiscellaneous

AGeneral ≐ AGeneral ⊔ BiographiesAutobiographies
         ⊔ ConferenceProceedings ⊔ GeneralLiteraryWorks

...

TheoryOfComputation ≐ TheoryOfComputation ⊔ FGeneral
                    ⊔ ComputationByAbstractDevices
                    ⊔ AnalysisOfAlgorithmsAndProblemComplexity
                    ⊔ LogicsAndMeaningsOfPrograms
                    ⊔ MathematicalLogicAndFormalLanguages
```

```
                    ⊔ FMiscellaneous

...
```

The next part of the model corresponds to diagrams 6 to 9, i.e. the Hasse-diagrams for the classification of types of study, seasons, languages, and modules:

```
TypeOfStudy ≐ CompleteMaster ⊔ Master ⊔ InternationalMaster
              ⊔ TypeOfStudy

Season ≐ Fall ⊔ Spring

Language ≐ English ⊔ Danish

Module ≐ M1A ⊔ M1B ⊔ M2A ⊔ M2B ⊔ M3A ⊔ M3B ⊔ M4A ⊔ M4B ⊔ M5A
         ⊔ M5B ⊔ M3-week ⊔ Other
```

3.6 Constraints

The constraints stated in this section basically derive from DTU regulations and common sense.

The following constraints are expressed in natural language, predicate logic (PL) and in description logics (DL). This is done to obtain a complete understanding of these constraints, as they are a key aspect of this system. Only constraints not already explicitly shown in the ER-diagrams and Hasse-diagrams are stated here.

Constraints 1 to 9 are concerned with the semesters being planned by students. These constraints must be checked to assure that the student is not planning a semester that goes against DTU regulations or common sense. Constraints 10 and 11 are concerned with DTU courses, and they must only be checked to assure that all DTU courses comply with the regulations and common sense. In a sense, the system is only to check that semester plans are done correctly, and therefore only constraints 1 to 9 are really important in this context. On the other hand, if constraints 10 and 11 are not fulfilled, it could interfere with the planning of semesters, for example in the case of a course that can never be planned. In the latter case, the corresponding authority should be contacted about the problem.

Constraint 1 - *The "overlaps with" relation is symmetric. This constraint is important when checking for constraint 6.*

(PL) $\forall(\texttt{overlapsWith(X,Y)} \leftrightarrow \texttt{overlapsWith(Y,X)})$

This reads: *For all courses, if and only if a course overlaps with another then the second course also overlaps with the first one.*

(DL) $\texttt{overlapsWith} \doteq \texttt{overlapsWith}^{-}$

This reads: *The set of courses that overlap with other courses is equal to the set of the inverse relation.*

Constraint 2 - *Students following the master study may only plan to take master courses.*

(PL) `∀((follows(X,T) ∧ isMaster(T)) →`
`((plans(X,S) ∧ includes(S,C)) → isMasterCourse(C)))`

This reads: *For all students, semesters and courses, if a student follows a master type of study then, if the student plans a semester and the semester includes a course, then the course is a master course.*

(DL) ∀follows.Master ⊑ ∀plans.(∀includes.MasterCourse)

This reads: *The set of all students that follow a master type of study is a subset of the set of all students that only plan semesters that only include master courses.*

Constraint 3 - *Students following the international master study may only plan to take international master courses.*

(PL) `∀((follows(X,T) ∧ isInternatMaster(T)) →`
`((plans(X,S) ∧ includes(S,C)) → isInternatMasterCourse(C)))`

This reads: *For all students, semesters and courses, if a student follows an international master type of study then, if the student plans a semester and the semester includes a course, then the course is an international master course.*

(DL) `∀follows.InternationalMaster ⊑`
`∀plans.(∀includes.InternationalMasterCourse)`

This reads: *The set of all students that follow an international master type of study is a subset of the set of all students that only plan semesters that only include international master courses.*

Constraint 4 - *A student who does not speak Danish may not plan to take a course taught in Danish.*

(PL) `∀(speaks(X,L) ∧ isEnglish(L) →`
`(plans(X,S) ∧ includes(S,C) → isTaughtIn(C,L) ∧ isEnglish(L)))`

This reads: *For all students, languages, semesters and courses, if a student speaks English then, if the student plans a semester and the semester includes a course then, the course is taught in English.*

(DL) `∀speaks.English ⊑ ∀plans.(∀includes.(∀isTaughtIn.English))`

This reads: *The set of all students that speak English is a subset of the set of all students that only plan semesters that only include courses taught in English.*

Constraint 5 - *All courses in a semester plan must be taught in the season corresponding to that semester.*

(PL) $\forall(\text{includes}(S,C) \wedge \text{isOn}(S,Se) \rightarrow \exists Sc(\text{isTaughtAt}(C,Sc) \wedge \text{inSeason}(Sc,Se)))$

This reads: *For all semesters, courses and seasons, if the semester includes a course and the semester is in a given season then there must exist a schedule so that the course is taught at this schedule and this schedule is in the given season.*

(DL) $\bot \doteq \exists \text{includes}^{-}.(\exists \text{isOn.Fall}) \sqcap \neg(\exists \text{isTaughtAt}.(\exists \text{inSeason.Fall}))$

$\bot \doteq \exists \text{includes}^{-}.(\exists \text{isOn.Spring}) \sqcap \neg(\exists \text{isTaughtAt}.(\exists \text{inSeason.Spring}))$

This reads: *The intersection of the set of courses included in Fall semesters and the complement set of courses taught in Fall schedules is empty. The same constraint applies to Spring semesters and schedules.*

The following constraints contain nominals, i.e. variables that must be instantiated to a particular semester plan in order to be checked. This is a deviation from the Description Logics, which is necessary in order to express the constraints.

Constraint 6 - *A semester plan may not contain two or more courses that overlap with each other.*

(PL) $\forall(\text{includes}(S,C1) \wedge \text{includes}(S,C2) \rightarrow \neg\text{overlapsWith}(C1,C2))$

This reads: *For all semesters and courses, if the plan includes a course and the plan includes another course then the first course does not overlap with the second course.*

(DL) $\bot \doteq \exists \text{includes}^{-}.(\alpha) \sqcap \exists \text{overlapsWith}.(\exists \text{includes}^{-}.(\alpha))$

This reads: *The intersection of the set of courses included in a semester* α *and the set of courses that overlap with the courses included in the semester* α *is empty.*

Constraint 7 - *A semester plan may not contain more than one course at the same module.*

(PL) $\forall S(\text{includes}(S,C1) \wedge \text{includes}(S,C2) \rightarrow \neg(\text{isTaughtAt}(C1,Sc1) \wedge \text{inModule}(Sc1,M) \wedge \text{isTaughtAt}(C2,Sc2) \wedge \text{inModule}(Sc2,M))$

This reads: *For all semesters, courses, schedules and modules, if the plan includes a course and the plan includes another course then, it is not possible that the first course is taught at a schedule and the second course is taught at another schedule and both schedules are in the same module.*

(DL) $\top \sqsubseteq \leq 1\ (\exists \text{include}^{-}.(\alpha) \sqcap \exists \text{isTaughtAt}.(\exists \text{inModule.M1A}))$

$\top \sqsubseteq \leq 1\ (\exists \text{include}^{-}.(\alpha) \sqcap \exists \text{isTaughtAt}.(\exists \text{inModule.M1B}))\ \ldots$

This reads: *There is at most one course in the intersection of the set of courses included in a semester α and the set of courses taught in a schedule of module M1A. The same constraint applies to modules M1B, M2A, M2B, etc.*

The following constraints also contain nominals, but these must be instantiated to a particular student plan in order to be checked.

Constraint 8 - *No student may plan to take a course he has already passed.*

(PL) $\forall(passed(X,Y) \rightarrow (plans(X,S) \rightarrow \neg includes(S,Y)))$

This reads: *For all students, semester plans and courses, if a student passed a course then, if the student plans a semester then the semester may not include that course.*

(DL) $\bot \doteq \exists passed^{-}.(\alpha) \sqcap \exists includes^{-}.(\exists plans^{-}.(\alpha))$

This reads: *The intersection of the set of the courses passed by a student* α *and the set of the courses included in the semesters planned by the student* α *is empty.*

Constraint 9 - *No student may plan to take a course that overlaps with a course he already passed.*

(PL) $\forall(plans(X,S) \wedge includes(S,C1) \rightarrow (passed(X,C2) \rightarrow \neg overlapsWith(C1,C2)))$

This reads: *For all students, semester plans and courses, if a student plans a semester and the semester includes a course then, if the student passed a second course then this course does not overlap with the first course.*

(DL) $\bot \doteq \exists passed^{-}.(\alpha) \sqcap \exists overlapsWith.(\exists includes^{-}.(\exists plans^{-}.(\alpha)))$

This reads: *The intersection of the set of the courses passed by a student* α *and the set of the courses that overlap with the courses included in the semesters planned by the student* α *is empty.*

The last two constraints are not very important to the system, as they will not be used to validate semester plans or students, but only courses in the Course Catalogue.

Constraint 10 - *Prerequisites of a course may not have the course itself as a prerequisite.*

(PL) $\forall(\neg isPrerequisite(X,X))$

This reads: *For all courses no course is a prerequisite of itself.*

(DL) $\bot \doteq \alpha \sqcap \exists atLeastOneOf^{-}.(\exists hasPrerequisite^{-}.(\alpha))$

This reads: *The intersection of the set containing a course* α *and the set of the courses in a prerequisite group of the course* α *is empty.*

Constraint 11 - *The "has prerequisite" relation together with the "at least one of" relation is transitive. This is to be understood as, if a course A has a prerequisite B in one of its prerequisite groups, B is considered a prerequisite of A. If C is a prerequisite of B it is automatically also a prerequisite of A.*

(PL) $\forall(isPrerequisite(X1,X2) \rightarrow ((hasPrerequisite(X2,G) \wedge atLeastOneOf(G,X1)) \vee (isPrerequisite(X1,Z) \wedge isPrerequisite(Z,X2))))$

This reads: *For all courses and prerequisite groups, if a course is a prerequisite of another course then, the second course has a prerequisite group and the prerequisite*

group contains the first course, or the first course is a prerequisite of a third course that in turn is a prerequisite of the second course.

(DL) $(\exists hasPrerequisite.(\exists atLeastOneOf.\top))^{+} \sqsubseteq \exists hasPrerequisite.(\exists atLeastOneOf.\top)$

This reads: *The relation between courses and the courses included in their prerequisite groups is transitive.*

3.7 User Queries

This section enumerates some of the questions that students may be interested in asking the system. These questions are only examples of what the system could be able of answering. And of course, the answer to these questions must comply with the above-mentioned constraints.

A formalization of these questions in description logics is also given here. The user queries will contain some specific instances of a concept, also known as nominals. The nominals in the following questions will be represented by variables. These variables must be substituted by the according nominals when the results must be found.

1. *What courses cover my areas of interest? (Student α)*

$$result \doteq \forall covers.(\forall isInterestedIn^{-}.\alpha)$$

2. *What courses are there which I have completely fulfilled their prerequisites?(Student ψ)*

$$G1 \doteq \forall atLeastOneOf.(\forall passed^{-}.(\psi))$$
$$C2 \doteq \forall hasPrerequisite.(G1)$$
$$G3 \doteq \forall hasPrerequisite^{-}.(C2)$$
$$C4 \doteq \forall\ hasPrerequisite.(G3 \sqcap \neg G1)$$
$$result \doteq C2 \sqcap \neg C4$$

Question 2 is very complex; therefore an explanation of the description logics statement is given here. Venn diagrams for the example are found in diagram 3-10:

Each course that the student has passed may be in zero, one or more prerequisite groups. A set of all the prerequisite groups that has at least one of the courses passed by the student is stated in G1.

Each prerequisite group in G1 is a prerequisite of exactly one course. A set of all the courses of the groups in G1 is stated in C2. The courses passed by the student are

completely disjoint from C2 because of constraint 6. C2 is the set of courses that the student has fulfilled some prerequisites.

The courses in C2 have one or more prerequisite groups as prerequisites. Some of these prerequisite groups do not have any of the courses passed by the student, others do. The set of these groups is stated in the set G3. This set G3 contains the set G1, i.e. G1 is a subset of G3. If we remove G1 from G3, i.e. G3\G1, we get the set of prerequisite groups that do not have courses passed by the student.

Each group in the set G3\G1 is a prerequisite of exactly one course. A set of all the courses of the groups in G3\G1 is stated in C4. C4 is a subset of C2. C4 is exactly the set of courses that the student has fulfilled some of the prerequisites but not all. If we remove the set C4 from C2, we obtain the set of all courses that the student has fulfilled exactly all of the prerequisites, which is what we have been looking for.

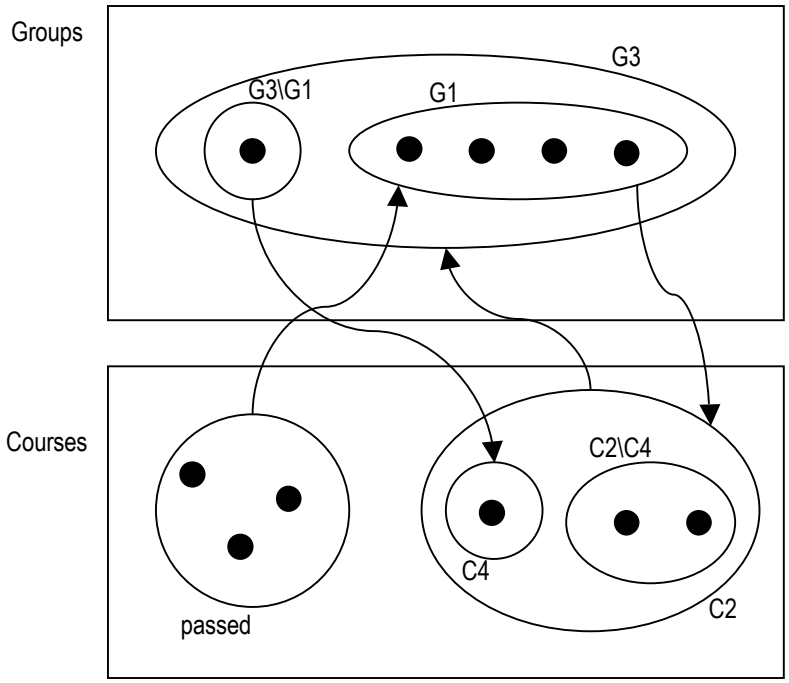


Figure 3-10: Venn-diagram for question 3.

4 REQUIREMENTS SPECIFICATION

The domain of the system to be developed has been extensively analyzed. Now the problem to be solved by the system must be specified formally. This chapter contains all the requirements that the system to be built must fulfill, expressed as a Use Case Model, Supplementary Requirements and a Mock-up Model. No technical details about how the requirements are to be met are given here, but in the design section.

The system to be developed is a web portal where students can enter information about themselves and create semester plans containing the courses they intend to follow. The system must then warn the student if the planned semester goes against the constraints described in the domain analysis, corresponding to DTU rules, or against any options the student has solicited (e.g. topics of interest, prerequisites, etc.).

4.1 Use Case model

The following use case model expresses the functional requirements of the system.

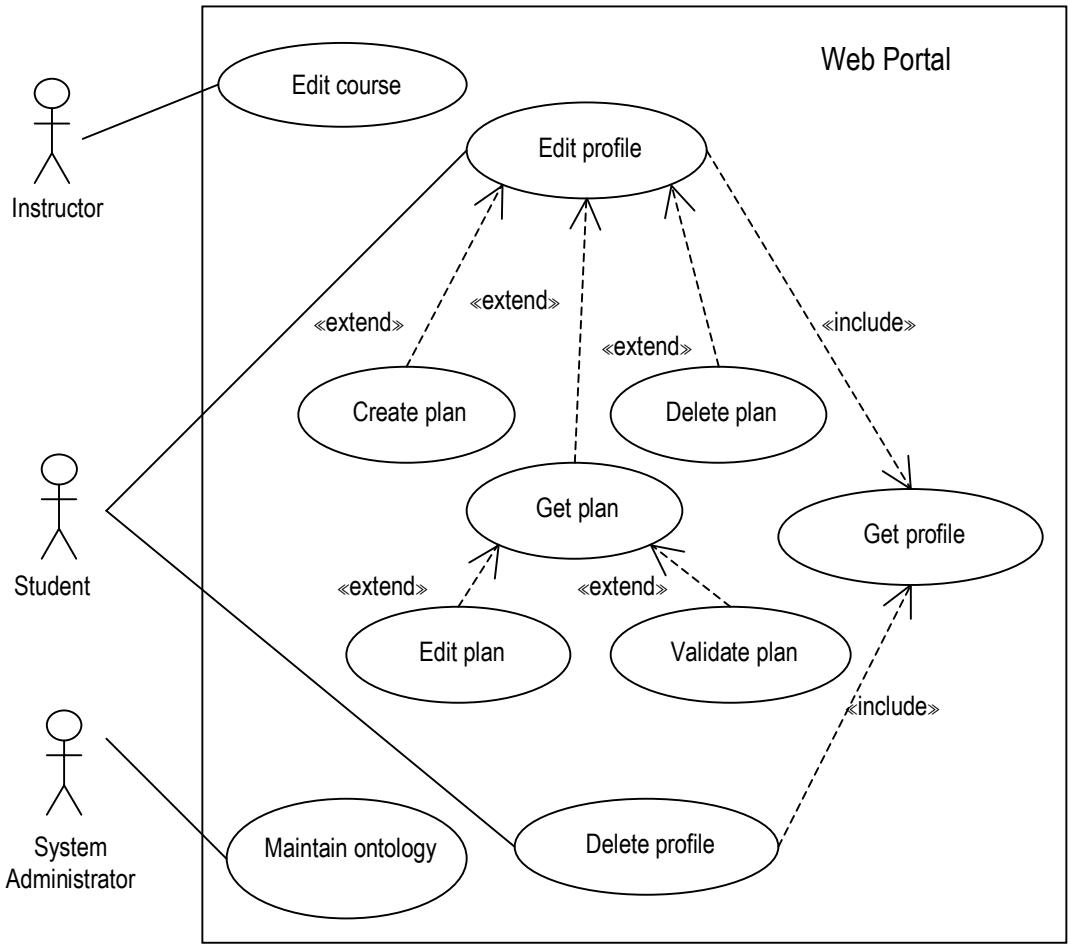


Figure 4-1: Use Case diagram

4.1.1 Actors

Instructor – a person who offers one or more courses at DTU, interested in making his course easy to find for interested students.

Student – a person following a study at DTU, interested in planning a semester.

System Administrator – a person who maintains the system.

4.1.2 Use Case descriptions

Name: Edit Course
Description: The instructor edits the information about which ACM topics are covered by a course, and classify the course by indicating its type.
Preconditions: The course exists in the DTU course catalogue.
Basic course of action:

1. The instructor gives the course number to the system.
2. The system shows the information about ACM topics related to the course if any.
3. The system shows the course type classification if any.
4. The instructor may change the information about ACM topics related to the course.
5. The instructor may change the course type.
6. The system saves the changes.

Name: Get Profile
Description: The system shows the student's profile.
Post conditions: A profile for the given student exists in the system.
Basic course of action:

1. The student gives the student number to the system.
2. The system shows the information about the student saved in the system.

Alternative course of action:

A. At step 2 of the original use case, if no profile for the student exists in the system, a new one is created.
B. The student must enter a type of study and the language of the student.
C. The new profile is saved.

Name: Edit Profile
Description: The student edits his profile information
Basic course of action:

1. Get Profile use case.
2. The student may add or remove course numbers of the courses he has already passed.
3. The student may add or remove ACM topics he is interested in.
4. The system saves the changes.

Name: Delete Profile
Description: The student removes his profile from the system.
Post conditions: No profile for the given student exists in the system.
Basic course of action:
1. Get Profile use case.
2. The system deletes the profile.

Name: Create Plan
Description: The student creates a new semester plan.
Preconditions: After step 1 of the Edit Profile use case.
Post conditions: A new semester plan exists in the student's profile.
Basic course of action:
1. The student chooses a season for the semester plan (fall or spring).
2. The student gives a unique name to identify the new plan.
3. The system creates a new empty semester plan for the given season.

Alternative course of action:
A1. At step 1 of the original use case, if the student does not provide a season, a default one is given (it does not matter which season is given as default).

Alternative course of action:
B1. At step 2 of the original use case, if the student does not provide a name to the plan, or if the name already exists, the system will show the corresponding error message.

Name: Delete Plan
Description: The student removes a plan from his profile.
Preconditions: After step 1 of the Edit Profile use case. The plan to be removed already exists in the student's profile.
Basic course of action:
1. The student selects a plan from his profile.
2. The system deletes the plan from his profile.

Name: Get Plan
Description: The system shows the selected semester plan.
Preconditions: After step 1 of the Edit Profile use case. The plan already exists in the student's profile.
Basic course of action:
1. The student selects a plan from his profile.
2. The system shows the contents of the chosen semester plan.

Name: Edit Plan
Description: The student edits the selected plan.
Preconditions: After step 2 of the Get Plan use case.
Basic course of action:
1. The student may add courses to the semester plan.
2. The student may remove courses to the semester plan.
3. The system saves the changes to the plan.

Alternative course of action:

A1. At step 2 of the original use case, if the course to be added does not exist in the Course Catalogue, or if the course does not have any schedules in the season corresponding to the semester plan, an error message is issued, and the course is not added to the plan.

Name: Validate Plan
Description: The system checks if the selected plan complies with DTU rules, and eventual selected restrictions.
Preconditions: After step 2 of the Get Plan use case.
Basic course of action:

1. The student may select that all courses in the plan must cover the student's topics of interest.
2. The student may select that all the prerequisites of the courses in the plan must be fulfilled by the student's passed courses.
3. The student asks the system to validate the semester plan.
4. The system answers if the semester plan complies with the given constraints or not. A list of courses not complying with the constraints, if any, is also given.

Name: Maintain ontology
Description: The system administrator wants to make some changes to the system's ontology, e.g. in case of DTU rules having been changed.
Basic course of action:

1. The system administrator makes the necessary changes directly in the file containing the ontology.
2. The system administrator restarts the system.

4.2 Supplementary Requirements

The following requirements comprise important information about the minimum quality of the system, concerning performance, security, usability, etc.

1. This first version of the system is to be a prototype only.
2. No security aspects need to be supported in this version of the system. This means that any user can see, edit and delete each other's profiles, semester plans and course information. This is not to be acceptable in the working system, but will suffice for prototyping purposes.
3. The system should be available through the Internet, and should only require a web browser to be accessed by the users. At least Internet Explorer v.6 or later must be supported in this prototype version.
4. Students must be able to plan their semester using the system without the need of assistance or a user's manual (i.e. the system must be self explanatory).
5. No user interface need to be provided for the system administrator in this version of the system. This of course requires that the system administrator is familiar with the implementation of the system. In a working system, a user friendly interface for this user should be considered.

4.3 Screen Mock-ups

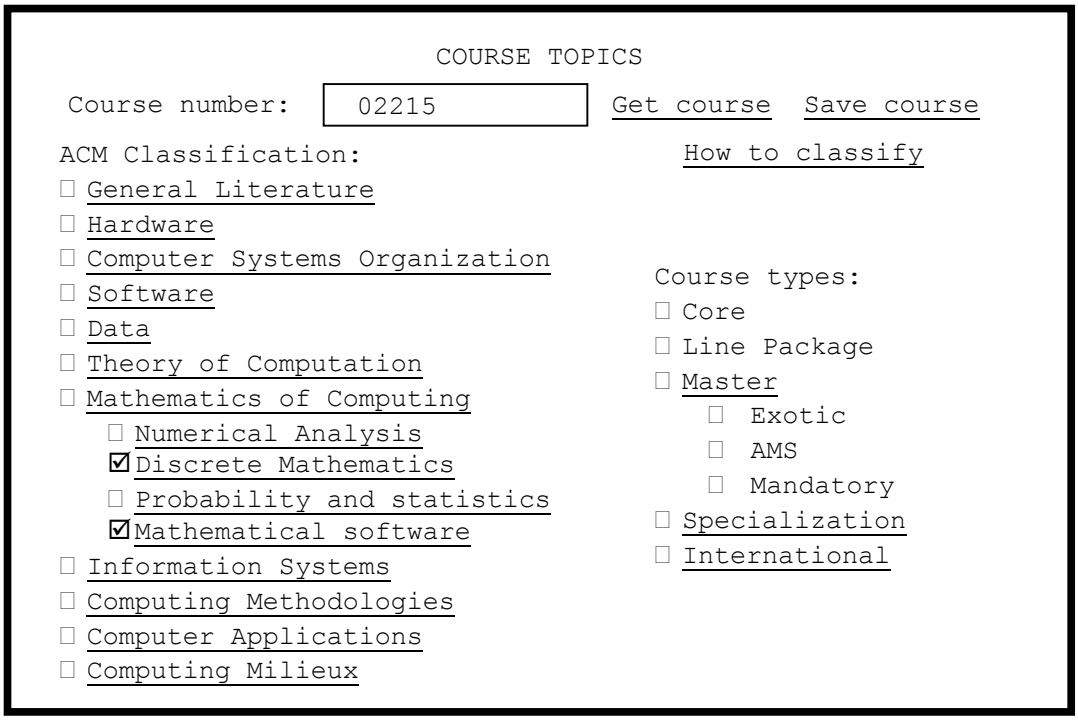
This section specifies the user-interface requirements. There are three interfaces of interest in this system. The first interface is the one used by the instructors who want to edit the topics and types related to their courses. The second one is the interface the student uses to handle his profile. The last one is the one the student uses to handle his semester plan.

The notation used in these mock-ups is the following:

Buttons or links, indicating an action activator are underlined text.
⊙ Choices where only one item can be selected are circles, and the selected circle has a dot in it.
☑ Choices where more than one item can be selected are squares, and the selected squares have marks in them.
Texts without any formatting are only information labels.
A text field for entering text is a box.

4.3.1 Course information interface

The following mock-up is a model for the interface used by instructors to classify their courses according to the ACM topics they cover.



The first button, the Get course button, is used to retrieve earlier saved information about a course, if any. The instructor must first give the course number and click on the Get

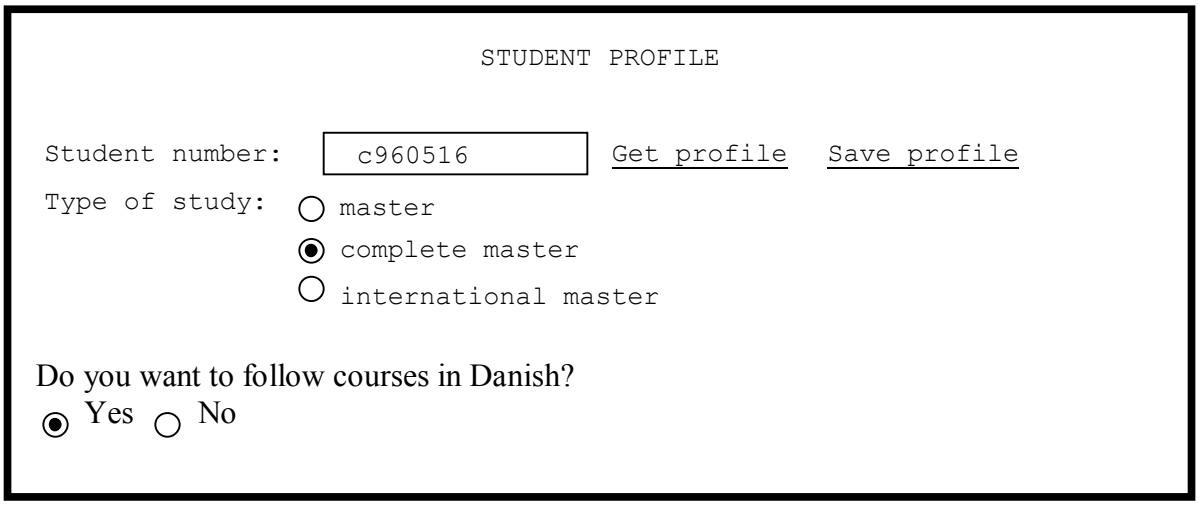
course button. If the given course has been classified before, its topics and types are then shown. If no course number is given an error message is issued.

The second button, the Save course button, saves the given classification for the given course. This action will overwrite any previous classification of the given course.
To classify a course, some topics of the ACM's Computing Classification System and types must be chosen before saving the course. Only the first level of ACM topics and types are initially shown. If the instructor clicks on an item its sub-items will appear. Items with no sub-items are not buttons.

The How to classify button links to a page explaining how to classify works using the ACM's Computing Classification System[9]. This guide is aimed to classify published papers, but can also be used to classify courses.

4.3.2 Student profile interface

The following mock-up is a model for the interface used by students to create and edit their profiles.

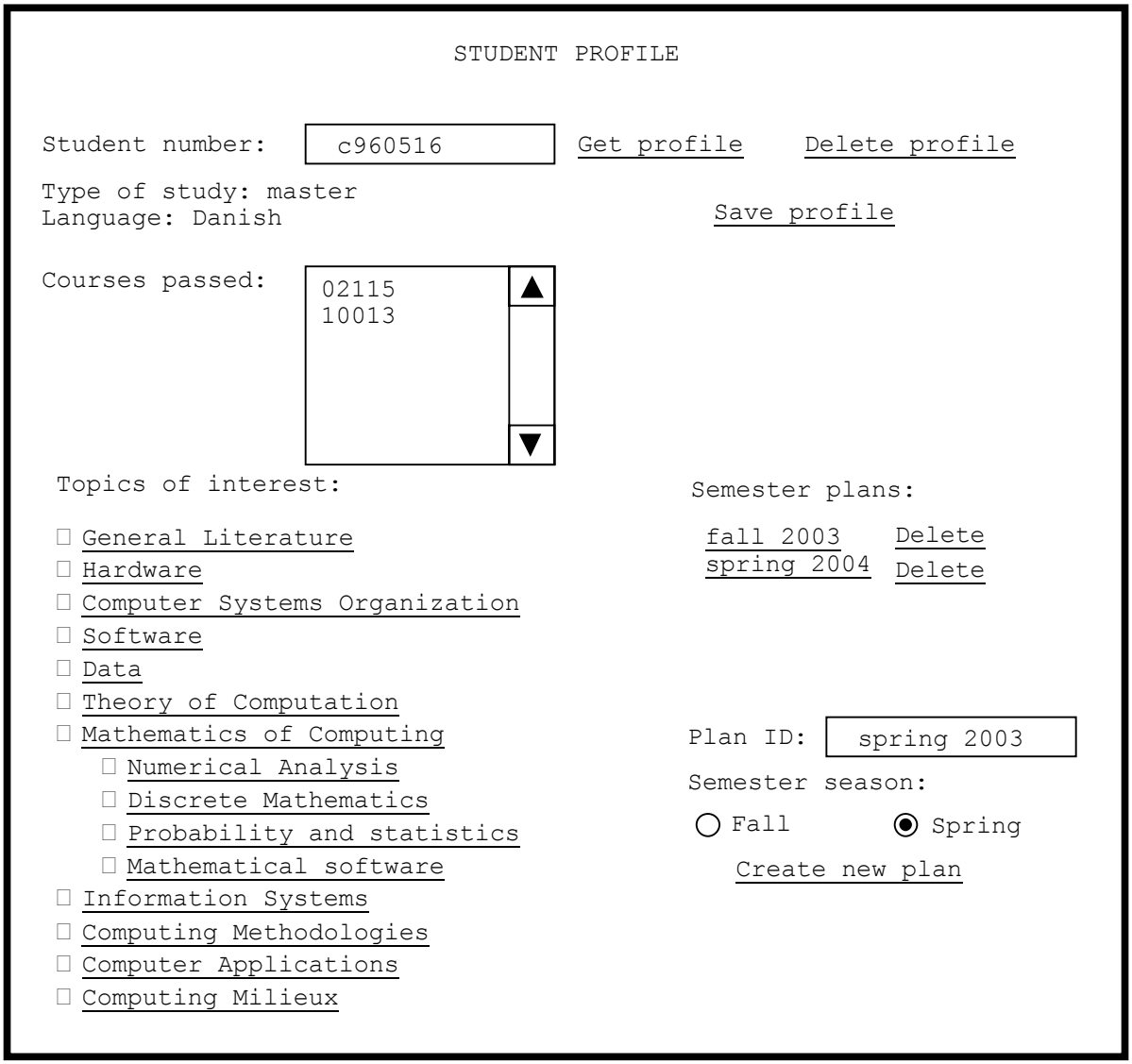


The Get profile button is used to retrieve earlier saved information about a student, if any. The student must first give his student number and click on the Get profile button.

If the student is using the system for the first time, or if he has deleted his profile, he must first create a new profile. The student's type of study and language must be entered, and the profile must be saved by clicking on the Save profile button.

If the student has created his profile before, his information will be shown. If no student number is given an error message is issued.

[9] http://www.acm.org/class/how_to_use.html



The Save profile button saves any changes made to the given profile. This action will overwrite any previous profile for the given student. The changes in the profile may be adding or removing courses from the passed courses[10] list, editing the topics of interest, creating or deleting semester plans.

The list of passed courses must contain one course number per line, as shown in the mock-up. The list can be edited by adding more lines to the list, or by removing any of the lines.

To indicate areas of interest, the student must choose some topics from the ACM's Computing Classification System. Only the first level of ACM topics is initially shown.

[10] The list of passed courses may include courses that are no longer in the Course Catalogue. Any courses containing letters in the course number (e.g. c4912) should be written without the letters (e.g. 4912).

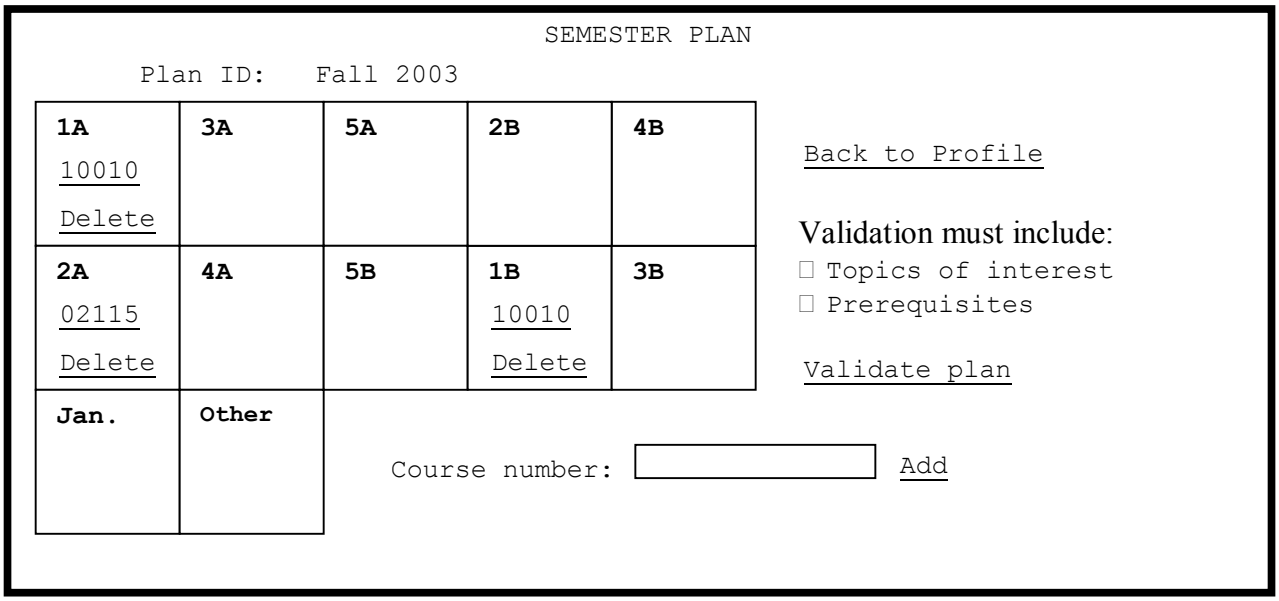
If the student clicks on a topic its sub-topics will appear. Topics with no sub-topics are not buttons. The student does not have to follow any rules when choosing his topics of interests. He can choose as many topics as he feels like.

The student profile interface is also used to create, remove and select semester plans. To create a plan a unique plan ID and a season must be given. To remove a plan the selected plan's delete button must be clicked. To select a plan the plan ID button must be clicked, and the student will be directed to the semester planning interface.

The Delete profile button removes the given profile from the system. This will also remove any semester plans included in the profile.

4.3.3 Semester planning interface

The following mock-up is a model for the interface used by students to plan a given semester.



This interface is accessed from the student profile interface by choosing one of the plans previously created by the student. This interface contains a semester schedule.

The student may add a course to the plan by giving the course number and clicking on the Add button. If the course does not exist in the Course Catalogue, or if it is not held in the season corresponding to the plan, an error message will be shown, and the course will not be added to the plan.

To delete a course from the plan, the Delete button next to the course number must be clicked.

To verify if the semester plan complies with the system constraints (DTU rules), the Validate plan button should be clicked.

Notice that the information given in the student's profile will be taken into account when validating the semester plan (e.g. if the student does not speak Danish, no courses in Danish will be allowed in the semester plan). The student can also ask the system to validate topics covered by the courses (according to the student's topics of interest) and course prerequisites (according to the student's passed courses).

If the student wants to see more information about a course, he just has to click on the `number` of the corresponding course in the schedule, and the course page in the course catalogue will be shown.

When the student is finished working on his schedule he must click on the Back to Profile button. This will return the student to the profile interface.

5 SYSTEM DESIGN

The previous chapter described what the system must be able to solve. This chapter describes how the system must do it.

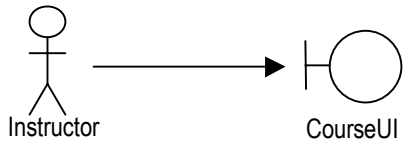
Due to the low level of complexity of the system, there is no need to separate the analysis phase from the design phase of this project. This chapter thus contains an analysis model consisting of collaboration diagrams realizing the use cases described in the previous chapter, and the corresponding analysis classes. Besides this chapter contains a design model consisting of the deployment diagram, the component diagram and the class diagram.

This chapter also contains details on how the system database, or more specifically the knowledge base, is to be built. At the end of the chapter a discussion about the tools to be used in the development of the system is presented.

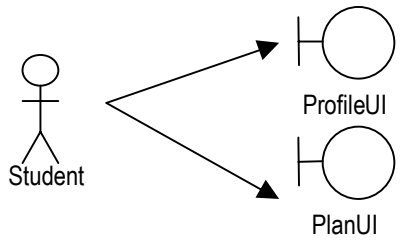
5.1 Analysis classes

This section recognizes the classes necessary to achieve the requirements specified in the previous chapter. The analysis classes are divided into Boundary classes, Entity classes and Control classes.

5.1.1 Boundary Classes

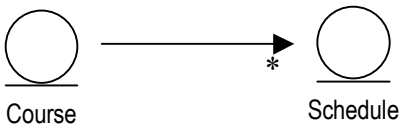


The CourseUI class supports the interaction between the Instructor actor and the Edit Course use case. This class allows an instructor to classify courses according to their type and the topics they cover.

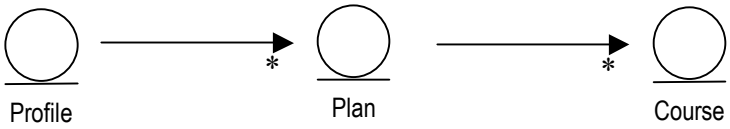


The ProfileUI and PlanUI classes support the interaction between the Student actor and the Edit Profile and Delete Profile use cases. The ProfileUI class allows a student to create a new profile, edit a profile, delete a profile, create a semester plan and delete a semester plan. The PlanUI class allows a student to edit and validate a semester plan.

5.1.2 Entity Classes



The Course class is used to represent DTU courses. The Course class contains information about course classification given by the Instructor, as well as schedule information derived from the DTU Course Catalogue. The Schedule class holds information about when a course is taught.



The Profile class is used to represent a student's profile. All information provided by the student is contained in this class. The Profile class may contain zero, one or more semester plans, which are represented by their corresponding Plan class. The Plan class contains information about the semester season and the courses contained in the plan.

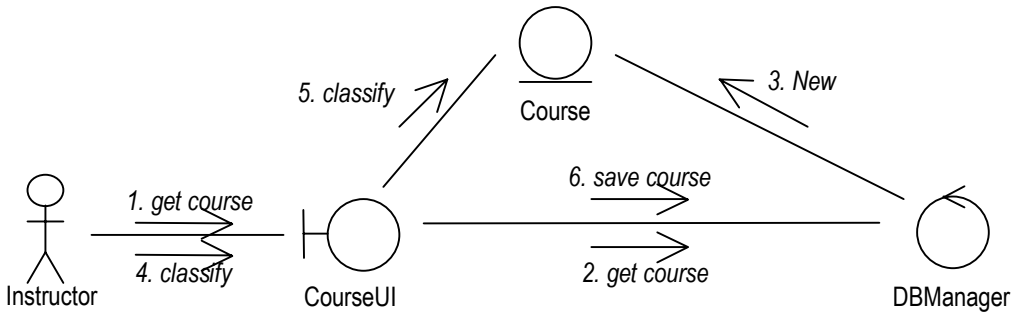
5.1.3 Control Classes

The DBManager class is the only control class in the system, and is used to coordinate all transactions among the boundary classes and the Knowledge Base. This class is also responsible for the Validate Plan use case.

5.2 Collaboration diagrams

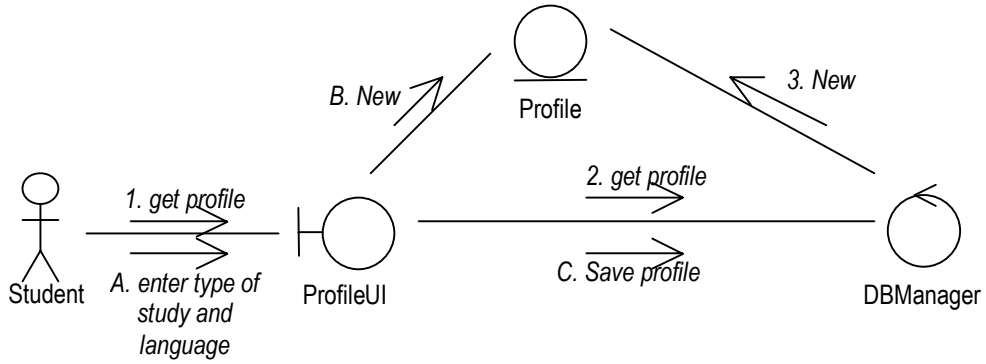
The following collaboration diagrams realize each of the use cases specified in the system requirements.

5.2.1 Realization of the Edit Course use case



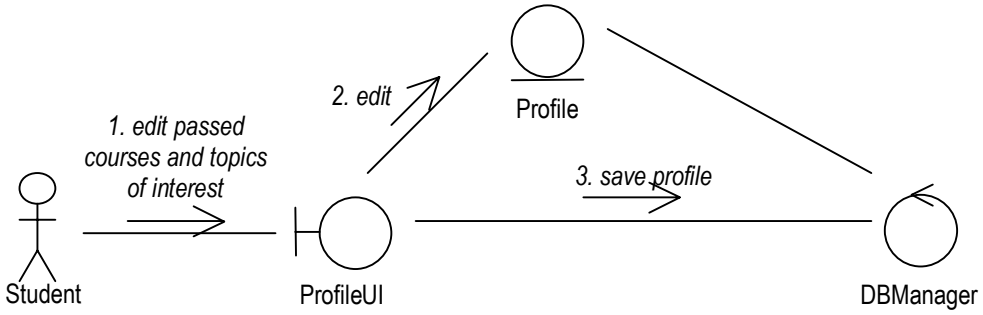
The instructor indicates which course he wants to edit (1). The DBManager retrieves the course from the knowledge base and creates the corresponding Course instance (2, 3). The instructor then classifies the course with respect to its type and its covered topics (4, 5). Finally the newly classified course is saved (6).

5.2.2 Realization of the Get Profile use case



The student requests for his profile (1). The DBManager retrieves the profile from the knowledge base and creates the corresponding instance (2, 3). If the profile does not exist in the knowledge base, instead of step (3) the student must enter his type of study and language (A). A new profile instance for the student is created (B) and saved in the knowledge base by the DBManager (C).

5.2.3 Realization of the Edit Profile use case

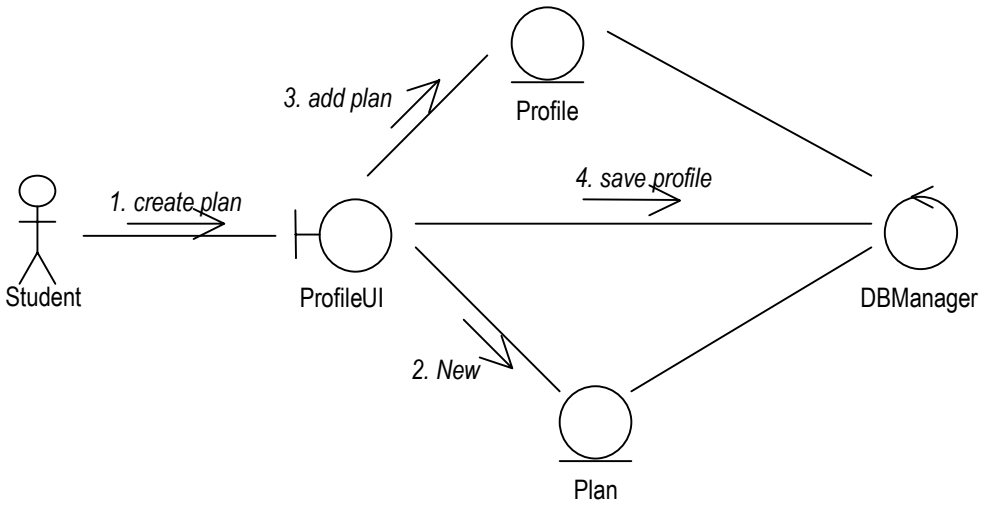


After the execution of the Get Profile use case, the student may add or remove passed courses and topics of interest (1, 2). The modified profile is saved in the knowledge base by the DBManager (3).

5.2.4 Realization of the Delete Profile use case

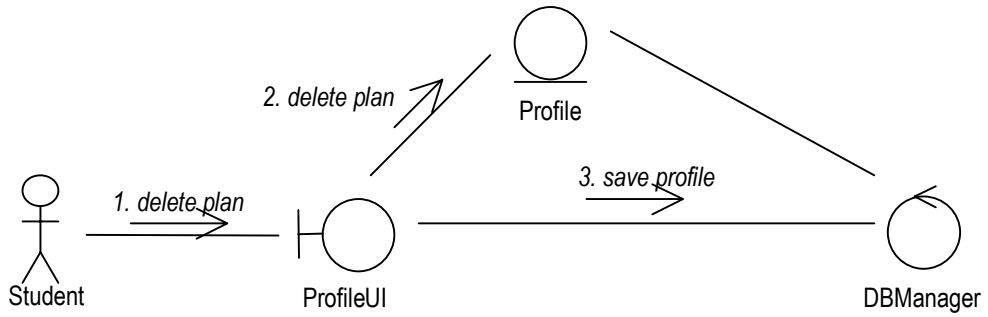
After the execution of the Get Profile use case, the student requests the deletion of the profile (1). The profile is removed from the knowledge base by the DBManager (2).

5.2.5 Realization of the Create Plan use case



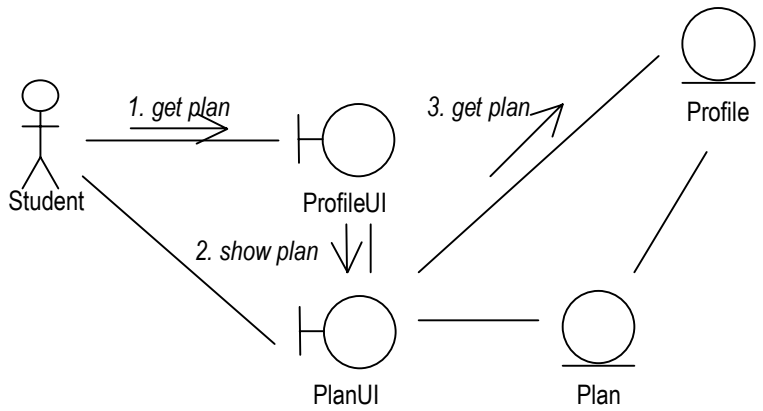
T
he student requests the creation of a new semester plan by entering a season and a plan ID (1, 2). The plan is added to the student's profile (3) and the profile is saved in the knowledge base by the DBManager (4).

5.2.6 Realization of the Delete Plan use case



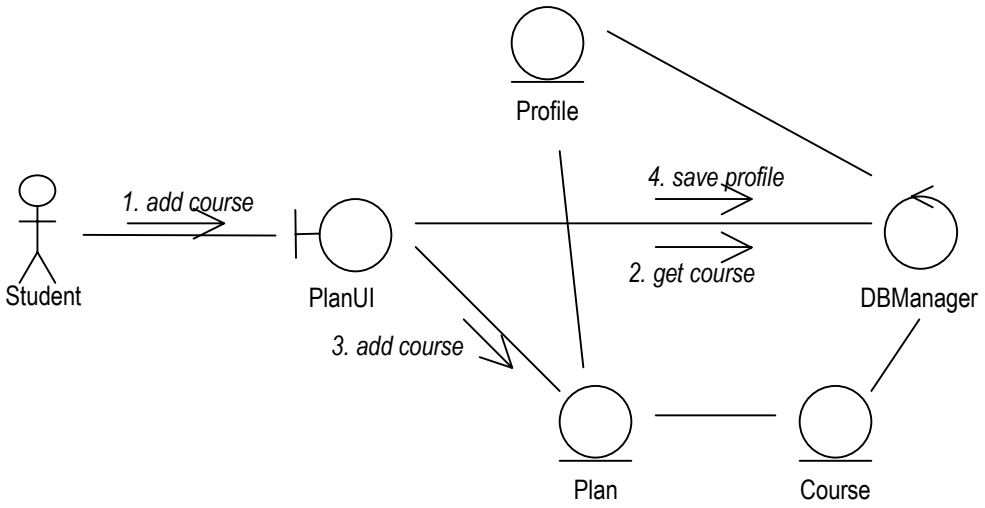
The student requests the deletion of a given plan in his profile (1). The plan is removed from the student's profile (2) and the profile is saved in the knowledge base by the DBManager (3).

5.2.7 Realization of the Get Plan use case

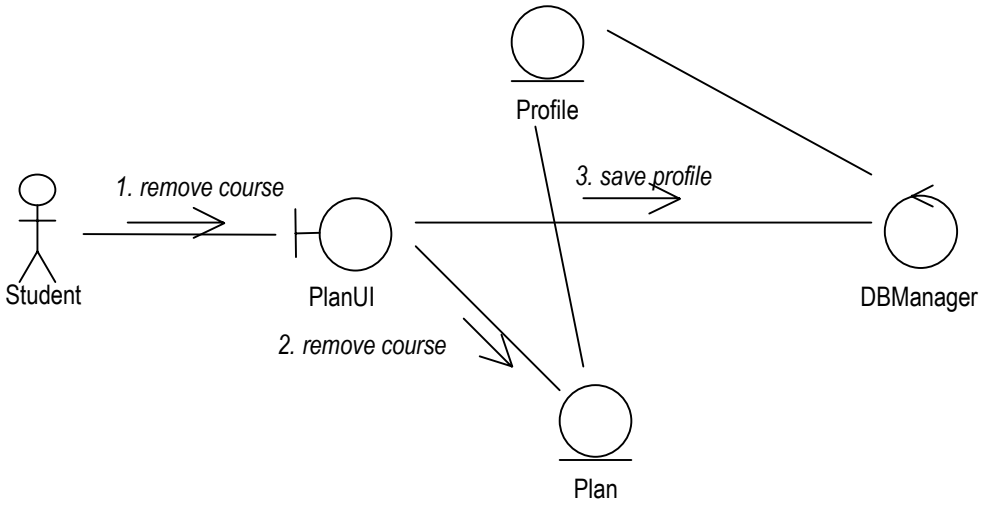


The student requests to see the given semester plan. The PlanUI class retrieves the plan from the profile instance and shows it to the student.

5.2.8 Realization of the Edit Plan use case

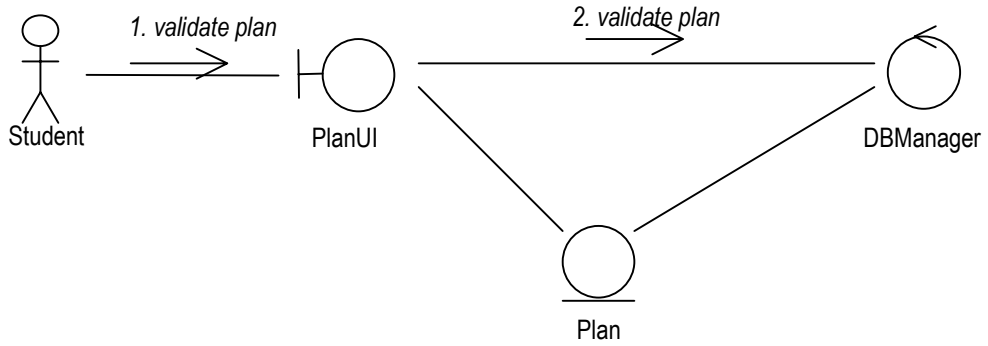


The student may add a course to the semester plan (1). The course is retrieved from the knowledge base by the DBManager (2) and added to the plan (3). The profile that contains the modified plan is saved in the knowledge base by the DBManager (4).



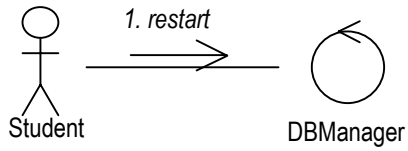
The student may remove a course from the semester plan (1, 2). The profile that contains the modified plan is saved in the knowledge base by the DBManager (3).

5.2.9 Realization of the Validate Plan use case



The student may request for the plan shown to be validated, eventually indicating if topics of interest and prerequisites must be included in the validation (1). The DBManager validates the plan (2); if any courses do not comply with the system ontology, they are listed to the student as not complying courses.

5.2.10 Realization of the Maintain Ontology use case

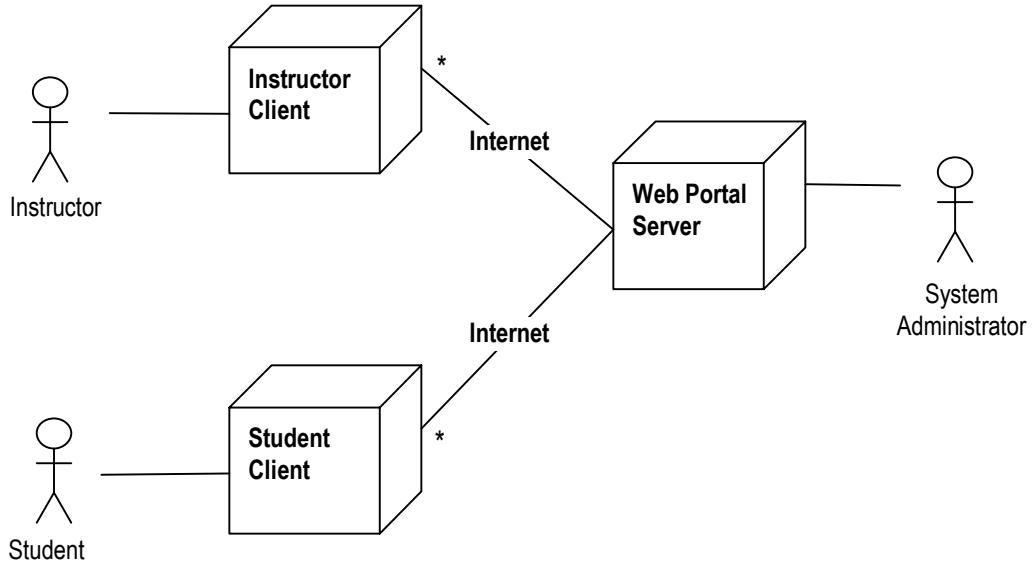


The system administrator changes a given ontology by accessing the ontology file and editing it. No interface is provided for this purpose. The DBManager must be restarted after the ontology has been modified.

When the ontology file is modified, its contents must be checked for syntax errors and validated (i.e. checked for consistency). If there are consistency errors, the system administrator must check the new ontology for mistakes. If no mistakes are found, the proper DTU instances should be contacted, so that DTU rules can be checked for consistency.

The ACM Computing Classification System is assumed not to change very often. If the changes are only the addition of new topics in the tree, and not removal, they will have very little importance in the current system being developed. On the other hand, if topics are removed, all data concerning course topics and students' interests will have to be revised. This latter case is assumed here never to occur.

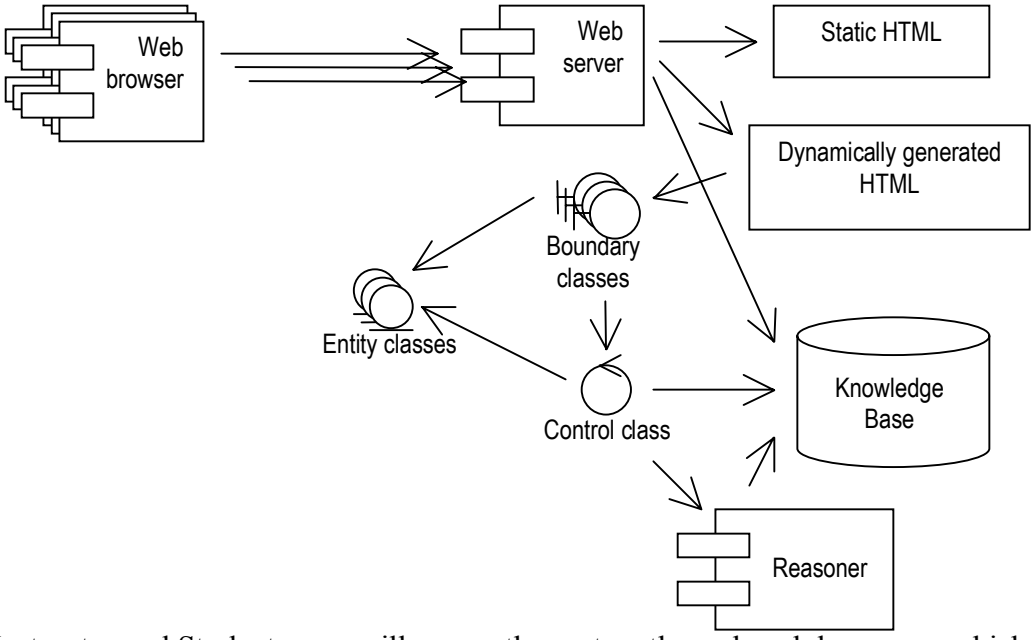
5.3 Deployment diagram



The Web Portal will execute in one server node and a number of client nodes. Instructors will access the portal through Instructor Clients, while students will access it through Student Clients. The System Administrator accesses the system directly through the Web Portal Server.

The clients must be provided with web browsers, and the server will be provided with a web server. The clients communicate with the server using the Internet, and will only receive HTML pages and send HTTP requests to the server. The HTML pages sent to the clients must be dynamically generated at the request time.

5.4 Component diagram



The Instructor and Student users will access the system through web browsers, which will communicate with our Web server via the Internet. The Web server has access to three kinds of components: Static HTML pages, dynamically generated HTML pages and the files in the knowledge base.

The static HTML pages and the files in the knowledge base are sent as they are to the Web browser requesting them. The dynamically generated HTML pages are created at request time, according to the contents of the knowledge base at that time.

The dynamically generated HTML pages communicate only with the boundary classes to interact with the knowledge base. Only through them is it possible to change the contents of the knowledge base (or by editing the files directly on the server, which is only allowed for the system administrator).

Furthermore, the control class communicates with a reasoner that has as purpose to validate semester plans according to the system ontology. This reasoner has access to the knowledge base.

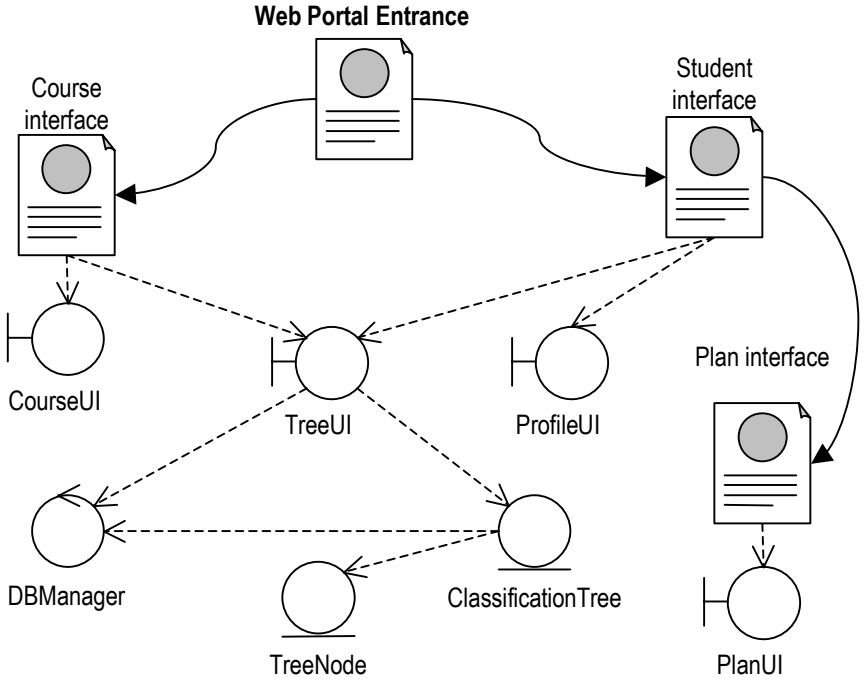
5.4.1 Security Issues

Even though security is not required in the prototype version of this system, some issues are worthwhile mentioning.

Security in the system should be enforced at the Web Server level, through SSL communication. Access to different components should then require authentication. Static HTML pages and the part of the knowledge base corresponding to the ontology and courses should be visible for all. The part of the knowledge base corresponding to student profiles should only be visible by the corresponding profile owners. The dynamically

generated HTML pages allow users to create, remove and delete contents of the knowledge base. The dynamically generated HTML pages that correspond to course classification should only be accessed by instructors, and all instructors should be able to modify any course. The pages that correspond to profiles and plans should only be accessed by the owner of the corresponding profile.

5.5 The Interface Components



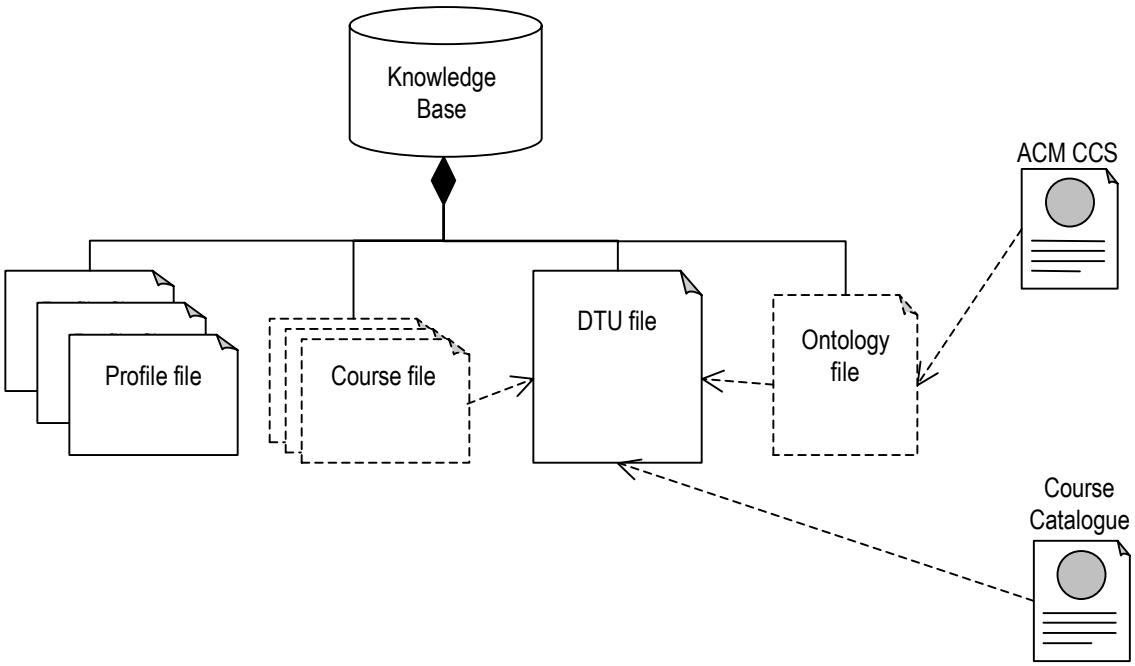
The Interface of the Web portal consists of static HTML pages and dynamically generated HTML pages. The entrance of the Web portal is a static HTML page that links to the course classification interface and the student profile interface. The student interface links to the plan interface. The CourseUI class generates the dynamic part of the course interface, the ProfileUI class the dynamic part of the student interface and the PlanUI class the dynamic part of the plan interface.

Both the course interface and the student interface must also contain information about classifications. For example, the type of the course is a course classification, the topics covered by the course too, as well as the type of study and the topics of interest. These classifications have as purpose to help the instructor or student enter information about a course or profile. All these classifications are contained in the ontology of the system, in the knowledge base. These classifications can be visualized as trees, where each tree node is a concept class[11] together with a human readable label for the class.

[11] There is a very important difference between a concept class and a design class. Concept classes are the classes forming a classification, e.g. Course class, AMSCourse class, CoreCourse class, etc. Design classes, which also correspond to analysis and implementation classes, are the boundary, entity and control classes of the system, e.g. DBManager class, ProfileUI class, etc.

The TreeUI class is a boundary class that generates a classification tree to be displayed in the corresponding HTML page. This boundary class makes use of a ClassificationTree entity class to hold the information about the classification. The classes forming the classification are accessed through the DBManager control class. Each classification class becomes an instance of the TreeNode entity class.

5.6 The Knowledge Base



The knowledge base design is mostly determined by the ER-diagrams, Hasse diagrams and Description Logics model shown in the Domain Analysis. What should be considered in this section is exactly where the content of the knowledge base comes from and how it is maintained and used by the system.

Every time the system is restarted, the DTU file containing information about DTU rules (ontology) and courses must be generated, so that it reflects any changes to these information sources.

The system will handle 3 kinds of data. The first kind will be the data given to the system, through a graphical interface (Profile files and Course files). The Course files are not part of the knowledge base directly, but contain additional information about courses, stored in XML format. This information will be inserted in the DTU file every time the file is generated. This kind of data is stored permanently in the knowledge base until the users modify or delete them.

The second type is the Ontology file (the metadata, the T-box). This ontology is also stored permanently in the database, and is given by the system administrator directly into system

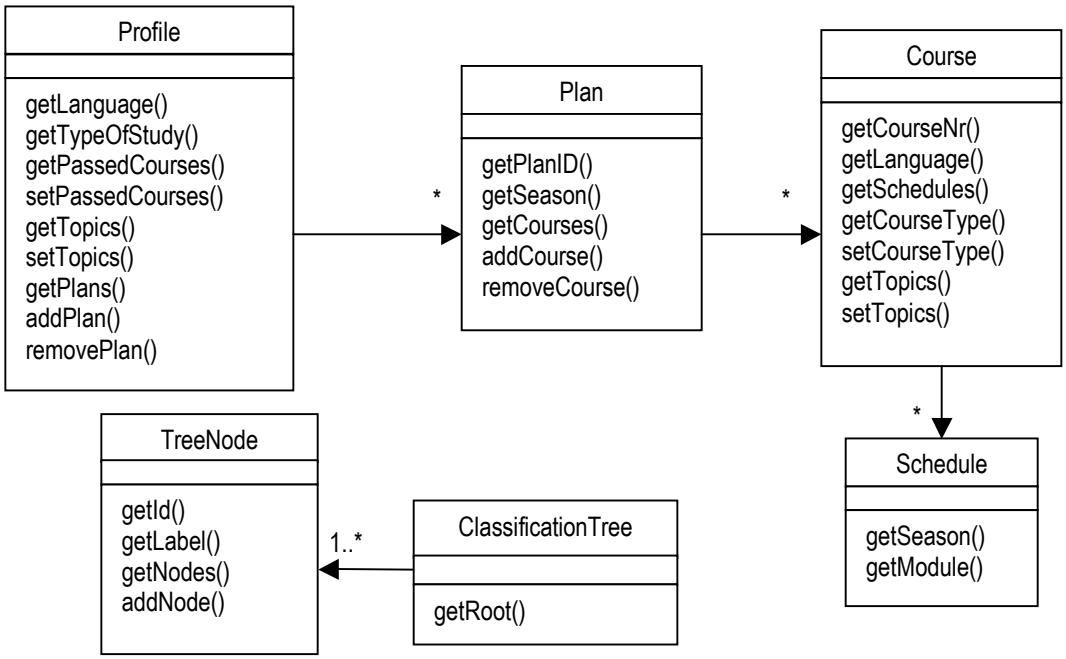
files. The difference here is that there is no graphical interface for data input and the user itself is responsible for the correctness and consistency of the input files.

The last kind of data is all the data that is extracted from other sources. This is the data about courses, which is found in the Course Catalogue, and the data about ACM CSS topics. This data must be updated often, as changes in the course catalogue happen frequently. ACM topics are not changed so often though. The data contained in the course catalogue must be periodically extracted and transformed to match the schema given by the model in the Domain Analysis. The course part of the knowledge base is in this way a Data Warehouse, and the periodicity of the data update must depend on how often the course catalogue changes and the time needed for each update.

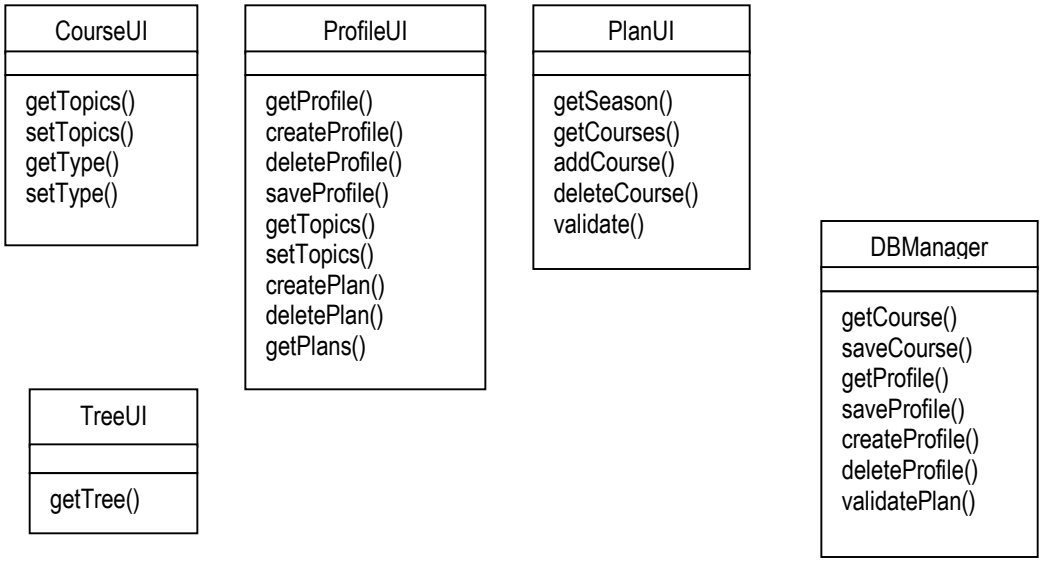
The graphical interface must make sure that no inconsistent data is inserted in the knowledge base. Any changes in the ontology must be checked to maintain consistency with existing data.

5.7 Class diagram

From the above sections it is possible to extract the final class diagram. The necessary methods for each class were recognized from the collaboration diagrams. No attributes are given here, because all attributes should be accessed through the corresponding get and set methods. Only public methods are shown.



The first part of the class diagram represents the entity classes of the system. These classes repeat some of the information contained in the knowledge base. This is necessary because the interface is not dynamically generated according to the knowledge base content.



The second part of the class diagram represents the boundary and control classes of the system.

5.8 Standards and Tools

In the following the standards and tools chosen for the development of the system will be mentioned. More details about how those standards and tools are used in this project can be found in the implementation chapter of this document.

5.8.1 Creating the knowledge base

The choice of standards is practically defined by the project title itself: Semantic Web Technologies. This means that all of the system data (i.e. the knowledge base) will be stored in DAML+OIL. The current version of DAML+OIL, which was the one chosen for this system, is the version of March 2001.

5.8.2 Storing the knowledge base

The knowledge base will not contain a large amount of data, at least not in the prototype phase of the system. Therefore there is no need for an efficient database management system, why the knowledge base will be directly implemented as text files. These files can be edited with any text editor available.

5.8.3 The contents of the knowledge base (the A-box)

Part of the knowledge base content will come from other web resources, these being the courses from the DTU course catalogue and the topics from the ACM Classification. These resources are accessed through HTML pages. To automate the extraction of the necessary information from these resources, Compaq's Web Language (WebL) has been chosen. WebL is a scripting language for processing documents in the web that has built-

in knowledge of web protocols and markup documents. This tool is also known as a "screen scraper".

5.8.4 Managing the knowledge base

To build the control engine of the system the Jena Toolkit, from HP Labs Semantic Web Research, has been chosen. This toolkit contains parsers that support RDF/XML and DAML+OIL, and a query language (RDQL) for RDF documents. It also provides an extensive application program interface (API) supporting general manipulation of RDF and DAML+OIL documents, including inference. The Jena Toolkit is completely written in Java[12], which is also the programming language chosen for developing the control engine.

5.8.5 The user interface

The graphical user interface will be developed using Java Server Pages (JSP), and therefore a JSP enabled web-server must be used, as for example the Apache Tomcat web-server. The user will interact with the system through a web-browser. The web-browser that will be used to test this prototype version of the system will be the MS Internet Explorer version 6.0.

5.8.6 The knowledge base validation

Validation of the knowledge base is the key functionality of the Semantic Web Portal. This is the part of the system that actually checks if the semantics (T-box) of the knowledge base are being complied by its instances (A-box), i.e. if the semester plan created by a student complies with DTU rules.

Validation of the knowledge base is necessary in two situations: when creating the knowledge base (syntax and T-box coherence check), and when adding new instances to the knowledge base (A-box consistency check). As the knowledge base is implemented in DAML+OIL, two tools were considered to validate it. The first one was DAML Validator, which is provided and recommended by the DARPA Agent Markup Language Program[13], and the second one was RACER, a description logics reasoner provided by Volker Haarslev and Ralf Möller that started to support DAML since December 2002.

DAML Validator is much more user friendly than RACER, and is very useful when creating the knowledge base. That is because RACER does not provide helpful error messages, making it almost impossible to find what is causing a syntax error. DAML Validator provides explanatory error messages containing even the position of the syntax errors.

The creation of the DAML files could also be done using OILED, which also supports DAML+OIL. Oiled is a graphical interface for creating a knowledge base correctly and exporting it to DAML. On the other hand, DAML files created by OILED are not easily read by humans. I preferred to create the files manually using a common text editor and

[12] The version of Java used in this project is 1.4.
[13] DAML Program – http://www.daml.org

validating them using the DAML Validator, so that I would get a better understanding of the language itself.

When checking the A-box for consistency I found out that DAML Validator was not able to check for certain inconsistencies, and that RACER was. Some inconsistencies are not able to be checked by any of the tools, as in the case of insufficient information about the used concepts. RACER uses Open World Assumption, i.e. what cannot be proven is not considered to be false. This means that when concepts are not sufficiently defined, and the consistency cannot be proven to be right, it simply answers that consistency cannot be proven. DAML Validator uses the Closed World Assumption, i.e. that when concepts are not sufficiently defined it may provide wrong answers about the A-box consistency.

A concept is sufficiently defined in DAML when sufficient conditions are stated in order for concept membership to be established. This is achieved by using enumeration of concept instances, or by stating that a concept is equal to other concept sufficiently defined, or by Boolean combinations of these.

A test was made to check the validation abilities of both tools. In the test, very simple models were used. See the test description and results in the Test chapter of this document. After the test, the choice of A-box validation tools was RACER version 1.7.6, and the choice of syntax checker was DAML Validator version 2003.03.18. Be aware that the DAML Validator is still under development and improvements are made regularly. The chosen version of DAML Validator supports the `rdf:parseType="daml:collection"`[14], which was not supported by previous versions. RACER also supports this feature.

For accessing the services of RACER using methods a Java API called JRACER can be used.

[14] DAML+OIL needs to represent unordered collections of items in a number of constructions, such as intersectionOf, unionOf, oneOf, disjointUnionOf and Disjoint. DAML+OIL exploits the rdf:parseType attribute to extend the syntax for RDF with a convenient notation for such collections. Whenever an element has the rdf:parseType attribute with value "daml:collection", the enclosed elements must be interpreted as elements in a list structure.

6 IMPLEMENTATION

This chapter concentrates on the implementation of the knowledge base, which is the core of the Semantic Web content of the system. Every step of the implementation of the knowledge base will be described in detail. The rest of the implementation will not be so detailed, as it is very similar to other system development projects, and not so interesting in the context of this project. Nevertheless, special considerations to the system implementation will be depicted as thoroughly as necessary. Furthermore, the installation of the tools to be used is described here.

6.1 Implementing the knowledge base

The implementation of the knowledge base consists of the conversion of the description logics model from the domain analysis into a DAML+OIL file, and the insertion of the instances that are not related to course classification or student profiles (these are to be inserted via the graphical interface).

The contents of the file must be wrapped inside the following tags, which indicate that this is an XML file, that the namespaces rdf, rdfs and daml are used in the file, that the file is placed at `http://localhost/DAML_FILES/T-box/dtu.daml` and that this file contains an ontology that imports `daml+oil`:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns     ="http://localhost/DAML_files/T-box/dtu.daml#"
>

  <daml:Ontology rdf:about="">
   <rdfs:comment>
      An ontology about DTU course planning. This plan only takes into
      account course classification, languages and types of study.
   </rdfs:comment>
   <daml:imports rdf:resource="http://www.daml.org/2001/03/daml+oil"/>
  </daml:Ontology>

  <!-- The rest of the file comes here ... -->

</rdf:RDF>
```

The conversion of the part of the description logics model corresponding to the ER-diagrams is very straightforward. All the relations in the ER-diagram become DAML properties with specific range and domain entities. Here is an example of the speaks property:

```
<daml:ObjectProperty rdf:ID="speaks">
  <rdfs:label>speaks</rdfs:label>
  <rdfs:comment>The language the student speaks.</rdfs:comment>
  <rdfs:domain rdf:resource="#Student"/>
  <rdfs:range rdf:resource="#Language"/>
</daml:ObjectProperty>
```

The label and comment tags are only included to add human-readability. All other properties in the ER-diagrams are expressed in this way. Properties that are unambiguous, i.e. they can only appear in a relation once, have the UnambiguousProperty type. Here is an example of an unambiguous property, where a semester plan belongs to only one student:

```
<daml:ObjectProperty rdf:ID="plans">
  <rdfs:domain rdf:resource="#Student"/>
  <rdfs:range rdf:resource="#Semester"/>
  <rdf:type rdf:resource=
          "http://www.daml.org/2001/03/daml+oil#UnambiguousProperty"/>
</daml:ObjectProperty>
```

The only attribute in the ER-diagram is the course name attribute, which is a string:

```
<daml:DatatypeProperty rdf:ID="courseName">
  <rdfs:label>Course name</rdfs:label>
  <rdfs:domain rdf:resource="#AnyCourse"/>
  <rdfs:range
           rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</daml:DatatypeProperty>
```

The entities in the ER-diagrams become concept classes in the DAML file:

```
<daml:Class rdf:ID="Language">
  <rdfs:label>Language</rdfs:label>
</daml:Class>
```

The cardinality constraints expressed in the ER-diagrams are also expressed in the corresponding concept class:

```
<daml:Class rdf:ID="PrerequisiteGroup">
  <rdfs:subClassOf>
    <daml:Restriction daml:minCardinality="1">
      <daml:onProperty rdf:resource="#atLeastOneCourse"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
```

The whole ER-model can be expressed using the above mentioned DAML+OIL constructs.

The Hasse-diagrams are easier to implement in DAML+OIL than the ER-diagrams. In this part of the model we simply express which concept classes are subclasses of other concept classes:

```
<daml:Class rdf:ID="English">
  <rdfs:label>English</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Language"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <English rdf:ID="EN">
      <rdfs:label>English</rdfs:label>
    </English>
  </daml:oneOf>
</daml:Class>
```

Because of the limitations of the validation tools used, all classes must be sufficiently defined, i.e. defined using the oneOf construct as seen in the example above. This is very easy to achieve with classes containing only one instance, as the one above. Other classes, as for example the Course class, are more complicated.

Courses are classified while the system is being used, by the instructors. This means that one concept class that is defined with a set of courses (e.g. Core courses) may suddenly have one more instance included in its enumeration (oneOf construct), or maybe even removed from it. This implies that the concept class definition must be changed during the use of the system.

The final DAML file containing the domain model is dynamically created when the system is started, so that it reflects any changes made to the Course Catalogue at start time. But the requirement of sufficiently defined classes also requires that the DAML file be re-created every time a course is classified.

Before describing how the final version of the DAML file, containing all courses, is dynamically created, the rest of the static part of the knowledge base must be implemented. Let's continue with the rest of the Hasse-diagrams.

One of the largest classifications used in this system is the ACM topic classification. Even though this classification could be implemented manually into DAML, this would be a fastidious job because of its extent. A WebL script has been implemented in this project in order to read the HTML page containing the ACM topic classification and creating the corresponding part of the DAML file we need.

The WebL script, called topic_reader.webl in this project, has been implemented by identifying the template of the HTML page containing the ACM classification, and the information that should be extracted from it, then elaborating on how this information should be transformed into DAML.

Here is an example of a part of the HTML page containing the ACM classification:

```
<ul>
<li><a name = "A">A.</a> General Literature
  <ul>
  <li><a name = "A.0">A.0</a> GENERAL
    <ul>
    <li><em>Biographies/autobiographies</em>
    <li><em>Conference proceedings</em>
    <li><em>General literary works (e.g., fiction, plays)</em>
    </ul>
  <li><a name = "A.1">A.1</a> INTRODUCTORY AND SURVEY
  <li><a name = "A.2">A.2</a> REFERENCE (e.g., dictionaries,
                              encyclopedias, glossaries)
  <li><a name = "A.m">A.m</a> MISCELLANEOUS
</ul>
```

The topics in the ACM classification can be found in a hierarchy of HTML lists, where most topics have both an id and a name (label), e.g. A.0 and General, and some have only a name (label), e.g. Conference proceedings. The latter are always at the bottom of the topic hierarchy.

The names (labels) of concept are not unique, but the IDs are. As we need unique names to identify the ACM topic classes we are going to create, the IDs will be used for this purpose. This means that topics that do not have an ID must get one from our WebL script at generation time, that will always be the same and unique for each of them. The script simply takes the ID of the parent of the topics and adds an extra numeration to them. This would cause the topics without IDs in the example above to receive the following ids:

```
<a>A.0.1</a> Biographies/autobiographies
<a>A.0.2</a> Conference proceedings
<a>A.0.3</a> General literary works (e.g., fiction, plays)
```

The DAML content corresponding to the example would then be like beneath. Notice that here the concept classes are also sufficiently defined, with exactly one instance per topic:

```
 <daml:Class rdf:ID="A">
   <rdfs:label>General Literature</rdfs:label>
   <rdfs:subClassOf rdf:resource="#Topic"/>
   <daml:unionOf rdf:parseType="daml:collection">
     <daml:Class>
       <daml:oneOf rdf:parseType="daml:collection">
         <A rdf:ID="topic-A"/>
       </daml:oneOf>
     </daml:Class>
     <daml:Class rdf:about="#A.0"/>
     <daml:Class rdf:about="#A.1"/>
     <daml:Class rdf:about="#A.2"/>
     <daml:Class rdf:about="#A.m"/>
   </daml:unionOf>
 </daml:Class>
```

```
<daml:Class rdf:ID="A.0">
  <rdfs:label>GENERAL</rdfs:label>
  <rdfs:subClassOf rdf:resource="#A"/>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class>
      <daml:oneOf rdf:parseType="daml:collection">
        <A.0 rdf:ID="topic-A.0"/>
      </daml:oneOf>
    </daml:Class>
    <daml:Class rdf:about="#A.0.1"/>
    <daml:Class rdf:about="#A.0.2"/>
    <daml:Class rdf:about="#A.0.3"/>
  </daml:unionOf>
</daml:Class>

<daml:Class rdf:ID="A.0.1">
  <rdfs:label>Biographies/autobiographies</rdfs:label>
  <rdfs:subClassOf rdf:resource="#A.0"/>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class>
      <daml:oneOf rdf:parseType="daml:collection">
        <A.0.1 rdf:ID="topic-A.0.1"/>
      </daml:oneOf>
    </daml:Class>
  </daml:unionOf>
</daml:Class>

...
```

After generating this part of the DAML file, it can be included manually to the static part of the DAML file.

The last static part of the DAML file is the one corresponding to the constraints expressed in the description logics model. Most of the constraints were not implemented, because they contain existential quantifications (constraints 5 to 11), which can not be handled by RACER, the tool for validating A-boxes (see 7.1 Test of A-box Validation Tools). If they were implemented, the running time of the A-box validation would only be increased, with no additional functionality in the system.

Constraint 1 is very easily implemented using the inverseOf construct in DAML. This constraint indicates that the overlapsWith relation is symmetric:

```
<daml:ObjectProperty rdf:ID="overlapsWith">
  <rdfs:label>overlaps with</rdfs:label>
  <rdfs:comment>
    Course that can not give credit points together with the other
    course.
  </rdfs:comment>
  <rdfs:domain rdf:resource="#AnyCourse"/>
  <rdfs:range rdf:resource="#AnyCourse"/>
  <daml:inverseOf rdf:resource="#overlapsWith"/>
</daml:ObjectProperty>
```

Constraints 2 and 3 have been reformulated and embedded in the definition of the student concept class. This definition expresses that there are three types of students, Complete Master students, Master students and International Master students. Complete Master students have as only requisite to follow the Complete Master type of study. Master students correspond to constraint 2 and International Master students correspond to constraint 3. The whole definition becomes as follow:

```
<daml:Class rdf:ID="Student">
  <rdfs:label>Student</rdfs:label>
  <rdfs:comment>A DTU student.</rdfs:comment>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#CompleteMasterStudent"/>
    <daml:Class rdf:about="#MasterStudent"/>
    <daml:Class rdf:about="#InternationalMasterStudent"/>
  </daml:unionOf>
</daml:Class>

<daml:Class rdf:ID="CompleteMasterStudent">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#follows"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#follows"/>
      <daml:toClass rdf:resource="#CompleteMaster"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#plans"/>
      <daml:toClass>
        <daml:Restriction>
          <daml:onProperty rdf:resource="#includes"/>
          <daml:toClass rdf:resource="#Course"/>
        </daml:Restriction>
      </daml:toClass>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>

<daml:Class rdf:ID="MasterStudent">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#follows"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#follows"/>
      <daml:toClass rdf:resource="#Master"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#plans"/>
      <daml:toClass>
        <daml:Restriction>
          <daml:onProperty rdf:resource="#includes"/>
          <daml:toClass rdf:resource="#MasterCourse"/>
        </daml:Restriction>
      </daml:toClass>
```

```
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>

<daml:Class rdf:ID="InternationalMasterStudent">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#follows"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#follows"/>
      <daml:toClass rdf:resource="#InternationalMaster"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#plans"/>
      <daml:toClass>
        <daml:Restriction>
          <daml:onProperty rdf:resource="#includes"/>
          <daml:toClass
                  rdf:resource="#InternationalMasterCourse"/>
        </daml:Restriction>
      </daml:toClass>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>
```

Constraint 4 has been implemented in a similar manner. The student concept class has been modified to express that besides being either Complete Master, Master or International Master, the student has furthermore to belong to either English student or Danish student, corresponding to the language the student speaks. As in constraint 4, English students (students that speak English) may only plan to take courses taught in English. Danish students on the other hand may take courses taught in any language. The whole definition becomes as follows:

```
<daml:Class rdf:ID="Student">
  <rdfs:label>Student</rdfs:label>
  <rdfs:comment>A DTU student.</rdfs:comment>
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class>
      <daml:unionOf rdf:parseType="daml:collection">
        <daml:Class rdf:about="#CompleteMasterStudent"/>
        <daml:Class rdf:about="#MasterStudent"/>
        <daml:Class rdf:about="#InternationalMasterStudent"/>
      </daml:unionOf>
    </daml:Class>
    <daml:Class>
      <daml:unionOf rdf:parseType="daml:collection">
        <daml:Class rdf:about="#EnglishStudent"/>
        <daml:Class rdf:about="#DanishStudent"/>
      </daml:unionOf>
    </daml:Class>
  </daml:intersectionOf>
</daml:Class>
```

```
<daml:Class rdf:ID="EnglishStudent">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#speaks"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#speaks"/>
      <daml:toClass rdf:resource="#English"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#plans"/>
      <daml:toClass>
        <daml:Restriction>
          <daml:onProperty rdf:resource="#includes"/>
          <daml:toClass>
            <daml:Restriction>
              <daml:onProperty rdf:resource="#isTaughtIn"/>
              <daml:toClass rdf:resource="#English"/>
            </daml:Restriction>
          </daml:toClass>
        </daml:Restriction>
      </daml:toClass>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>

<daml:Class rdf:ID="DanishStudent">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#speaks"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#speaks"/>
      <daml:toClass rdf:resource="#Danish"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#plans"/>
      <daml:toClass>
        <daml:Restriction>
          <daml:onProperty rdf:resource="#includes"/>
          <daml:toClass>
            <daml:Restriction>
              <daml:onProperty rdf:resource="#isTaughtIn"/>
              <daml:toClass rdf:resource="#Language"/>
            </daml:Restriction>
          </daml:toClass>
        </daml:Restriction>
      </daml:toClass>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>
```

This concludes the implementation of the static part of the DAML file.

The dynamic part of the DAML file is the one containing course information. This part of the file is to be generated every time the system is restarted or every time a course is

classified. A WebL script has been developed for generating the complete DAML file to be used by the system, having as input the static parts of the file, the HTML pages of the DTU Course Catalogue and XML files containing any classification information about courses. This script is called course_reader.webl. The final DAML file generated by this script is called dtu.daml.

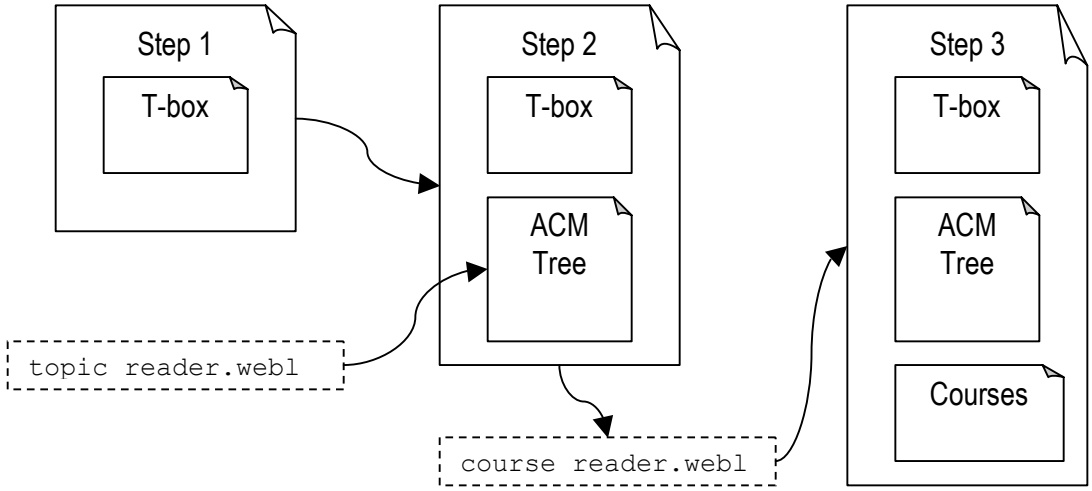


Figure 6-1: Step 1 is done manually. Step 2 is done by creating the ACM Tree with topic_reader.webl and adding it manually to the knowledge base. Step 3 is done by course_reader.webl every time the system is restarted.

Any changes done to course classifications are not directly saved in the file dtu.daml. All information about courses that are not already in the DTU Course Catalogue is saved in a separate file for each course. These files will be used by the course_reader.webl script to create the course instances every time the DAML file is generated.

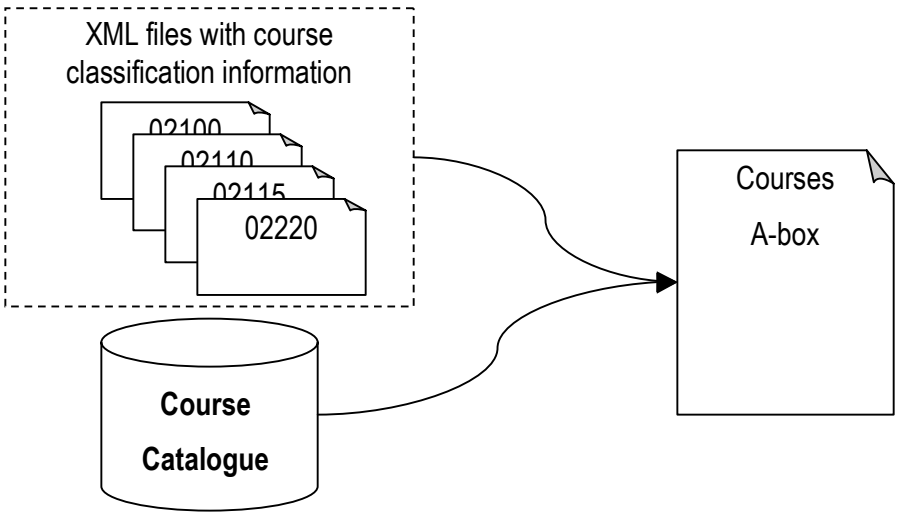


Figure 6-2: course_reader.webl reads information from the course catalogue and from XML files containing course classifications to create the course instances.

Here is an example of the content of a XML file with course classification information. The name of the file is the course number that identifies the course being classified:

```
<?xml version='1.0' encoding='UTF-8'?>
<xml>
  <type>MasterCourse</type>
  <covers>A.1</covers>
  <covers>A.2</covers>
</xml>
```

The Course Catalogue contains two kinds of HTML pages that are interesting for our system. The first one lists all existing courses at IMM, and is found at the URL http://shb.dtu.dk/default.asp?institut=imm&soeg=S%F8g+i+studieh%E5ndbogen. The only information of interest in this page is the course numbers for IMM courses. Other courses where this department is involved are not included in the courses of interest, as they belong to other institutes.

For each course number found in the above mentioned HTML page, another HTML page from the Course Catalogue is found at the URL http://shb.dtu.dk/default.asp?page=3&detail=f&lang=uk&kurser=x, where x is the course number. From this page, information about the course, as course name, language, schedules, prerequisites, overlapping courses, are retrieved.

The DAML content produced by the course_reader.webl script will be the course instances containing the information from the Course Catalogue and the XML files with classification information. Finally this script will also define the course concept classes sufficiently by enumerating all the courses that belong to each class. This is done to enable the system to check consistency of the semester plans according to the constraints implemented above. Here is an example of a course type concept class with exactly one course:

```
<daml:Class rdf:ID="CoreCourse">
  <rdfs:subClassOf rdf:resource="#Course"/>
  <rdfs:label>Core course</rdfs:label>
  <rdfs:comment>
    Mandatory course for complete master students.
  </rdfs:comment>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#LinePackageCoreCourse"/>
    <daml:Class rdf:about="#MandatoryCoreCourse"/>
    <daml:Class>
      <daml:oneOf rdf:parseType="daml:collection">
        <AnyCourse rdf:about='#Dummy'/>

        <CoreCourse rdf:ID='c02220'>
          <courseName>Concurrent Systems</courseName>
          <isTaughtIn rdf:resource='#EN'/>
          <isTaughtAt rdf:resource='#E-1A'/>
          <isTaughtAt rdf:resource='#E-1B'/>
          <hasPrerequisite>
            <PrerequisiteGroup>
```

```
            <atLeastOneCourse rdf:resource='#c02130'/>
          </PrerequisiteGroup>
        </hasPrerequisite>
        <hasPrerequisite>
          <PrerequisiteGroup>
            <atLeastOneCourse rdf:resource='#c02140'/>
          </PrerequisiteGroup>
        </hasPrerequisite>
        <covers rdf:resource='#topic-A'/>
      </CoreCourse>

    </daml:oneOf>
  </daml:Class>
 </daml:unionOf>
</daml:Class>
```

Course types that contain no courses, because no course has been classified as that type yet, may not be empty, as they will then not be sufficiently defined. A dummy[15] course has been created to solve this problem, but the interface of the system must ensure that this course never be included in a semester plan or list of passed courses. Here is an example of an empty course type:

```
<daml:Class rdf:ID="ExoticCourse">
  <rdfs:subClassOf rdf:resource="#MasterCourse"/>
  <rdfs:label>Humanistic course</rdfs:label>
  <rdfs:comment>Exotic course.</rdfs:comment>
  <daml:sameClassAs>
    <daml:Class>
      <daml:oneOf rdf:parseType="daml:collection">
        <AnyCourse rdf:about='#Dummy'/>
      </daml:oneOf>
    </daml:Class>
  </daml:sameClassAs>
</daml:Class>
```

This concludes the implementation of the whole dtu.daml file.

6.2 Installing the tools

The prototype of this system is being run on a Microsoft Windows XP platform equipped with a Java™ 2 Platform, Standard Edition, v 1.4.1.

The first tool to be installed is the Tomcat web-server. Tomcat v. 4.1.24 for Windows can be downloaded from http://jakarta.apache.org/builds/jakarta-tomcat-4.0/release/v4.1.24/bin/jakarta-tomcat-4.1.24.exe. This is quite straightforward to install. After installation the file `TOMCAT_HOME/conf/server.xml`[16] must be modified in the line

[15] A dummy course is an unexistent course, whose ID differs from other courses' by not containing digits. This course is of type AnyCourse, as it should never be permitted in a semester plan. The dummy course is included in the enumeration of every course type to prevent empty enumerations.

[16] `TOMCAT_HOME` refers to the directory where Tomcat has been installed.

containing `port="8080"` to `port="80"`. This will configure the web-server to use port 80 as the default HTTP port. Use the Start Tomcat script to start the web-server.
The root of our system will then be `TOMCAT_HOME/webapps/ROOT`. The DAML file dtu.daml must be placed in `TOMCAT_HOME/webapps/ROOT/DAML_files/T-box`. The directories `TOMCAT_HOME/webapps/ROOT/DAML_files/A-box/courses` and `TOMCAT_HOME/webapps/ROOT/DAML_files/T-box/students` must be created, so that the XML files containing course classification and the profile DAML files can be placed there.

The WebL tool is needed to produce the dtu.daml file. The dynamic part of the file is generated when the system is started. For the system to be able to run WebL scripts, the WebL.jar file must be placed in the `TOMCAT_HOME/webapps/ROOT/WEB-INF/lib/` directory. This jar file can be found in the WebL package that can be downloaded from http://www.research.compaq.com/SRC/WebL/WebL3.0h.html. To run a WebL script manually just follow the instructions in the `Readme.txt` file included in the package.

Now that the web-server is running, and the dtu.daml file has been generated and placed correctly, it can be syntax checked using the DAML Validator. If your web-server has access to the Internet (as it should have), you should change the local namespace in the dtu.daml file from `"http://localhost/DAML_files/T-box/dtu.daml#"` to `"http://xxx.xx.xx.xxx/DAML_files/T-box/dtu.daml#"` where `xxx.xx.xx.xxx` is the IP address of your machine. Then go to the web page http://www.daml.org/validator/ and input `http://xxx.xx.xx.xxx/DAML_files/T-box/dtu.daml` in the `URI` textbox and click `Submit`. The result of the syntax check with eventual warnings and error messages will then be presented in HTML format.

Next you will need the jar files contained in the JENA Toolkit, which can be downloaded from http://www.hpl.hp.com/semweb/download/Jena-1.6.1.zip. Place the files `jena.jar`, `xerces.jar`, `icu4j.jar`, `concurrent-1.3.0.jar`, `jakarta-oro-2.0.5.jar`, `antlr.debug.jar` and `sesame-client.jar` in the `TOMCAT_HOME/webapps/ROOT/WEB-INF/lib/` directory. These can be found in the /lib/ directory of the JENA Toolkit package.

Finally the reasoner RACER must be installed and started. It can be downloaded from http://www.fh-wedel.de/~mo/racer/racer-1-7.zip. Unzip the file in any directory, here called `RACER_HOME`, and start RACER by entering the following line in a DOS console: `RACER_HOME/racer.exe -http 0 -p 8088`. This will enable the TCP interface of RACER at port 8088 to be used by the system, and will disable the HTTP interface. The JRACER Java layer to access RACER services through Java methods must also be available to the web-server. Just unzip the JRACER package, which can be downloaded from http://www.fh-wedel.de/~mo/racer/jracer-1-7.zip, to the `TOMCAT_HOME/webapps/ROOT/WEB-INF/classes/` directory.

Now all tools are installed and ready to be used, and we can implement the rest of the system.

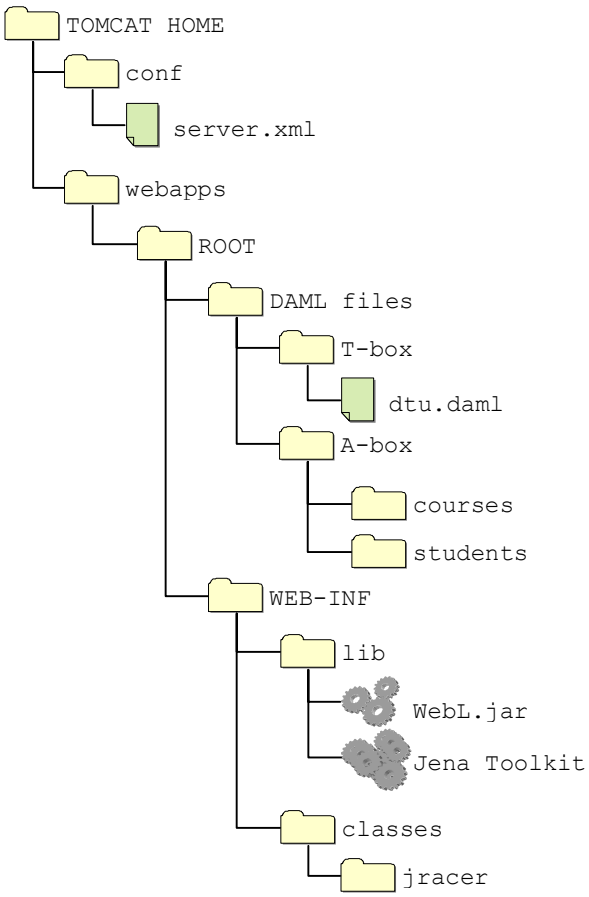


Figure 6-3: The directory structure of the web-server, with its main components.

6.3 The control engine

The control engine consists basically of the DBManager Java class. DBManager is a static class and all its public methods are synchronized to handle concurrency problems when managing the knowledge base. This class contains a number of static attributes holding information about files location, web-server address, ports, etc.

Every time DBManager is loaded by the web-server, an initialization method is run, executing the course_reader.webl script to create the dtu.daml file, loading the knowledge base into RACER (RACER must be running before this class may be loaded) and reading the DAML files into a Java model (using the DAML+OIL parser in Jena Toolkit). This initialization method is also run every time the course classifications are modified, to generate the correct dtu.daml file and to reinitialize the knowledge base held by RACER.

The files containing student and semester plan instances can only be created by DBManager. These files are maintained independently of each other, as one student profile with plans should not have any influence on other students' profiles and plans.

The file names contain the student number, making them unique, as every student may have at most one profile. For example, a student with the student number c960516 will have the DAML file c960516.daml. Here is an example of a DAML file containing a profile:

```
<?xml version='1.0' encoding='WINDOWS-1252'?>
<rdf:RDF
    xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
    xmlns:daml='http://www.daml.org/2001/03/daml+oil#'
    xmlns:dtu='http://localhost/DAML_files/T-box/dtu.daml#'
>
    <daml:Ontology
            rdf:about='http://localhost/DAML_files/T-box/dtu.daml'/>
    <dtu:Student rdf:about='http://localhost/DAML_files/
                            A-box/students/c960516.daml#c960516'>
        <dtu:follows rdf:resource='http://localhost/DAML_files/
                            T-box/dtu.daml#international-master'/>
        <dtu:speaks rdf:resource='http://localhost/DAML_files/
                            T-box/dtu.daml#EN'/>
        <dtu:plans>
            <dtu:Semester rdf:about='http://localhost/DAML_files/
                            A-box/students/c960516.daml#loi'>
                <dtu:includes rdf:resource='http://localhost/DAML_files/
                            T-box/dtu.daml#c02335'/>
                <dtu:isOn rdf:resource='http://localhost/DAML_files/
                            T-box/dtu.daml#E-'/>
            </dtu:Semester>
        </dtu:plans>
        <dtu:plans>
            <dtu:Semester rdf:about='http://localhost/DAML_files/
                            A-box/students/c960516.daml#sp09'>
                <dtu:isOn rdf:resource='http://localhost/DAML_files/
                            T-box/dtu.daml#F-'/>
                <dtu:includes rdf:resource='http://localhost/DAML_files/
                            T-box/dtu.daml#c02262'/>
            </dtu:Semester>
        </dtu:plans>
    </dtu:Student>
</rdf:RDF>
```

The public methods of DBManager are:

`Course getCourse(String courseNr)` – This method returns the Course instance that has the given course number as ID.

`String saveCourse(String courseNr, String type, String[] selectedTopics)` – This method saves the course instance that has the given course number as ID. The course type and covered topics are eventually replaced by the given ones, if they are not null. If the course type or covered topics are changed from previous values, the XML file containing the classification is created or modified (if already existent) and the initialization of the class is run.

`Profile getProfile(String studentNr)` – This method returns the Profile instance that has the given student number as ID. DAML files containing a student profile are only read and included in the Java model if this method is called with their student number.

`String saveProfile(String studentNr, String passedCourses, Iterator plans, String[] topics)` – This method saves the profile instance that has the given student number as ID. The passed courses, semester plans and topics of interest are eventually replaced by the given ones, if they are not null. The DAML file corresponding to the student profile is modified.

`String createProfile(String studentNr, String language, String typeOfStudy)` – This method creates a new student profile with the given student number, language and type of study. The corresponding DAML file is created.

`String deleteProfile(String studentNr)` – This method deletes the DAML file corresponding to the given student number, and removes the profile from the Java model.

`String validatePlan(String studentNr, Plan plan)` – This method checks if the plan is consistent with the T-box and the profile of the student who planned it. The profile and plan information are entered in RACER, the consistency of the A-box is checked, and finally the profile and plan information are removed from RACER again, so that it is ready for the next validation. Unfortunately, the only information returned by this method is if the plan is consistent or not. No explanatory messages are given.

6.4 Entity classes

The entity classes were implemented to hold information from the knowledge base that will be used by the interface. These classes have mostly get and set methods that assist the interface classes in performing their job. These classes are the Course class, the Schedule class, the Profile class and the Plan class.

A special entity class is the ClassificationTree class, which holds information about a class hierarchy of a concept in the knowledge base. This class contains a tree structure of the classes, where each node of the tree (TreeNode class) contains the ID of the class and its label.

The entity classes contain as little information from the knowledge base as possible, limiting themselves to containing only what is needed by the interface. This allows the T-box of the knowledge base to be modified to a certain extent, i.e. this maintains a certain flexibility to the knowledge base structure.

6.5 The interface

The interface of the system consists of Java Server Pages generating HTML and Boundary classes assisting the JSP in their tasks. The boundary classes send requests from the user to the DBManager and convert the responses to HTML code, which is then used by the corresponding JSP to generate the whole HTML page.

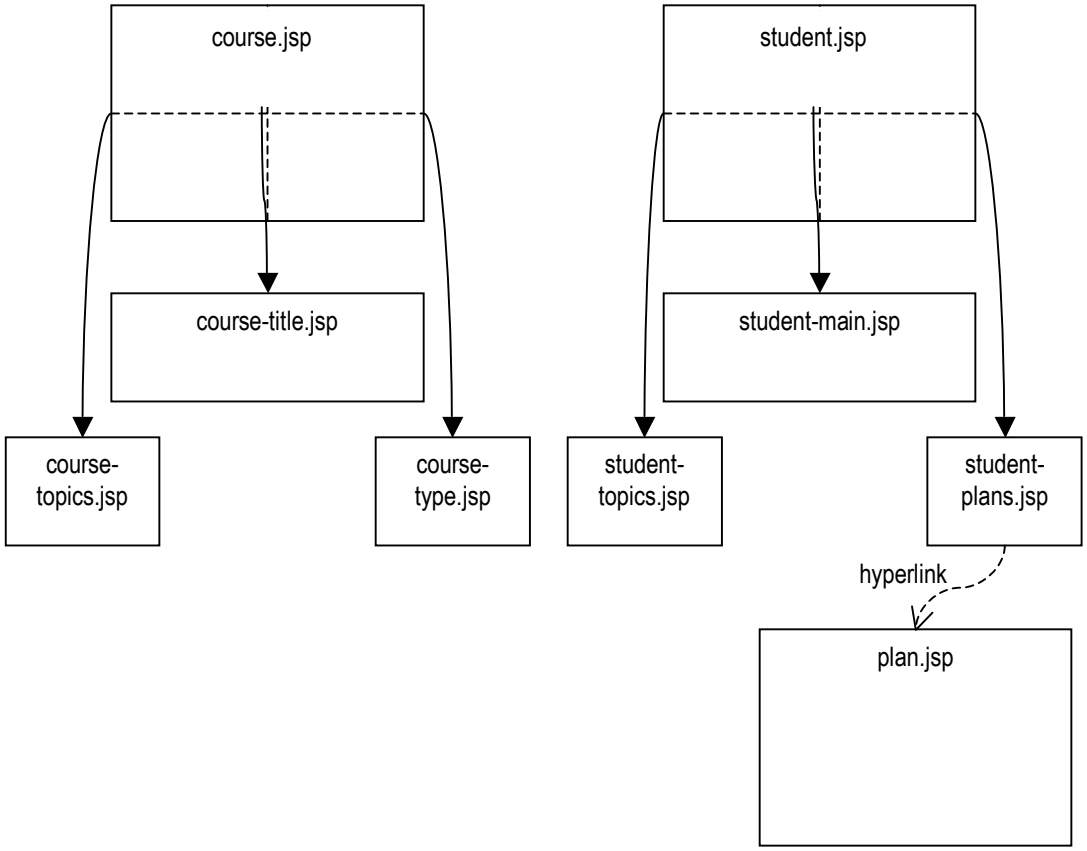


Figure 6-4: The structure of the Java Server pages.

The Course interface starts at `course.jsp`. If this page is called without a course number, it simply displays a HTML page asking for a course number.

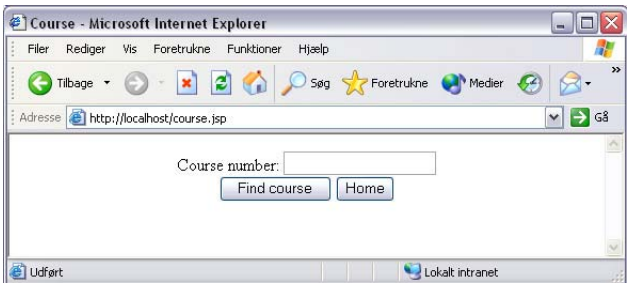


Figure 6-5: course.jsp without a course number.

If the course number is given, this page will display a HTML page with frames containing `course-title.jsp`, `course-topics.jsp` and `course-type.jsp`. The `course-title.jsp` simply displays information about the course. The `course-topics.jsp` shows the classification tree for ACM topics, where the topics covered by the course are selected. The `course-type.jsp` shows the classification tree for course types, where the course type is selected. The selected items in `course-topics.jsp` and `course-type.jsp` can be changed and saved.

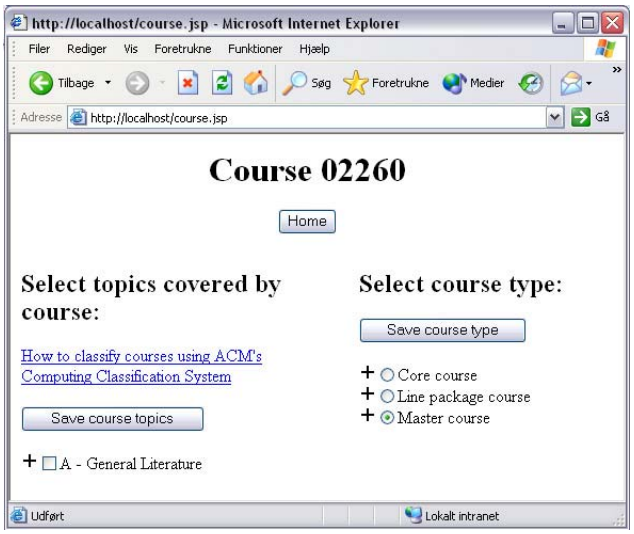


Figure 6-6: course.jsp with course number 02260.

The Student interface starts at `student.jsp`. If this page is called without a student number, it simply displays a HTML page asking for a student number.

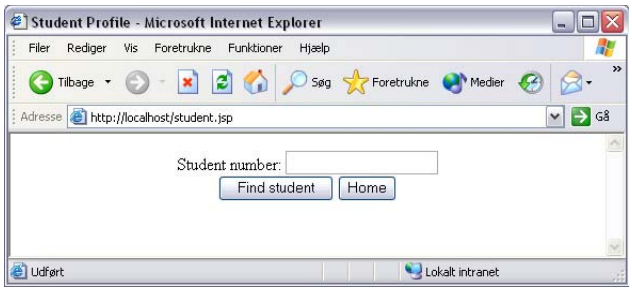


Figure 6-7: student.jsp without a student number.

If the student number is given, this page will display a HTML page with frames containing `student-main.jsp`, `student-topics.jsp` and `student-plans.jsp`.

If no profile for the given student yet exists, the `student-topics.jsp` and `student-plans.jsp` will be blank, and the `student-main.jsp` will display a HTML page that enables the creation of a student profile. This page will request the student's language and type of study and the profile may be created.

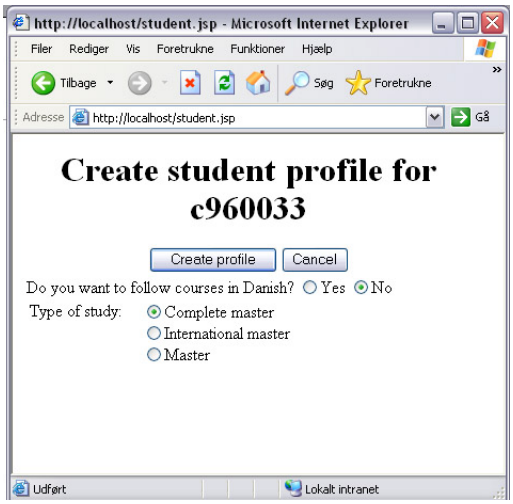


Figure 6-8: student.jsp with a student number that does not have a profile yet.

If a profile for the given student already exists, the `student-main.jsp` displays language and course type information and information about passed courses can be entered and saved here. The `student-topics.jsp` shows the classification tree for ACM topics, where the student's topics of interest are selected. The selected items in `student-topics.jsp` can be changed and saved. The `student-plans.jsp` shows a list of semester plans that the student has already made. This list may be empty. Besides, this page allows entering a new plan ID and a season for creating a new semester plan. Already created plans may also be deleted here. Clicking on a plan ID from the plan list will link to the Plan interface.

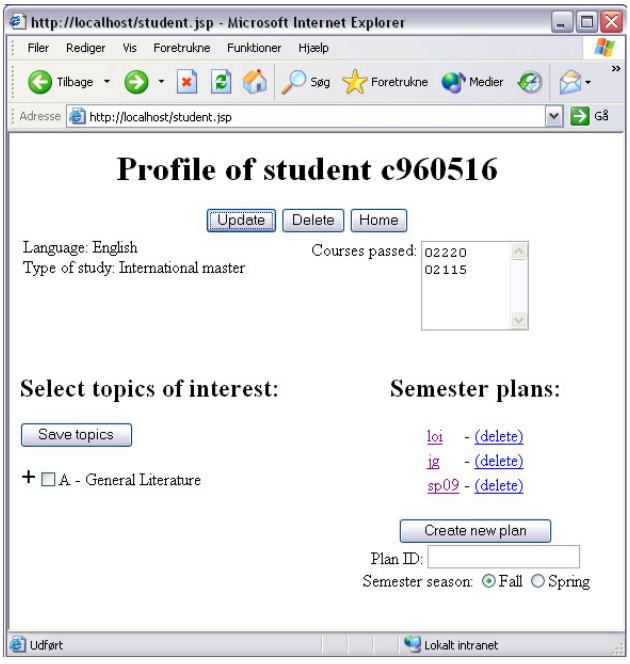


Figure 6-9: student.jsp with a student number that already has a profile.

The Plan interface consists only of `plan.jsp`. This page must always be called with a student number and a plan ID. This page displays a HTML containing a semester schedule plan. One or more courses can be placed in each schedule. For entering a new course to the plan, a course number may be entered. The course must exist in the current version of the Course Catalogue, and it must be taught at the season of the semester plan. To remove courses from the plan, the delete button next to the course must be clicked. To check if the semester plan complies with the system constraints, simply click on the Check plan button.

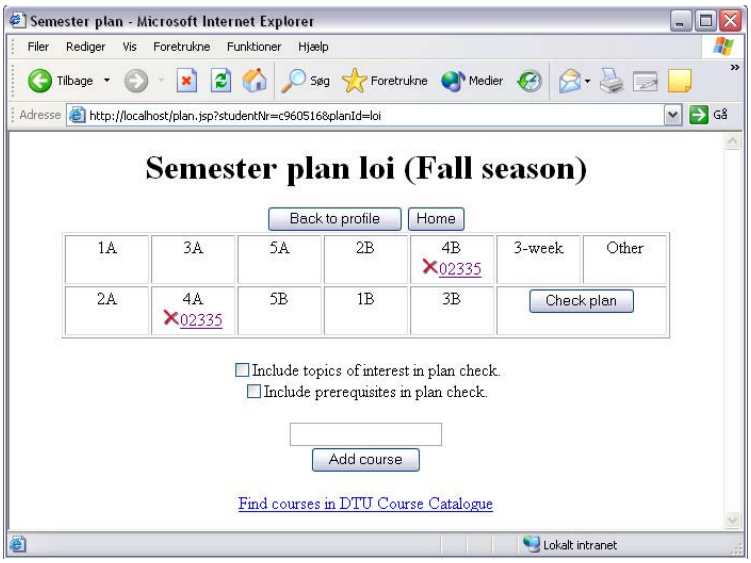


Figure 6-10: plan.jsp with a course already in the plan.

The interfaces do not permit inconsistent data to be saved through to the knowledge base. All information entered through the interface is therefore validated for correct input, and no input is enabled when the input is not allowed in a specific case. Examples of these are that course numbers must be checked to contain only digits, and courses may only contain one course type. The semester plan's consistency, on the other hand, is not checked by the interface, but must be checked through the validation method.

7 TEST

This chapter contains the tests made during and after the development of the system. The first test was simply to determine an appropriate tool for solving the A-box validation part of the system. The following tests were done in order to check the correctness of the system: A syntax check of the DAML files, a test of system requirements fulfillment, a unit test and an integration test.

7.1 Test of A-box Validation tools

In order to be able to choose which knowledge base checker could handle A-box consistency validation, two tests were performed: The Nationality example and The Color Option example. Both tests had as purpose to check a very simple model with an inconsistent instance. The DAML+OIL models of the examples are found in the appendix.

The tools participating in the test were DAML Validator and RACER. Based on the test results, RACER was chosen as the validation tool.

7.1.1 The Nationality example

The model of this example contains a person entity, a nationality entity and a country entity. A person has a nationality and is born in a country.

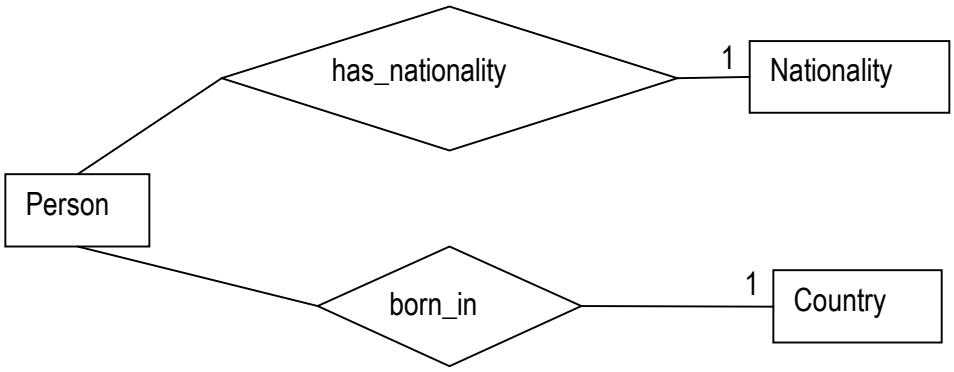


Figure 7-1: The Nationality ER-diagram

The country in this example has been classified as European country and other country. The nationality as well has been classified as European nationality and other nationality.

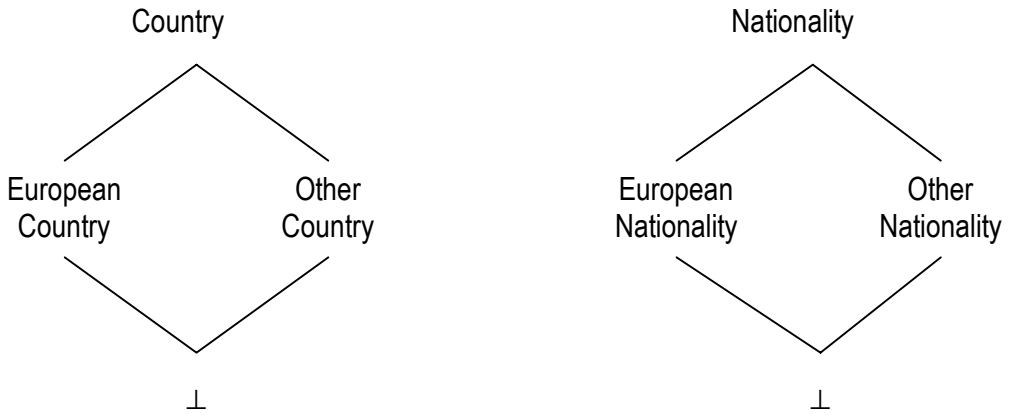


Figure 7-2: The Nationality Hasse diagrams.

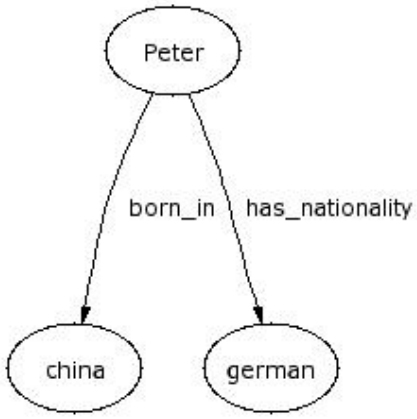
The model only requires that persons born in European countries have European nationalities, and persons born in other countries have other nationalities. No person is allowed to be both European and other at the same time. Notice that this model does not exclude the case of a person born in England with a German nationality, but this is only a simple example.

```
Person ≐ EuropeanPerson ⊔ OtherPerson

⊥ ≐ EuropeanPerson ⊓ OtherPerson

EuropeanPerson ≐ =1 born_in.EuropeanCountry ⊓
      =1 has_nationality.EuropeanNationality

OtherPerson ≐ =1 born_in.OtherCountry ⊓
      =1 has_nationality.OtherNationality
```



The following A-box was instantiated:

```
{german} ⊑ EuropeanNationality

{chinese} ⊑ OtherNationality

{germany} ⊑ EuropeanCountry

{china} ⊑ OtherCountry

{peter} ⊑ Person

{(peter, german)} ⊑ has_nationality
```

```
{(peter, china)} ⊑ born_in
```

When asking RACER if this A-box is consistent the answer is True, even though it is not. DAML Validator did not give any warnings or error messages in this case.

The model was changed so that the concepts of our T-box were sufficiently defined, i.e. the members of the concepts were enumerated. In this case RACER answered that the A-box could not be proven to be consistent. DAML Validator did not give any warnings or error messages in this case either.

```
EuropeanCountry ≐ {germany}

OtherCountry ≐ {china}

EuropeanNationality ≐ {german}

OtherNationality ≐ {chinese}
```

This example proofs that if concepts are not sufficiently defined, neither RACER nor DAML Validator are useful for checking A-box consistency. This test added an additional requirement to the system, that all concepts in the model be defined sufficiently before validating the semester plans.

7.1.2 The Color Option example

The model of this example contains a uniform entity, an item entity and a color entity. A uniform has one color and one or more uniform parts. An item has one or more color options.

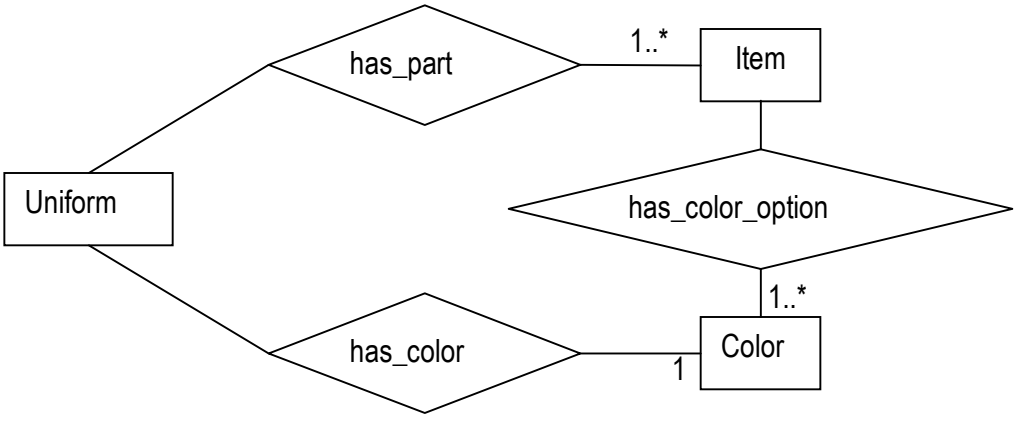


Figure 7-3: The Color Option ER-diagram

The color in this example has been classified as dark color and light color.

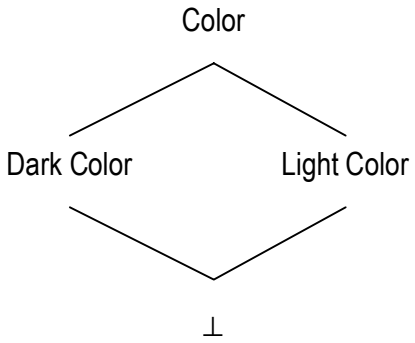


Figure 7-4: The Color Option Hasse diagram.

This model requires that if the uniform has a light color, then all of its parts exist in a light color option, and if uniforms have a dark color, its parts must exist in a dark color. This constraint is to check that a uniform is not formed by parts that are not available in the uniform's color. Because of the result of the previous test, the concepts dark color and light color, as well as the item concept were sufficiently defined.

```
Color ≐ DarkColor ⊔ LightColor

DarkColor ≐ {blue}

LightColor ≐ {red}

Item ≐ {pants, blouse}

{(blouse, red), (pants, red), (pants, blue)} ⊑ has_color_option

Uniform ≐ DarkUniform ⊔ LightUniform

DarkUniform ≐ =1 has_color.DarkColor ⊓

   ≤1 has_part.(∃has_color_option.DarkColor)

LightUniform ≐ =1 has_color.LightColor ⊓

    ≤1 has_part.(∃has_color_option.LightColor)
```

The following A-box was instantiated:

```
{uniform1} ⊑ Uniform

{(uniform1, blue)} ⊑ has_color
```

```
{(uniform1, pants), (uniform1, blouse)} ⊑ has_part
```

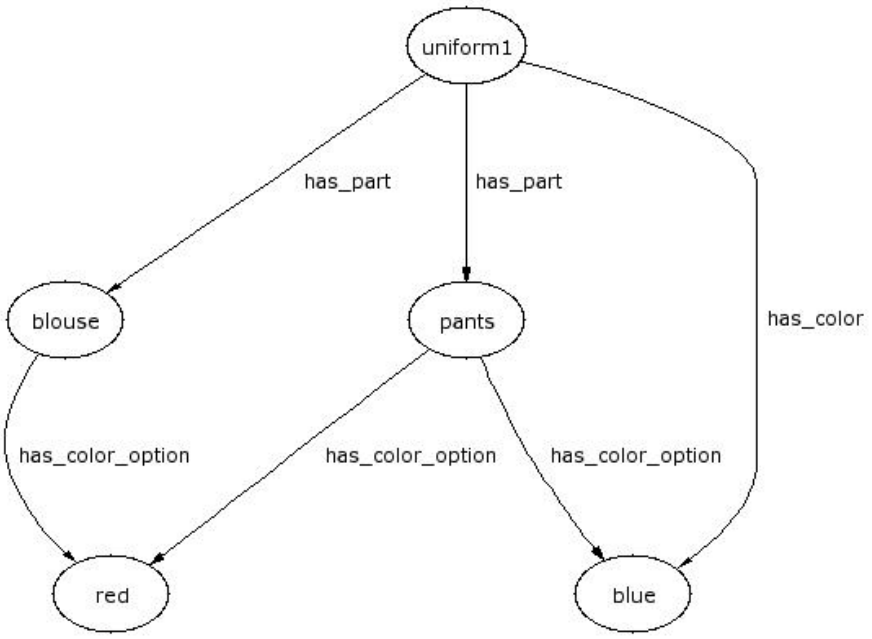


Figure 7-5: An inconsistent A-box.

When asking RACER if this A-box is consistent the answer is True, even though it is not. DAML Validator did not give any warnings or error messages in this case.

Relations, also called properties or roles, can not be sufficiently defined in DAML. Only the domain and range of the property can be sufficiently defined. In our example the property has_color_option may contain the tuple (blouse, blue), even though it is not defined in our example. Therefore RACER is unable to find any inconsistency in the given A-box.

This test added an additional requirement to the system, namely that all constraints in the model do not contain existential quantifications for properties, as these can not be validated.

7.2 Checking the syntactical correctness of DAML files

The DAML Validator takes a DAML file as input and produces an HTML output file containing any warnings or error information about the input file. The `dtu.daml` file and example students' DAML files were checked for syntax errors using DAML Validator.

Errors found in students' files were not corrected directly in the DAML file, as these are created by the interface and the DBManager. Therefore the interface and control engine were corrected in order to produce syntactically correct DAML files.

Errors found in ACM topic classes or instances, as well as errors found in course instances were corrected in the corresponding WebL program that generated the files.

These syntax checks should be done every time the T-box or a WebL program or the interface or the control engine is modified.

7.3 Checking that the requirements have been met

This part of the test checks that the system implemented is able to complete the normal flow of all use cases from the requirements specification, and that alternative flows are handled correctly. Besides, this section checks if all supplementary requirements have been met.

7.3.1 Test of Edit Course use case

Normal flow: The course number of an existing course was entered in the system. Some course topics were selected and saved. A course type was selected and saved. The system did not allow more than one course type to be selected. The course classification changes were saved correctly.

Alternative flow: The course number of a non-existing course was entered in the system. The system provided an error message indicating the mistake.

7.3.2 Test of Get Profile use case

Normal flow 1: The student number of a non-existing profile was entered in the system. The student's language and type of study were selected. The system did not allow more than one language and one type of study to be selected. The student profile was created correctly.

Normal flow 2: The student number of an existing profile was entered in the system. The student profile is shown.

Alternative flow: A student number containing characters that are not acceptable in the knowledge base was entered. The system provided an error message indicating the mistake and naming which characters are accepted.

7.3.3 Test of Edit Profile use case

Normal flow: Some passed courses were entered and saved. Some topics of interest were selected and saved. The changes performed on the profile were saved correctly. *Courses not in the Course Catalogue and not specified as prerequisites or overlapping courses to the ones in the catalogue were not accepted by the system, as they do not exist in the knowledge base*. This error was corrected.

Alternative flow: Some passed courses containing characters other than digits (which are not acceptable) were entered. The system provided an error message indicating the mistake.

7.3.4 Test of Delete Profile use case

Normal flow: The profile was deleted correctly.

7.3.5 Test of Create Plan use case

Normal flow: A plan ID was entered and a season selected. The system did not allow more than one season to be selected. The system did not allow that a season not be selected. The semester plan was created correctly.

Alternative flow 1: When no plan ID was entered the system provided an error message indicating the mistake.

Alternative flow 2: *When the plan ID already existed the system provided no error message indicating the mistake*. The plan though was not created again. *But if different seasons were given, the semester plan contains two seasons, which is inconsistent with the knowledge base*. This error was corrected.

Alternative flow 3: When a plan ID containing characters that are not acceptable in the knowledge base was entered the system provided an error message indicating the mistake and naming which characters are accepted.

7.3.6 Test of Delete Plan use case

Normal flow: The plan was deleted correctly.

7.3.7 Test of Get Plan use case

Normal flow: The plan was shown correctly.

7.3.8 Test of Edit Plan use case

Normal flow: Courses were added to the plan and saved correctly. *Courses were not removed from the plan and therefore not saved correctly.* This error only occurs when clicking the OK button too fast. The cause of this error could not be found and therefore was not corrected. This is not considered a serious problem, as clicking on the delete button again successfully deletes the course from the plan.

Alternative flow 1: When a course that does not exist was added the system provided an error message indicating the mistake.

Alternative flow 2: When a course that has no schedules at the season of the plan was added the system provided an error message indicating the mistake.

7.3.9 Test of Validate Plan use case

Normal flow 1: When a valid plan was validated the answer was correct[17] but the answering time was unacceptably long.

Normal flow 2: When an invalid plan was validated the answer was correct but the answering time was unacceptably long.

7.3.10 Test of Maintain ontology use case

Normal flow: When the system was restarted it continued to function correctly.

7.3.11 Test of supplementary requirements

The system functions as specified in the supplementary requirements on a MS Internet Explorer v.6.0. The system has a very simple graphical interface and error messages are helpful in assisting the user correct eventual mistakes.

One of the requirements that were not accomplished in its detail was the updating of the course instances in the knowledge base. It was required that this part of the knowledge base is updated periodically in order to reflect changes made to the Course Catalogue. The system only updates course instances when one of the courses is classified or when the system is restarted, and this does not guarantee a periodical updating. Even though this error is not severe for a prototype version, it should not be accepted in a running version of the system.

The dynamic parts of the `dtu.daml` file were correctly generated every time it was tested. One of the tests though revealed an error. The test consisted of restarting the system when there was no connection to the Internet, and therefore the Course Catalogue could not be accessed. This locked the system that stopped responding, even after the Internet connection was re-established. The lock could only be removed by restarting the web-server. This error is not acceptable for a running version of the system.

During the usability test, a not severe error was found. The error is that only the entrance of the web portal issues a message if the knowledge base is being loaded (as this action may take several minutes to complete). All other pages simply take very long to appear in this situation.

[17] The validation only included the constraints 2, 3 and 4, as the other constraints could not be implemented.

7.4 Test Conclusion

The system requirement of validating semester plans, which can be considered the most crucial one for the system, was not satisfactorily met. This problem could not be solved and therefore the system is unacceptable.

Because the number of concept instances in the system has influence on the running time of the validation function, the ACM classification was not added to the T-box in its whole extent, but only a very little part of it was added for exemplification purposes only. This did not solve the problem, but allowed the response time to be reduced for test purposes.

The further implementation of validation functions (the addition of a check of topics of interest and prerequisites) was not completed, as the main validation already was estimated to be unsolvable in a satisfactorily manner.

Some of the errors found in the test were not corrected either. These errors are not severe in a prototype version, but are not acceptable in a running version of the system. As the main function of the system could not be implemented, a running version of the system is not realistic, and therefore correcting these errors was simply not worth the effort.

8 CONCLUSION

The goal of this thesis was to develop a system using a new technology. This conclusion is divided in two parts. The first part describes what was learnt about this new technology during the project. The second part refers more specifically to developing a Web system using new technologies.

8.1 Semantic Web

The semantic web element in this project is the knowledge base. The contents of the knowledge base comprise not only the system data but also the data schema (i.e. metadata). The knowledge base is available through the Internet and can therefore be accessed in a XML format by other systems and agents that will be able to understand and process its contents.

It was possible to augment this knowledge base with some complex description logics expressing additional constraints to the data. This is normally not possible when working with standard database systems.

The downside of the semantic web is that even though we can express very intricate information with this new technology, there has not been much advance in developing the tools that actually handle the information. Especially the description logics contained in semantic web documents can not be exploited satisfactorily with the tools available at the moment.

The part of the semantic web content that can be used in its full potential can also be created with simpler, less expressive languages or well-known database systems. DAML+OIL is intended to be both dynamic and flexible. A certain tension exists between the use of restrictions for validation purposes versus its use for inference purposes. In this project validation was favored, but it seems that most of DAML+OIL researchers and developers give preferentiality to inference. In any of the cases, handling extensive and complex DAML+OIL documents require a lot of computation. Obviously, further research is necessary before we will be able to actually implement systems with this technology. But it seems that the project around the Semantic Web is becoming less ambitious, thus removing a great part of the requirements specified for DAML+OIL to develop a new language (OWL).

DAML+OIL is the currently recommended standard for adding description logics to semantic web documents. It is difficult to say if DAML+OIL is based on knowledge about description logics, or if it simply coincided in its purpose and ended as a description logics language. But my impression is that the people promoting the use of DAML+OIL try to hide from users that it actually is description logics. Description Logics is a very complex discipline that requires an extensive knowledge of logics theory, while DAML+OIL is intended to be used by the general public to add metadata markup to their web documents. My opinion is that this is a very ambitious vision.

Semantic Web technologies are still in an early stage of research. Many of the tools and standards have been modified and improved in the same period this project was being made. Semantic Web technologies are not mature enough for system developers to take full advantage of them yet.

8.2 Developing a system with new technologies

The system developed in this project is able to assist students (and course instructors) to add semantic data to documents available through the Web. Unfortunately it was not possible to meet the requirement that the system be able to assist students plan their semesters. Only very few of DTU rules can be checked by the system, and the running time of this function is unsatisfactorily long.

The cause of the development failure was the absence of proper tools to solve the problem. Adding the creation of a proper tool to the scope of this project is unrealistic, as it would require a more extensive period of time and more resources.

Many optimizations of the system could be suggested, in order to make the system faster, more secure, more user-friendly, etc. None of these suggestions would make this system usable, without solving the problem around semester plan validation.

The system requirements could have been changed to include search of courses instead of validation. The course search requirement was actually considered as an extension of the current requirements. But in an ideal system development, the original requirements should always be met before implementing extensions, and this could not be accomplished in this project. Other technologies could have been considered to meet the system requirements, but this would also fall out of the scope of this project.

The main failure of this project is that it was allowed to extend itself to the point of a prototype, without being sure that the objectives of the project were possible or not. Too many resources were used in vain. But this conclusion is only in a system development point of view. This project contributed to a better understanding of Semantic Web technologies. It became clearer what the limitations of these technologies are and what they are capable of actually solving.

I would not recommend the use of new technologies, which are still in a research stage, to be used in developing commercial systems, as the risk of failure is too great. They should only be used if there are enough resources available to contribute to the research.

9 REFERENCES

[ACM] *The ACM Computing Classification System [1998 Version] Valid in 2002* http://www.acm.org/class/1998/

[DAML] F.V. Harmelen, P.F. Patel-Schneider, I. Horrocks, *Reference description of the DAML+OIL (March 2001) ontology markup language*, http://www.daml.org/2001/03/reference

[DAML2] M. Dean, *DAML+OIL for Application Developers: An Introduction to the Semantic Web*, SPAWAR Systems Center, 2002, http://www.daml.org/2002/03/tutorial/slide1-0.html

[DL] F. Baader, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider, *The Description Logic Handbook: Theory, implementation, and applications*, Appendix I, 2002.

[DL2] *Semantic Web Chalk Talk – Amateur Intro to Description Logics*, HP Labs, 2001, http://www.hpl.hp.com/semweb/download/DescriptionLogicsIntro.pdf

[ER] H. Garcia-Molina, J. D. Ullman, J. Widom, *Database Systems The Complete Book*, Prentice Hall, 2002.

[XML] K. Ahmed et al., *Professional XML meta data*, Wrox Press, 2001

[RACER] V. Haarslev, R. Möller, *RACER Users' Guide and Reference Manual Version 1.7.6*, 2002

[RDF] T. Bray, *What is RDF?*, O'Reilly XML.com, 2001, http://www.xml.com/pub/a/2001/01/24/rdf.html

[UML] I. Jacobson, G. Booch, J. Rumbaugh, *The Unified Modeling Language User Guide*, Addison-Wesley Pub Co, 2000.

[USDP] I. Jacobson, G. Booch, J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley Pub Co, First edition, 1999.

[WEBL] H. Marais, *WebL – A Programming Language for the Web*, Compaq Systems Research Center (SRC), 1999

9.1 Web Resources containing information about Semantic Web

daml.org – http://www.daml.org/

W3C Semantic Web Activity – http://www.w3.org/2001/sw/

W3C Web Ontology (WebOnt) Working Group – http://www.w3.org/2001/sw/WebOnt/

semanticweb.org (Semantic Web news) – http://www.semanticweb.org/

OntoWeb (European Semantic Web Thematic Network) – http://www.ontoweb.org

HP Labs Semantic Web Research – http://www.hpl.hp.com/semweb/

APPENDIX A – THE DTU ONTOLOGY FILE

This copy of the dtu.daml file does not contain the courses from the Course Catalogue, but only the static part of the file.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns     ="http://localhost/DAML_files/T-box/dtu.daml#"
>

  <daml:Ontology rdf:about="">
    <rdfs:comment>
      An ontology about DTU course planning.
      This plan only takes into account course classification,
      languages and types of study.
    </rdfs:comment>
    <daml:imports rdf:resource="http://www.daml.org/2001/03/daml+oil"/>
  </daml:Ontology>

  <!-- ********************** PROPERTIES ********************** -->

  <daml:ObjectProperty rdf:ID="follows">
    <rdfs:label>follows</rdfs:label>
    <rdfs:comment>The type of study the student follows.</rdfs:comment>
    <rdfs:domain rdf:resource="#Student"/>
    <rdfs:range rdf:resource="#TypeOfStudy"/>
  </daml:ObjectProperty>

  <daml:ObjectProperty rdf:ID="plans">
    <rdfs:label>plans</rdfs:label>
    <rdfs:comment>The semester the student is planning to take.</rdfs:comment>
    <rdfs:domain rdf:resource="#Student"/>
    <rdfs:range rdf:resource="#Semester"/>
    <rdf:type
       rdf:resource="http://www.daml.org/2001/03/daml+oil#UnambiguousProperty"/>
  </daml:ObjectProperty>

  <daml:ObjectProperty rdf:ID="includes">
    <rdfs:label>includes</rdfs:label>
    <rdfs:comment>The courses included in the semester plan.</rdfs:comment>
    <rdfs:domain rdf:resource="#Semester"/>
    <rdfs:range rdf:resource="#Course"/>
  </daml:ObjectProperty>

  <daml:ObjectProperty rdf:ID="isTaughtIn">
    <rdfs:label>is taught in</rdfs:label>
    <rdfs:comment>The language the course is taught in.</rdfs:comment>
    <rdfs:domain rdf:resource="#AnyCourse"/>
    <rdfs:range rdf:resource="#Language"/>
  </daml:ObjectProperty>

  <daml:ObjectProperty rdf:ID="speaks">
    <rdfs:label>speaks</rdfs:label>
    <rdfs:comment>The language the student speaks.</rdfs:comment>
    <rdfs:domain rdf:resource="#Student"/>
    <rdfs:range rdf:resource="#Language"/>
  </daml:ObjectProperty>
```

```
<daml:ObjectProperty rdf:ID="isOn">
  <rdfs:label>is on</rdfs:label>
  <rdfs:comment>The season of the semester.</rdfs:comment>
  <rdfs:domain rdf:resource="#Semester"/>
  <rdfs:range rdf:resource="#Season"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="isTaughtAt">
  <rdfs:label>is taught at</rdfs:label>
  <rdfs:comment>A course schedule.</rdfs:comment>
  <rdfs:domain rdf:resource="#Course"/>
  <rdfs:range rdf:resource="#Schedule"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="inSeason">
  <rdfs:label>in season</rdfs:label>
  <rdfs:domain rdf:resource="#Schedule"/>
  <rdfs:range rdf:resource="#Season"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="inModule">
  <rdfs:label>inModule</rdfs:label>
  <rdfs:domain rdf:resource="#Schedule"/>
  <rdfs:range rdf:resource="#Module"/>
</daml:ObjectProperty>

<daml:DatatypeProperty rdf:ID="courseName">
  <rdfs:label>Course name</rdfs:label>
  <rdfs:domain rdf:resource="#Course"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</daml:DatatypeProperty>

<daml:ObjectProperty rdf:ID="overlapsWith">
  <rdfs:label>overlaps with</rdfs:label>
  <rdfs:comment>
    Course that can not give credit points together with the other course.
  </rdfs:comment>
  <rdfs:domain rdf:resource="#AnyCourse"/>
  <rdfs:range rdf:resource="#AnyCourse"/>
  <daml:inverseOf rdf:resource="#overlapsWith"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="hasPrerequisite">
  <rdfs:label>has prerequisite</rdfs:label>
  <rdfs:comment>A prerequisite group of this course.</rdfs:comment>
  <rdfs:domain rdf:resource="#Course"/>
  <rdfs:range rdf:resource="#PrerequisiteGroup"/>
  <rdf:type
     rdf:resource="http://www.daml.org/2001/03/daml+oil#UnambiguousProperty"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="atLeastOneCourse">
  <rdfs:label>at least one</rdfs:label>
  <rdfs:comment>A course in this prerequisite group.</rdfs:comment>
  <rdfs:domain rdf:resource="#PrerequisiteGroup"/>
  <rdfs:range rdf:resource="#AnyCourse"/>
</daml:ObjectProperty>
```

```
<daml:ObjectProperty rdf:ID="covers">
  <rdfs:label>covers</rdfs:label>
  <rdfs:comment>A topic covered by this course.</rdfs:comment>
  <rdfs:domain rdf:resource="#Course"/>
  <rdfs:range rdf:resource="#Topic"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="passed">
  <rdfs:label>passed</rdfs:label>
  <rdfs:comment>The courses the student has passed.</rdfs:comment>
  <rdfs:domain rdf:resource="#Student"/>
  <rdfs:range rdf:resource="#AnyCourse"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="isInterestedIn">
  <rdfs:label>is interested in</rdfs:label>
  <rdfs:comment>The topics the student is interested in.</rdfs:comment>
  <rdfs:domain rdf:resource="#Student"/>
  <rdfs:range rdf:resource="#Topic"/>
</daml:ObjectProperty>

<!-- ********************** TYPE OF STUDY ********************** -->

<daml:Class rdf:ID="TypeOfStudy">
  <rdfs:label>Type of study</rdfs:label>
  <rdfs:comment>One of the types of study a student can follow at DTU.
  </rdfs:comment>
</daml:Class>

<daml:Class rdf:ID="Master">
  <rdfs:label>Master</rdfs:label>
  <rdfs:subClassOf rdf:resource="#TypeOfStudy"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <Master rdf:ID="master">
      <rdfs:label>Master</rdfs:label>
      <rdfs:comment>The master of 2 years.</rdfs:comment>
    </Master>
  </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="CompleteMaster">
  <rdfs:label>Complete master</rdfs:label>
  <rdfs:subClassOf rdf:resource="#TypeOfStudy"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <CompleteMaster rdf:ID="complete-master">
      <rdfs:label>Complete master</rdfs:label>
      <rdfs:comment>The complete master of 5 years.</rdfs:comment>
    </CompleteMaster>
  </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="InternationalMaster">
  <rdfs:label>International master</rdfs:label>
  <rdfs:subClassOf rdf:resource="#TypeOfStudy"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <InternationalMaster rdf:ID="international-master">
      <rdfs:label>International master</rdfs:label>
      <rdfs:comment>
        The master of 2 years for international students.
      </rdfs:comment>
    </InternationalMaster>
  </daml:oneOf>
</daml:Class>
```

```
<!-- ********************** LANGUAGE ********************** -->

<daml:Class rdf:ID="Language">
  <rdfs:label>Language</rdfs:label>
</daml:Class>

<daml:Class rdf:ID="English">
  <rdfs:label>English</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Language"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <English rdf:ID="EN">
      <rdfs:label>English</rdfs:label>
    </English>
  </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="Danish">
  <rdfs:label>Danish</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Language"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <Danish rdf:ID="DA">
      <rdfs:label>Danish</rdfs:label>
    </Danish>
  </daml:oneOf>
</daml:Class>

<!-- ********************** SEMESTER ********************** -->

<daml:Class rdf:ID="Semester">
  <rdfs:label>Semester</rdfs:label>
  <rdfs:comment>A semester plan.</rdfs:comment>
  <daml:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#isOn"/>
    </daml:Restriction>
  </daml:subClassOf>
</daml:Class>

<!-- ********************** STUDENT ********************** -->

<daml:Class rdf:ID="CompleteMasterStudent">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#follows"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#follows"/>
      <daml:toClass rdf:resource="#CompleteMaster"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#plans"/>
      <daml:toClass>
        <daml:Restriction>
          <daml:onProperty rdf:resource="#includes"/>
          <daml:toClass rdf:resource="#Course"/>
        </daml:Restriction>
      </daml:toClass>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>
```

```
<daml:Class rdf:ID="MasterStudent">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#follows"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#follows"/>
      <daml:toClass rdf:resource="#Master"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#plans"/>
      <daml:toClass>
        <daml:Restriction>
          <daml:onProperty rdf:resource="#includes"/>
          <daml:toClass rdf:resource="#MasterCourse"/>
        </daml:Restriction>
      </daml:toClass>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>

<daml:Class rdf:ID="InternationalMasterStudent">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#follows"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#follows"/>
      <daml:toClass rdf:resource="#InternationalMaster"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#plans"/>
      <daml:toClass>
        <daml:Restriction>
          <daml:onProperty rdf:resource="#includes"/>
          <daml:toClass rdf:resource="#InternationalMasterCourse"/>
        </daml:Restriction>
      </daml:toClass>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>

<daml:Class rdf:ID="EnglishStudent">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#speaks"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#speaks"/>
      <daml:toClass rdf:resource="#English"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#plans"/>
      <daml:toClass>
        <daml:Restriction>
          <daml:onProperty rdf:resource="#includes"/>
          <daml:toClass>
            <daml:Restriction>
              <daml:onProperty rdf:resource="#isTaughtIn"/>
              <daml:toClass rdf:resource="#English"/>
            </daml:Restriction>
          </daml:toClass>
        </daml:Restriction>
```

```
      </daml:toClass>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>

<daml:Class rdf:ID="DanishStudent">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#speaks"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#speaks"/>
      <daml:toClass rdf:resource="#Danish"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#plans"/>
      <daml:toClass>
        <daml:Restriction>
          <daml:onProperty rdf:resource="#includes"/>
          <daml:toClass>
            <daml:Restriction>
              <daml:onProperty rdf:resource="#isTaughtIn"/>
              <daml:toClass rdf:resource="#Language"/>
            </daml:Restriction>
          </daml:toClass>
        </daml:Restriction>
      </daml:toClass>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>

<daml:Class rdf:ID="Student">
  <rdfs:label>Student</rdfs:label>
  <rdfs:comment>A DTU student.</rdfs:comment>
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class>
      <daml:unionOf rdf:parseType="daml:collection">
        <daml:Class rdf:about="#CompleteMasterStudent"/>
        <daml:Class rdf:about="#MasterStudent"/>
        <daml:Class rdf:about="#InternationalMasterStudent"/>
      </daml:unionOf>
    </daml:Class>
    <daml:Class>
      <daml:unionOf rdf:parseType="daml:collection">
        <daml:Class rdf:about="#EnglishStudent"/>
        <daml:Class rdf:about="#DanishStudent"/>
      </daml:unionOf>
    </daml:Class>
  </daml:intersectionOf>
</daml:Class>

<!-- ********************** PREREQUISITE GROUP ********************** -->

<daml:Class rdf:ID="PrerequisiteGroup">
  <rdfs:label>Prerequisite group</rdfs:label>
  <rdfs:comment> A group of subjects where one of them must be passed to
    fullfil the prerequisite.</rdfs:comment>
  <rdfs:subClassOf>
    <daml:Restriction daml:minCardinality="1">
      <daml:onProperty rdf:resource="#atLeastOneCourse"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
```

```
<!-- ********************** SEASON ********************** -->

<daml:Class rdf:ID="Season">
  <rdfs:label>Season</rdfs:label>
  <rdfs:comment>Either fall or spring.</rdfs:comment>
</daml:Class>

<daml:Class rdf:ID="Fall">
  <rdfs:comment>Fall season, including January.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Season"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <Fall rdf:ID="E-">
      <rdfs:label>Fall</rdfs:label>
    </Fall>
  </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="Spring">
  <rdfs:comment>Spring season, including June.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Season"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <Spring rdf:ID="F-">
      <rdfs:label>Spring</rdfs:label>
    </Spring>
  </daml:oneOf>
</daml:Class>

<!-- ********************** MODULE ********************** -->

<daml:Class rdf:ID="Module">
  <rdfs:label>Module</rdfs:label>
  <rdfs:comment>A period of time in a week.</rdfs:comment>
</daml:Class>

<daml:Class rdf:ID="M1A">
  <rdfs:label>1A</rdfs:label>
  <rdfs:comment>Monday 8:00 - 12:00.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Module"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <M1A rdf:ID="m-1A">
      <rdfs:label>1A</rdfs:label>
    </M1A>
  </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="M2A">
  <rdfs:label>2A</rdfs:label>
  <rdfs:comment>Monday 13:00 - 17:00.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Module"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <M2A rdf:ID="m-2A">
      <rdfs:label>2A</rdfs:label>
    </M2A>
  </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="M3A">
  <rdfs:label>3A</rdfs:label>
  <rdfs:comment>Tuesday 8:00 - 12:00.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Module"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <M3A rdf:ID="m-3A">
```

```
      <rdfs:label>3A</rdfs:label>
    </M3A>
  </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="M4A">
  <rdfs:label>4A</rdfs:label>
  <rdfs:comment>Tuesday 12:00 - 17:00.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Module"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <M4A rdf:ID="m-4A">
      <rdfs:label>4A</rdfs:label>
    </M4A>
  </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="M5A">
  <rdfs:label>5A</rdfs:label>
  <rdfs:comment>Wednesday 8:00 - 12:00.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Module"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <M5A rdf:ID="m-5A">
      <rdfs:label>5A</rdfs:label>
    </M5A>
  </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="M1B">
  <rdfs:label>1B</rdfs:label>
  <rdfs:comment>Thursday 12:00 - 17:00.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Module"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <M1B rdf:ID="m-1B">
      <rdfs:label>1B</rdfs:label>
    </M1B>
  </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="M2B">
  <rdfs:label>2B</rdfs:label>
  <rdfs:comment>Thursday 8:00 - 12:00.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Module"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <M2B rdf:ID="m-2B">
      <rdfs:label>2B</rdfs:label>
    </M2B>
  </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="M3B">
  <rdfs:label>3B</rdfs:label>
  <rdfs:comment>Friday 12:00 - 17:00.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Module"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <M3B rdf:ID="m-3B">
      <rdfs:label>3B</rdfs:label>
    </M3B>
  </daml:oneOf>
</daml:Class>
```

```
<daml:Class rdf:ID="M4B">
  <rdfs:label>4B</rdfs:label>
  <rdfs:comment>Friday 8:00 - 12:00.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Module"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <M4B rdf:ID="m-4B">
      <rdfs:label>4B</rdfs:label>
    </M4B>
  </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="M5B">
  <rdfs:label>5B</rdfs:label>
  <rdfs:comment>Wednesday 12:00 - 17:00.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Module"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <M5B rdf:ID="m-5B">
      <rdfs:label>5B</rdfs:label>
    </M5B>
  </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="M3-week">
  <rdfs:label>3-week</rdfs:label>
  <rdfs:comment>3-week period</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Module"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <M3-week rdf:ID="m-3week">
      <rdfs:label>3-week</rdfs:label>
    </M3-week>
  </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="Other">
  <rdfs:label>Other</rdfs:label>
  <rdfs:comment>Period outside any of the other periods.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Module"/>
  <daml:oneOf rdf:parseType="daml:collection">
    <Other rdf:ID="m-other">
      <rdfs:label>other</rdfs:label>
    </Other>
  </daml:oneOf>
</daml:Class>

<!-- ********************** SCHEDULE ********************** -->

<daml:Class rdf:ID="Schedule">
  <rdfs:label>Schedule</rdfs:label>
  <rdfs:comment>
    The time of the week and season a course is taught at.
  </rdfs:comment>
  <rdfs:subClassOf>
    <daml:Class>
      <daml:intersectionOf rdf:parseType="daml:collection">
        <daml:Restriction daml:cardinality="1">
          <daml:onProperty rdf:resource="#inSeason"/>
        </daml:Restriction>
        <daml:Restriction daml:cardinality="1">
          <daml:onProperty rdf:resource="#inModule"/>
        </daml:Restriction>
      </daml:intersectionOf>
    </daml:Class>
  </rdfs:subClassOf>
```

```
<daml:oneOf rdf:parseType="daml:collection">
  <Schedule rdf:ID="F-1A">
    <rdfs:label>F-1A</rdfs:label>
    <inSeason rdf:resource="#F-"/>
    <inModule rdf:resource="#m-1A"/>
  </Schedule>
  <Schedule rdf:ID="F-2A">
    <rdfs:label>F-2A</rdfs:label>
    <inSeason rdf:resource="#F-"/>
    <inModule rdf:resource="#m-2A"/>
  </Schedule>
  <Schedule rdf:ID="F-3A">
    <rdfs:label>F-3A</rdfs:label>
    <inSeason rdf:resource="#F-"/>
    <inModule rdf:resource="#m-3A"/>
  </Schedule>
  <Schedule rdf:ID="F-4A">
    <rdfs:label>F-4A</rdfs:label>
    <inSeason rdf:resource="#F-"/>
    <inModule rdf:resource="#m-4A"/>
  </Schedule>
  <Schedule rdf:ID="F-5A">
    <rdfs:label>F-5A</rdfs:label>
    <inSeason rdf:resource="#F-"/>
    <inModule rdf:resource="#m-5A"/>
  </Schedule>
  <Schedule rdf:ID="F-1B">
    <rdfs:label>F-1B</rdfs:label>
    <inSeason rdf:resource="#F-"/>
    <inModule rdf:resource="#m-1B"/>
  </Schedule>
  <Schedule rdf:ID="F-2B">
    <rdfs:label>F-2B</rdfs:label>
    <inSeason rdf:resource="#F-"/>
    <inModule rdf:resource="#m-2B"/>
  </Schedule>
  <Schedule rdf:ID="F-3B">
    <rdfs:label>F-3B</rdfs:label>
    <inSeason rdf:resource="#F-"/>
    <inModule rdf:resource="#m-3B"/>
  </Schedule>
  <Schedule rdf:ID="F-4B">
    <rdfs:label>F-4B</rdfs:label>
    <inSeason rdf:resource="#F-"/>
    <inModule rdf:resource="#m-4B"/>
  </Schedule>
  <Schedule rdf:ID="F-5B">
    <rdfs:label>F-5B</rdfs:label>
    <inSeason rdf:resource="#F-"/>
    <inModule rdf:resource="#m-5B"/>
  </Schedule>
  <Schedule rdf:ID="F-3week">
    <rdfs:label>June</rdfs:label>
    <inSeason rdf:resource="#F-"/>
    <inModule rdf:resource="#m-3week"/>
  </Schedule>
  <Schedule rdf:ID="F-other">
    <rdfs:label>F-other</rdfs:label>
    <inSeason rdf:resource="#F-"/>
    <inModule rdf:resource="#m-other"/>
  </Schedule>
  <Schedule rdf:ID="E-1A">
    <rdfs:label>E-1A</rdfs:label>
```

```
      <inSeason rdf:resource="#E-"/>
      <inModule rdf:resource="#m-1A"/>
    </Schedule>
    <Schedule rdf:ID="E-2A">
      <rdfs:label>E-1A</rdfs:label>
      <inSeason rdf:resource="#E-"/>
      <inModule rdf:resource="#m-2A"/>
    </Schedule>
    <Schedule rdf:ID="E-3A">
      <rdfs:label>E-3A</rdfs:label>
      <inSeason rdf:resource="#E-"/>
      <inModule rdf:resource="#m-3A"/>
    </Schedule>
    <Schedule rdf:ID="E-4A">
      <rdfs:label>E-4A</rdfs:label>
      <inSeason rdf:resource="#E-"/>
      <inModule rdf:resource="#m-4A"/>
    </Schedule>
    <Schedule rdf:ID="E-5A">
      <rdfs:label>E-5A</rdfs:label>
      <inSeason rdf:resource="#E-"/>
      <inModule rdf:resource="#m-5A"/>
    </Schedule>
    <Schedule rdf:ID="E-1B">
      <rdfs:label>E-1B</rdfs:label>
      <inSeason rdf:resource="#E-"/>
      <inModule rdf:resource="#m-1B"/>
    </Schedule>
    <Schedule rdf:ID="E-2B">
      <rdfs:label>E-2B</rdfs:label>
      <inSeason rdf:resource="#E-"/>
      <inModule rdf:resource="#m-2B"/>
    </Schedule>
    <Schedule rdf:ID="E-3B">
      <rdfs:label>E-3B</rdfs:label>
      <inSeason rdf:resource="#E-"/>
      <inModule rdf:resource="#m-3B"/>
    </Schedule>
    <Schedule rdf:ID="E-4B">
      <rdfs:label>E-4B</rdfs:label>
      <inSeason rdf:resource="#E-"/>
      <inModule rdf:resource="#m-4B"/>
    </Schedule>
    <Schedule rdf:ID="E-5B">
      <rdfs:label>E-5B</rdfs:label>
      <inSeason rdf:resource="#E-"/>
      <inModule rdf:resource="#m-5B"/>
    </Schedule>
    <Schedule rdf:ID="E-3week">
      <rdfs:label>January</rdfs:label>
      <inSeason rdf:resource="#E-"/>
      <inModule rdf:resource="#m-3week"/>
    </Schedule>
    <Schedule rdf:ID="E-other">
      <rdfs:label>E-other</rdfs:label>
      <inSeason rdf:resource="#E-"/>
      <inModule rdf:resource="#m-other"/>
    </Schedule>
  </daml:oneOf>
</daml:Class>
```

```
<!-- ********************** TOPICS ********************** -->

<daml:Class rdf:ID="Topic">
  <rdfs:label>Topic</rdfs:label>
  <rdfs:comment>A topic from ACM CCS.</rdfs:comment>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#A"/>
  </daml:unionOf>
</daml:Class>

<daml:Class rdf:ID="A.0.1">
  <rdfs:label>Biographies/autobiographies</rdfs:label>
  <rdfs:subClassOf rdf:resource="#A.0"/>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class><daml:oneOf rdf:parseType="daml:collection">
      <A.0.1 rdf:ID="topic-A.0.1"/>
    </daml:oneOf></daml:Class>
  </daml:unionOf>
</daml:Class>

<daml:Class rdf:ID="A.0.2">
  <rdfs:label>Conference proceedings</rdfs:label>
  <rdfs:subClassOf rdf:resource="#A.0"/>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class><daml:oneOf rdf:parseType="daml:collection">
      <A.0.2 rdf:ID="topic-A.0.2"/>
    </daml:oneOf></daml:Class>
  </daml:unionOf>
</daml:Class>

<daml:Class rdf:ID="A.0.3">
  <rdfs:label>General literary works (e.g., fiction, plays)</rdfs:label>
  <rdfs:subClassOf rdf:resource="#A.0"/>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class><daml:oneOf rdf:parseType="daml:collection">
      <A.0.3 rdf:ID="topic-A.0.3"/>
    </daml:oneOf></daml:Class>
  </daml:unionOf>
</daml:Class>

<daml:Class rdf:ID="A.0">
  <rdfs:label>GENERAL</rdfs:label>
  <rdfs:subClassOf rdf:resource="#A"/>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class><daml:oneOf rdf:parseType="daml:collection">
      <A.0 rdf:ID="topic-A.0"/>
    </daml:oneOf></daml:Class>
    <daml:Class rdf:about="#A.0.1"/>
    <daml:Class rdf:about="#A.0.2"/>
    <daml:Class rdf:about="#A.0.3"/>
  </daml:unionOf>
</daml:Class>

<daml:Class rdf:ID="A.1">
  <rdfs:label>INTRODUCTORY AND SURVEY</rdfs:label>
  <rdfs:subClassOf rdf:resource="#A"/>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class><daml:oneOf rdf:parseType="daml:collection">
      <A.1 rdf:ID="topic-A.1"/>
    </daml:oneOf></daml:Class>
  </daml:unionOf>
</daml:Class>
```

```
<daml:Class rdf:ID="A.2">
  <rdfs:label>
    REFERENCE (e.g., dictionaries, encyclopedias, glossaries)
  </rdfs:label>
  <rdfs:subClassOf rdf:resource="#A"/>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class><daml:oneOf rdf:parseType="daml:collection">
      <A.2 rdf:ID="topic-A.2"/>
    </daml:oneOf></daml:Class>
  </daml:unionOf>
</daml:Class>

<daml:Class rdf:ID="A.m">
  <rdfs:label>MISCELLANEOUS</rdfs:label>
  <rdfs:subClassOf rdf:resource="#A"/>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class><daml:oneOf rdf:parseType="daml:collection">
      <A.m rdf:ID="topic-A.m"/>
    </daml:oneOf></daml:Class>
  </daml:unionOf>
</daml:Class>

<daml:Class rdf:ID="A">
  <rdfs:label>General Literature</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Topic"/>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class><daml:oneOf rdf:parseType="daml:collection">
      <A rdf:ID="topic-A"/>
    </daml:oneOf></daml:Class>
    <daml:Class rdf:about="#A.0"/>
    <daml:Class rdf:about="#A.1"/>
    <daml:Class rdf:about="#A.2"/>
    <daml:Class rdf:about="#A.m"/>
  </daml:unionOf>
</daml:Class>

<!-- ********************** COURSE ********************** -->

<AnyCourse rdf:ID='Dummy'>
  <isTaughtIn rdf:resource='#EN'/>
</AnyCourse>

<daml:Class rdf:ID="AnyCourse">
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Course"/>
    <daml:Class>
      <daml:oneOf rdf:parseType="daml:collection">
        <AnyCourse rdf:about='#Dummy'/>
      </daml:oneOf>
    </daml:Class>
  </daml:unionOf>
</daml:Class>

<daml:Class rdf:ID="Course">
  <rdfs:label>Course</rdfs:label>
  <rdfs:subClassOf rdf:resource="#AnyCourse"/>
  <rdfs:subClassOf>
    <daml:Class>
      <daml:intersectionOf rdf:parseType="daml:collection">
        <daml:Restriction daml:cardinality="1">
          <daml:onProperty rdf:resource="#isTaughtIn"/>
        </daml:Restriction>
        <daml:Restriction daml:cardinality="1">
```

```
          <daml:onProperty rdf:resource="#courseName"/>
        </daml:Restriction>
        <daml:Restriction daml:minCardinality="1">
          <daml:onProperty rdf:resource="#isTaughtAt"/>
        </daml:Restriction>
      </daml:intersectionOf>
    </daml:Class>
  </rdfs:subClassOf>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#LinePackageCourse"/>
    <daml:Class rdf:about="#CoreCourse"/>
    <daml:Class rdf:about="#MasterCourse"/>
    <daml:Class>
      <daml:oneOf rdf:parseType="daml:collection">
        <AnyCourse rdf:about='#Dummy'/>
      </daml:oneOf>
    </daml:Class>
  </daml:unionOf>
</daml:Class>

<daml:Class rdf:ID="LinePackageCourse">
  <rdfs:subClassOf rdf:resource="#Course"/>
  <rdfs:label>Line package course</rdfs:label>
  <rdfs:comment>Course included in a line package.</rdfs:comment>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#LinePackageCoreCourse"/>
    <daml:Class rdf:about="#LinePackageAMSCourse"/>
    <daml:Class rdf:about="#MandatoryLinePackageCourse"/>
    <daml:Class>
      <daml:oneOf rdf:parseType="daml:collection">
        <AnyCourse rdf:about='#Dummy'/>
      </daml:oneOf>
    </daml:Class>
  </daml:unionOf>
</daml:Class>

<daml:Class rdf:ID="CoreCourse">
  <rdfs:subClassOf rdf:resource="#Course"/>
  <rdfs:label>Core course</rdfs:label>
  <rdfs:comment>Mandatory course for complete master students.</rdfs:comment>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#LinePackageCoreCourse"/>
    <daml:Class rdf:about="#MandatoryCoreCourse"/>
    <daml:Class>
      <daml:oneOf rdf:parseType="daml:collection">
        <AnyCourse rdf:about='#Dummy'/>
      </daml:oneOf>
    </daml:Class>
  </daml:unionOf>
</daml:Class>

<daml:Class rdf:ID="MasterCourse">
  <rdfs:subClassOf rdf:resource="#Course"/>
  <rdfs:label>Master course</rdfs:label>
  <rdfs:comment>Course at master level.</rdfs:comment>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#ExoticCourse"/>
    <daml:Class rdf:about="#AMSCourse"/>
    <daml:Class rdf:about="#MandatoryCourse"/>
    <daml:Class rdf:about="#SpecializationCourse"/>
    <daml:Class rdf:about="#InternationalMasterCourse"/>
    <daml:Class>
      <daml:oneOf rdf:parseType="daml:collection">
```

```
        <AnyCourse rdf:about='#Dummy'/>
      </daml:oneOf>
    </daml:Class>
  </daml:unionOf>
</daml:Class>

<daml:Class rdf:ID="ExoticCourse">
  <rdfs:subClassOf rdf:resource="#MasterCourse"/>
  <rdfs:label>Humanistic course</rdfs:label>
  <rdfs:comment>Exotic course.</rdfs:comment>
  <daml:sameClassAs>
    <daml:Class>
      <daml:oneOf rdf:parseType="daml:collection">
        <AnyCourse rdf:about='#Dummy'/>
      </daml:oneOf>
    </daml:Class>
  </daml:sameClassAs>
</daml:Class>

<daml:Class rdf:ID="AMSCourse">
  <rdfs:subClassOf rdf:resource="#MasterCourse"/>
  <rdfs:label>AMS course</rdfs:label>
  <rdfs:comment>Course in Work, Environment and Society.</rdfs:comment>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#LinePackageAMSCourse"/>
    <daml:Class>
      <daml:oneOf rdf:parseType="daml:collection">
        <AnyCourse rdf:about='#Dummy'/>
      </daml:oneOf>
    </daml:Class>
  </daml:unionOf>
</daml:Class>

<daml:Class rdf:ID="MandatoryCourse">
  <rdfs:subClassOf rdf:resource="#MasterCourse"/>
  <rdfs:label>Mandatory course</rdfs:label>
  <rdfs:comment>Mandatory course for master students.</rdfs:comment>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#MandatoryLinePackageCourse"/>
    <daml:Class rdf:about="#MandatoryCoreCourse"/>
    <daml:Class>
      <daml:oneOf rdf:parseType="daml:collection">
        <AnyCourse rdf:about='#Dummy'/>
      </daml:oneOf>
    </daml:Class>
  </daml:unionOf>
</daml:Class>

<daml:Class rdf:ID="SpecializationCourse">
  <rdfs:subClassOf rdf:resource="#MasterCourse"/>
  <rdfs:label>Specialization course</rdfs:label>
  <rdfs:comment>
    Course at master level that gives a specialization.
  </rdfs:comment>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#InternationalSpecializationCourse"/>
    <daml:Class>
      <daml:oneOf rdf:parseType="daml:collection">
        <AnyCourse rdf:about='#Dummy'/>
      </daml:oneOf>
    </daml:Class>
  </daml:unionOf>
</daml:Class>
```

```
<daml:Class rdf:ID="InternationalMasterCourse">
  <rdfs:subClassOf rdf:resource="#MasterCourse"/>
  <rdfs:label>International master course</rdfs:label>
  <rdfs:comment>
    Course at master level for international students.
  </rdfs:comment>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#InternationalSpecializationCourse"/>
    <daml:Class>
      <daml:oneOf rdf:parseType="daml:collection">
        <AnyCourse rdf:about='#Dummy'/>
      </daml:oneOf>
    </daml:Class>
  </daml:unionOf>
</daml:Class>

<daml:Class rdf:ID="InternationalSpecializationCourse">
  <rdfs:subClassOf rdf:resource="#SpecializationCourse"/>
  <rdfs:subClassOf rdf:resource="#InternationalMasterCourse"/>
  <rdfs:label>International specialization course</rdfs:label>
  <rdfs:comment>
    Specialization course for international students.
  </rdfs:comment>
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#InternationalMandatoryCourse"/>
    <daml:Class>
      <daml:oneOf rdf:parseType="daml:collection">
        <AnyCourse rdf:about='#Dummy'/>
      </daml:oneOf>
    </daml:Class>
  </daml:unionOf>
</daml:Class>

<daml:Class rdf:ID="InternationalMandatoryCourse">
  <rdfs:subClassOf rdf:resource="#InternationalSpecializationCourse"/>
  <rdfs:label>International mandatory course</rdfs:label>
  <rdfs:comment>Mandatory course for international students.</rdfs:comment>
  <daml:sameClassAs>
    <daml:Class>
      <daml:oneOf rdf:parseType="daml:collection">
        <AnyCourse rdf:about='#Dummy'/>
      </daml:oneOf>
    </daml:Class>
  </daml:sameClassAs>
</daml:Class>

<daml:Class rdf:ID="LinePackageCoreCourse">
  <rdfs:subClassOf rdf:resource="#LinePackageCourse"/>
  <rdfs:subClassOf rdf:resource="#CoreCourse"/>
  <rdfs:label>Line package core course</rdfs:label>
  <rdfs:comment>
    Mandatory course for complete master students that is included in a line
    package.
  </rdfs:comment>
  <daml:sameClassAs>
    <daml:Class>
      <daml:oneOf rdf:parseType="daml:collection">
        <AnyCourse rdf:about='#Dummy'/>
      </daml:oneOf>
    </daml:Class>
  </daml:sameClassAs>
</daml:Class>
```

```
  <daml:Class rdf:ID="LinePackageAMSCourse">
    <rdfs:subClassOf rdf:resource="#LinePackageCourse"/>
    <rdfs:subClassOf rdf:resource="#AMSCourse"/>
    <rdfs:label>Line package AMS course</rdfs:label>
    <rdfs:comment>
      Course in Work, Environment and Society that is included in a line
      package.
    </rdfs:comment>
    <daml:sameClassAs>
      <daml:Class>
        <daml:oneOf rdf:parseType="daml:collection">
          <AnyCourse rdf:about='#Dummy'/>
        </daml:oneOf>
      </daml:Class>
    </daml:sameClassAs>
  </daml:Class>

  <daml:Class rdf:ID="MandatoryLinePackageCourse">
    <rdfs:subClassOf rdf:resource="#LinePackageCourse"/>
    <rdfs:subClassOf rdf:resource="#MandatoryCourse"/>
    <rdfs:label>Mandatory line package course</rdfs:label>
    <rdfs:comment>
      Mandatory course for master students that is included in a line
      package.
    </rdfs:comment>
    <daml:sameClassAs>
      <daml:Class>
        <daml:oneOf rdf:parseType="daml:collection">
          <AnyCourse rdf:about='#Dummy'/>
        </daml:oneOf>
      </daml:Class>
    </daml:sameClassAs>
  </daml:Class>

  <daml:Class rdf:ID="MandatoryCoreCourse">
    <rdfs:subClassOf rdf:resource="#CoreCourse"/>
    <rdfs:subClassOf rdf:resource="#MandatoryCourse"/>
    <rdfs:label>Mandatory core course</rdfs:label>
    <rdfs:comment>
      Mandatory course for master students that is also a core
      course.
    </rdfs:comment>
    <daml:sameClassAs>
      <daml:Class>
        <daml:oneOf rdf:parseType="daml:collection">
          <AnyCourse rdf:about='#Dummy'/>
        </daml:oneOf>
      </daml:Class>
    </daml:sameClassAs>
  </daml:Class>

</rdf:RDF>
```

APPENDIX B – THE NATIONALITY EXAMPLE FILE

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns     ="http://www.student.dtu.dk/~c960516/example.daml#"
>
  <daml:Ontology rdf:about="">
    <daml:imports rdf:resource="http://www.daml.org/2001/03/daml+oil"/>
  </daml:Ontology>

  <!-- ********************** INSTANCE ********************** -->

    <Person rdf:ID="Peter">
      <has_nationality rdf:resource="#german"/>
      <born_in rdf:resource="#china"/>
    </Person>

  <!-- ********************** ONTOLOGY ********************** -->
    <daml:Class rdf:ID="EuropeanPerson">
      <daml:intersectionOf rdf:parseType="daml:collection">
        <daml:Restriction daml:cardinality="1">
          <daml:onProperty rdf:resource="#has_nationality"/>
          <daml:toClass rdf:resource="#EuropeanNationality"/>
        </daml:Restriction>
        <daml:Restriction daml:cardinality="1">
          <daml:onProperty rdf:resource="#born_in"/>
          <daml:toClass rdf:resource="#EuropeanCountry"/>
        </daml:Restriction>
      </daml:intersectionOf>
    </daml:Class>

    <daml:Class rdf:ID="OtherPerson">
      <daml:intersectionOf rdf:parseType="daml:collection">
        <daml:Restriction daml:cardinality="1">
          <daml:onProperty rdf:resource="#has_nationality"/>
          <daml:toClass rdf:resource="#OtherNationality"/>
        </daml:Restriction>
        <daml:Restriction daml:cardinality="1">
          <daml:onProperty rdf:resource="#born_in"/>
          <daml:toClass rdf:resource="#OtherCountry"/>
        </daml:Restriction>
      </daml:intersectionOf>
    </daml:Class>

    <daml:ObjectProperty rdf:ID="has_nationality">
      <rdfs:domain rdf:resource="#Person"/>
      <rdfs:range rdf:resource="#Nationality"/>
    </daml:ObjectProperty>
```

```
    <daml:ObjectProperty rdf:ID="born_in">
      <rdfs:domain rdf:resource="#Person"/>
      <rdfs:range rdf:resource="#Country"/>
    </daml:ObjectProperty>

    <daml:Class rdf:ID="Nationality"/>
    <daml:Class rdf:ID="Country"/>

    <daml:Class rdf:ID="Person">
      <daml:disjointUnionOf rdf:parseType="daml:collection">
        <daml:Class rdf:about="#EuropeanPerson"/>
        <daml:Class rdf:about="#OtherPerson"/>
      </daml:disjointUnionOf>
    </daml:Class>

  <!-- ************** SUFFICIENT CONCEPT DEFINITIONS ************* -->

    <daml:Class rdf:ID="EuropeanNationality">
      <daml:subClassOf rdf:resource="#Nationality"/>
      <daml:oneOf rdf:parseType="daml:collection">
      <EuropeanNationality rdf:ID="german"/>
      </daml:oneOf>
    </daml:Class>

    <daml:Class rdf:ID="OtherNationality">
      <daml:subClassOf rdf:resource="#Nationality"/>
      <daml:oneOf rdf:parseType="daml:collection">
      <OtherNationality rdf:ID="chinese"/>
      </daml:oneOf>
    </daml:Class>

    <daml:Class rdf:ID="EuropeanCountry">
      <daml:subClassOf rdf:resource="#Country"/>
      <daml:oneOf rdf:parseType="daml:collection">
      <EuropeanCountry rdf:ID="germany"/>
      </daml:oneOf>
    </daml:Class>

    <daml:Class rdf:ID="OtherCountry">
      <daml:subClassOf rdf:resource="#Country"/>
      <daml:oneOf rdf:parseType="daml:collection">
      <OtherCountry rdf:ID="china"/>
      </daml:oneOf>
    </daml:Class>

</rdf:RDF>
```

APPENDIX C – THE COLOR OPTION EXAMPLE FILE

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns     ="http://www.student.dtu.dk/~c960516/example.daml#"
>
  <daml:Ontology rdf:about="">
    <daml:imports rdf:resource="http://www.daml.org/2001/03/daml+oil"/>
  </daml:Ontology>

  <!-- ********************** INSTANCE ********************** -->

    <Uniform rdf:ID="uniform1">
      <has_color rdf:resource="#blue"/>
      <has_part rdf:resource="#pants"/>
      <has_part rdf:resource="#blouse"/>
    </Uniform>

  <!-- ********************** ONTOLOGY ********************** -->

    <daml:ObjectProperty rdf:ID="has_color">
      <rdfs:domain rdf:resource="#Uniform"/>
      <rdfs:range rdf:resource="#Color"/>
    </daml:ObjectProperty>

    <daml:ObjectProperty rdf:ID="has_part">
      <rdfs:domain rdf:resource="#Uniform"/>
      <rdfs:range rdf:resource="#Item"/>
    </daml:ObjectProperty>

    <daml:ObjectProperty rdf:ID="has_color_option">
      <rdfs:domain rdf:resource="#Item"/>
      <rdfs:range rdf:resource="#Color"/>
    </daml:ObjectProperty>

    <daml:Class rdf:ID="Color"/>

    <daml:Class rdf:ID="LightColor">
      <daml:subClassOf rdf:resource="#Color"/>
      <daml:oneOf rdf:parseType="daml:collection">
      <Color rdf:ID="red"/>
      </daml:oneOf>
    </daml:Class>

    <daml:Class rdf:ID="DarkColor">
      <daml:subClassOf rdf:resource="#Color"/>
      <daml:oneOf rdf:parseType="daml:collection">
      <Color rdf:ID="blue"/>
      </daml:oneOf>
    </daml:Class>
```

```
<daml:Class rdf:ID="Item"/>

<daml:Class rdf:ID="Pants">
  <daml:subClassOf rdf:resource="#Item"/>
  <daml:oneOf rdf:parseType="daml:collection">
  <Item rdf:ID="pants">
    <has_color_option rdf:resource="#red"/>
    <has_color_option rdf:resource="#blue"/>
  </Item>
  </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="Blouse">
  <daml:subClassOf rdf:resource="#Item"/>
  <daml:oneOf rdf:parseType="daml:collection">
  <Item rdf:ID="blouse">
    <has_color_option rdf:resource="#red"/>
  </Item>
  </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="Uniform">
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#DarkUniform"/>
    <daml:Class rdf:about="#LightUniform"/>
  </daml:unionOf>
</daml:Class>

<daml:Class rdf:ID="DarkUniform">
  <daml:subClassOf rdf:resource="#Uniform"/>
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#has_color"/>
      <daml:toClass rdf:resource="#DarkColor"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#has_part"/>
      <daml:toClass>
        <daml:Restriction>
          <daml:onProperty rdf:resource="#has_color_option"/>
          <daml:hasClass rdf:resource="#DarkColor"/>
        </daml:Restriction>
      </daml:toClass>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>

<daml:Class rdf:ID="LightUniform">
  <daml:subClassOf rdf:resource="#Uniform"/>
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#has_color"/>
      <daml:toClass rdf:resource="#LightColor"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#has_part"/>
```

```
          <daml:toClass>
            <daml:Restriction>
              <daml:onProperty rdf:resource="#has_color_option"/>
              <daml:hasClass rdf:resource="#LightColor"/>
            </daml:Restriction>
          </daml:toClass>
        </daml:Restriction>
      </daml:intersectionOf>
    </daml:Class>

</rdf:RDF>
```