



Vrije Universiteit Brussel

FACULTEIT INGENIEURSWETENSCHAPPEN &
WETENSCHAPPEN

DEPARTMENT OF COMPUTER SCIENCE AND APPLIED COMPUTER
SCIENCE



Software Design Description

Software Engineering

Nicolas Carraggi, Youri Coppens, Christophe Gaethofs, Pieter Meire-
sone, Sam Van den Vonder, Fernando Suarez, Tim Witters

Academiejaar 2013-2014



Versiegeschiedenis

Tabel 1: Versiegeschiedenis

Versie	Datum	Auteur(s)	Commentaar
0.1	17/11/2013	Youri Coppens	Initiële versie
0.2	10/12/2013	Youri Coppens	Opstart inhoud document
1.0	13/12/2013	Youri Coppens	Oplevering Iteratie 1
2.0	4/03/2014	Youri Coppens	Oplevering Iteratie 2
3.0	18/04/2014	Youri Coppens	Oplevering Iteratie 3
4.0	16/05/2014	Youri Coppens	Finale oplevering

Inhoudsopgave

Versiegeschiedenis	ii
1 Introductie	1
1.1 Doel en scope	1
1.2 Acroniemen	1
1.3 Overzicht	1
2 Systeemarchitectuur	2
2.1 Model	2
2.2 Gebruikte technologie	2
3 Viewpoints	4
3.1 Context	4
3.2 Logica	6
3.2.1 Lagen	6
3.2.2 Componenten	7
3.3 Data	7
3.4 Structuur	10
3.4.1 Leslokalen en vakken	10
3.4.2 Logging	11
3.4.3 Scheduler	12
3.5 Algoritmes	16
3.5.1 Scheduling	16

Lijst van figuren

2.1	Een voorstelling van de werking van een systeem gebruikmakende van een MVC architectuur	2
3.1	Use case diagram met focus op de verschillende actoren	4
3.2	Use case diagram met focus op de actor administrator	5
3.3	Use case diagram met focus op de actoren professor en assistant	5
3.4	Use case diagram met focus op de actor student	6
3.5	EER diagram	8
3.6	Eerste deel van het database schema	9
3.7	Tweede deel van het database schema	9
3.8	Derde deel van het database schema	9
3.9	Vierde deel van het database schema	10
3.10	UML klassediagram voor leslokalen en vakken	11
3.11	UML klassediagram voor scheduling	14
3.12	UML klassediagram voor conflictdetectie te realiseren	15
3.13	UML klassediagram van de verschillende ConstraintViolations	16

Lijst van tabellen

1	Versiegeschiedenis	ii
1.1	Acroniemen	1

Hoofdstuk 1

Introductie

1.1 Doel en scope

Dit document beschrijft de softwarearchitectuur en het design van de CalZone webapplicatie. CalZone is een webapplicatie om lessenroosters te plannen en bezichtigen. Dit document zal gebruikt worden door de programmeurs, de designers en de testers van dit project als naslagwerk en documentatie om de werking van het systeem te vatten. Hierdoor is het aangeraden dit document te raadplegen indien men de applicatie wil uitbreiden en aanpassen.

1.2 Acroniemen

Tabel 1.1: Acroniemen

API	Application Programming Interface
DAO	Data Access Object
DB	Database
EE	Enterprise Edition
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JPA	Java Persistence API
JSP	Java Server Page
IDE	Integrated Development Environment
MVC	Model View Controller
SDD	Software Design Description
SDK	Software Development Kit
SRS	Software Requirements Specification
XML	Extensible Markup Language

1.3 Overzicht

Dit document volgt de IEEE Std 1016-2006TM standaard voor het opstellen van Software Design Descriptions. Dit document is beïnvloed door de requirements beschreven in de Software Requirements Specification van dit project.[5]

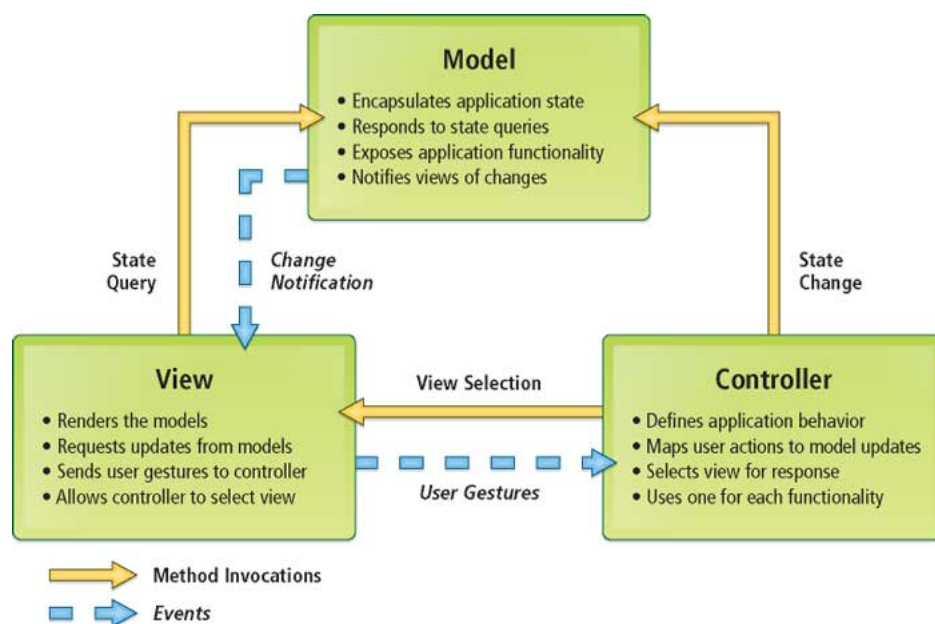
In hoofdstuk 2 wordt de gebruikte systeemarchitectuur toegelicht en in hoofdstuk 3 worden verschillende 'design viewpoints' besproken die relevant zijn voor het systeem.

Hoofdstuk 2

Systemeemarchitectuur

2.1 Model

CalZone is een webapplicatie. Gebruikers van het systeem bezoeken de applicatie via hun webbrowser. Deze browser kan de browser op hun computer zijn of de webbrowser op hun smartphone. Doorheen de ontwikkeling van het project wordt de focus vooral gericht op Android toestellen en browsers op deze toestellen wat betreft het mobiele aspect van de applicatie. CalZone heeft als architectuur gekozen voor het MVC-patroon.[12]



Figuur 2.1: Een voorstelling van de werking van een systeem gebruikmakende van een MVC architectuur

2.2 Gebruikte technologie

De programmeertaal die gebruikt wordt op logisch niveau is Java. In Java wordt gebruik gemaakt van het Spring MVC framework[8, 7] voor het ontwikkelen van CalZone. De IDE waarin geprogrammeerd wordt, is de meest recente versie van 'Eclipse Classic' met volgende uitbreidingen:

- De gehele collectie 'Web, XML, Java EE and OSGi Enterprise Development'

- Spring Tool Suite (uit de Eclipse Marketplace)
- De gehele collectie 'Maven Integration for Eclipse'

Het uitvoeren van de applicatie wordt mogelijk gemaakt door middel van Apache Maven[3] en Apache Tomcat[4]. De gebruikte databank voor de back-end is MySQL. De connecties vanuit het logisch niveau van het systeem naar de databank verlopen via Hibernate[9]. Voor de view wordt gebruik gemaakt van JSP's, een technologie om dynamisch webpagina's te genereren, .

De applicatie moet in staat zijn om lessenroosters te maken. Om dit doel te verwezenlijken wordt gebruik gemaakt van OptaPlanner [10]. Meer uitleg over OptaPlanner en het maken van lessenroosters staat beschreven in 3.2.2, 3.4.3 en 3.5.1.

Hoofdstuk 3

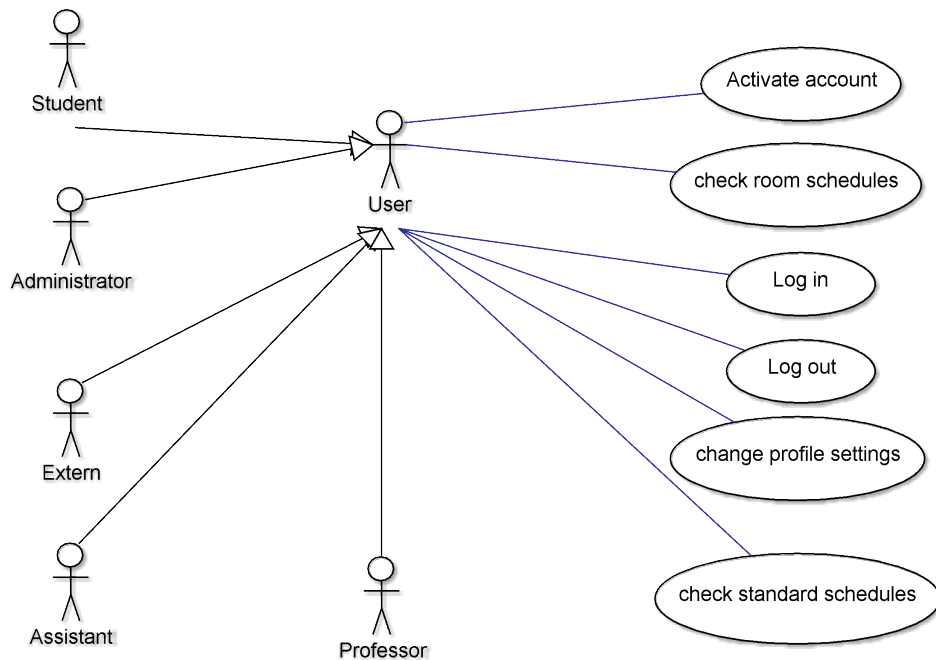
Viewpoints

In dit hoofdstuk worden *design viewpoints* besproken die relevant zijn voor het systeem.

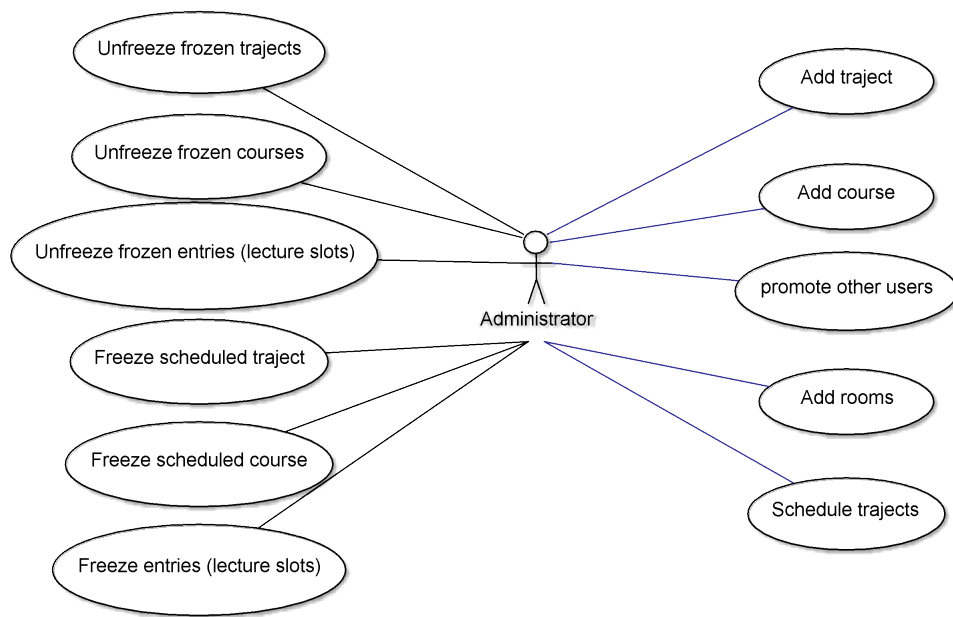
3.1 Context

De gebruikers van het systeem zijn onder te verdelen in 5 categorieën: externen, studenten, professoren, assistenten en programmabeheerders. Het design van CalZone houdt rekening met de functionaliteiten die specifiek aan deze gebruikers zijn toegekend zoals beschreven in het SRS[5] van dit project.

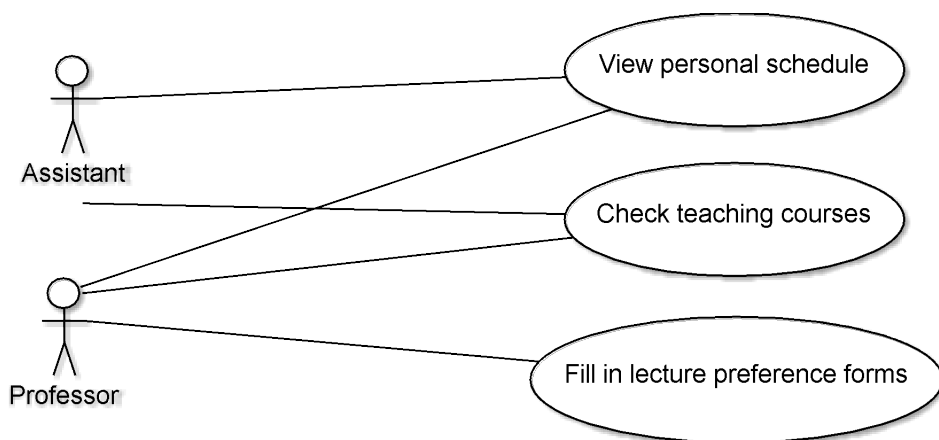
De functionaliteiten die iedere actor in het systeem bezit, worden geïllustreerd in de use case diagrammen 3.1, 3.2, 3.3 en 3.4.



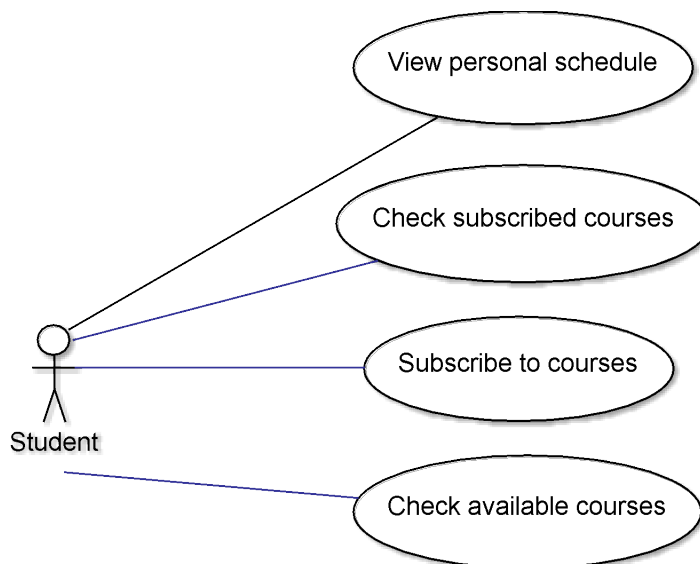
Figuur 3.1: Use case diagram met focus op de verschillende actoren



Figuur 3.2: Use case diagram met focus op de actor administrator



Figuur 3.3: Use case diagram met focus op de actoren professor en assistant



Figuur 3.4: Use case diagram met focus op de actor student

3.2 Logica

In deze sectie worden verschillende modules en packages besproken die deel uitmaken van het huidige logische niveau van het systeem.

3.2.1 Lagen

Database Repositories

CalZone maakt gebruik van een relationele databank, MySQL, als back-end voor dataopslag. Om informatie vanuit deze databank te lezen en er naartoe te schrijven zijn er enkele packages voorzien die een abstractie bieden voor het openen en sluiten van de connectie met de databank en het sturen van queries naar de databank. Dit is de eerste package die bijdraagt tot deze abstractielaag.

Service

Dit is de tweede package die abstractie biedt tussen de databank en het logische niveau van het systeem. Het abstractieniveau ten opzichte van de data-laag stijgt dankzij deze service-laag. De klassen in deze package voorzien dus functies om specifieke data vanuit de databank op te halen. Deze klassen voeren queries uit en laden de gevonden data in de daarvoor voorziene objecten.

Model

Deze package bevat alle klassen die data vanuit de databank bezitten. De klassen in deze package representeren dus de informatie waarmee het systeem functionaliteiten zal voorzien naar de gebruiker toe.

3.2.2 Componenten

Controllers

Deze klassen zijn verantwoordelijk om de HTTP-requests te verwerken. Deze klassen zorgen dus voor een propagatie van (functionaliteits)verzoeken van de front-end naar de back-end. Eveneens zorgen deze klassen ook voor een propagatie van data van de back-end naar de front-end. Dit laatste uit zich in content op de webpagina's. Met andere woorden zorgen de controllers dus voor de mogelijke views en voorzien dus de mogelijkheid om de functionaliteiten opgesomd in de use case diagrammen van sectie 3.1 uit te voeren.

Validators

Binnenin het systeem dient sommige data gecontroleerd te worden op geldigheid. Deze validator-klassen zijn verantwoordelijk om geldigheid van bepaalde informatie te controleren en aan te geven indien deze info niet geldig is.

Scheduler

In deze package zitten de klassen die bijdragen tot het maken en formuleren van lessenroosters. Ook vind je hier de configuraties voor OptaPlanner. Er is een configuratiebestand voor de uiteindelijke AI agent die lessenroosters zal trachten te maken op basis van verschillende constraints. OptaPlanner bezit een benchmarking systeem opdat men verschillende configuraties kan testen op verscheidene datasets. Meer hierover is te vinden in het Test Document van dit project. Deze package bezit ook een file met constraint regels. Deze file bevat code in de programmeertaal Drools[13].

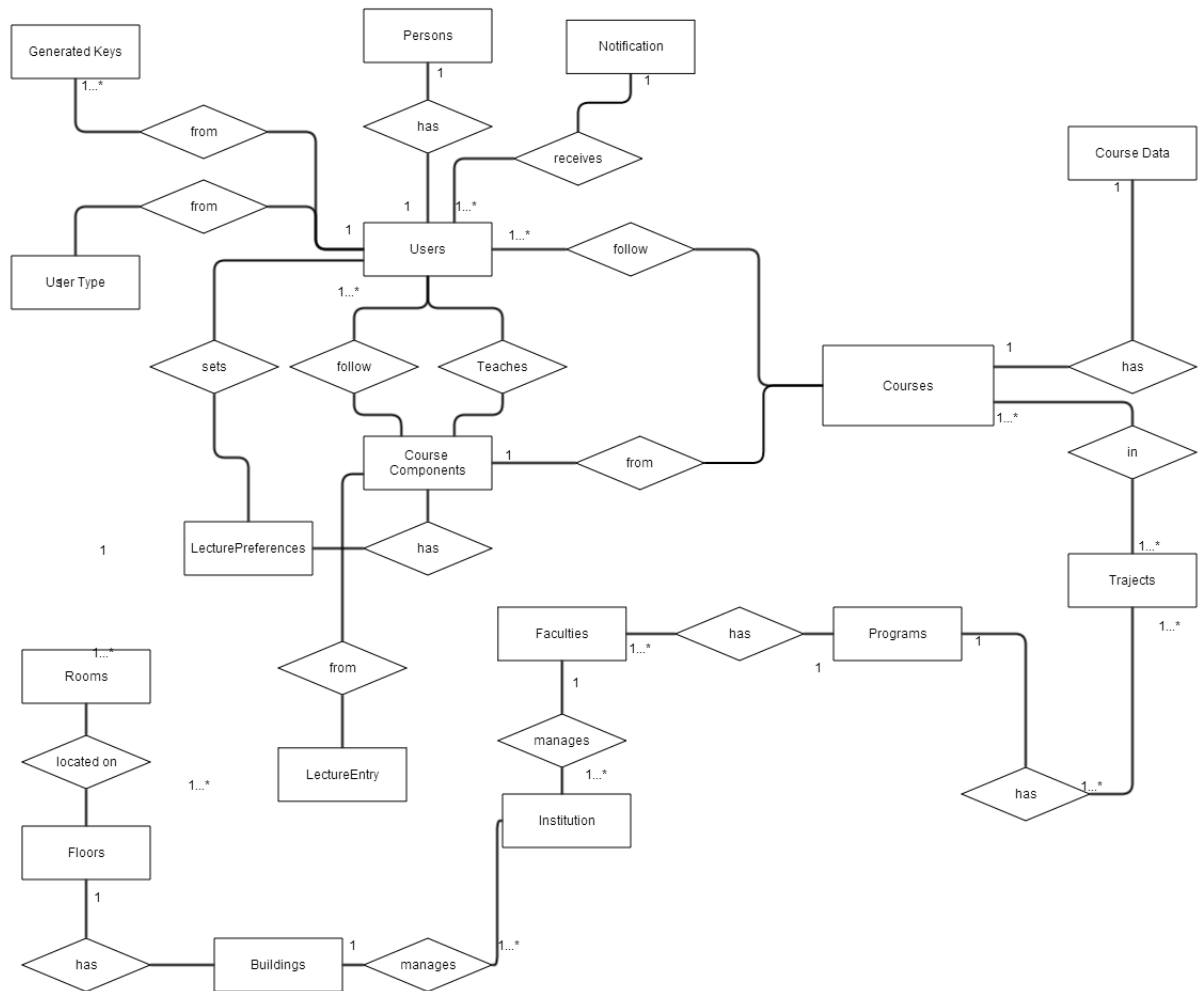
Exception

Zelfgemaakte Exceptions worden verzameld in deze package.

3.3 Data

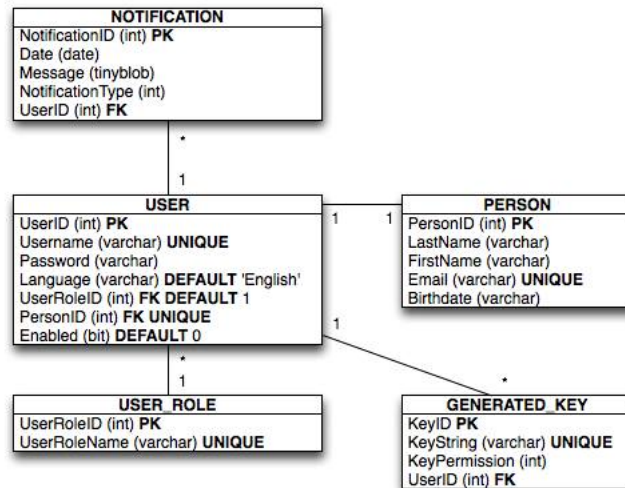
Deze sectie bevat het EER-model voor de databank. Eveneens bezit het ook een voorstelling van het databaseschema die dit model realiseert.

Figuur 3.5 toont het EER-model van de databank. Om overzicht te bewaren worden de attributen die iedere entiteit bezit niet weergegeven in dit model.

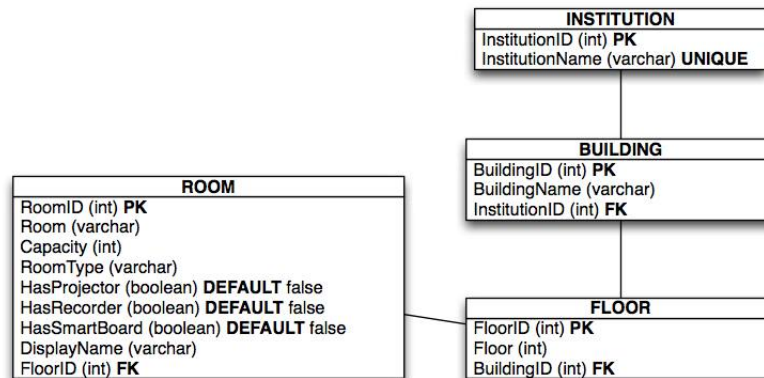


Figuur 3.5: EER diagram

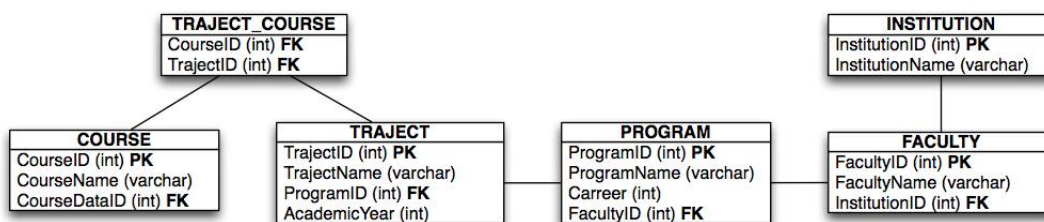
Figuren 3.6, 3.7, 3.8 en 3.9 tonen de huidige status van het database schema.



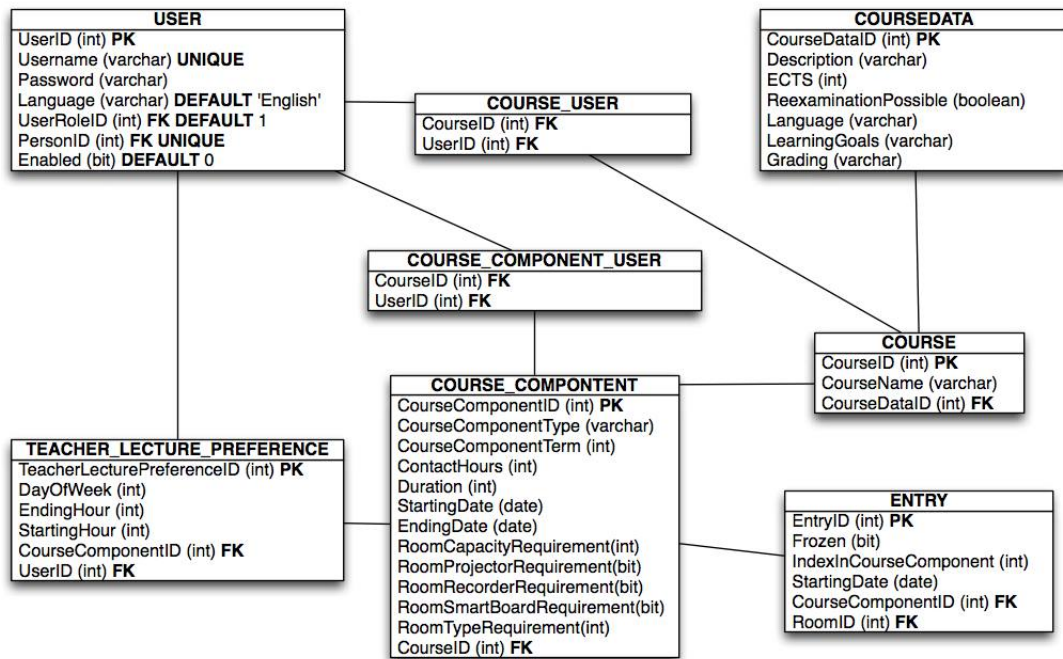
Figuur 3.6: Eerste deel van het database schema



Figuur 3.7: Tweede deel van het database schema



Figuur 3.8: Derde deel van het database schema



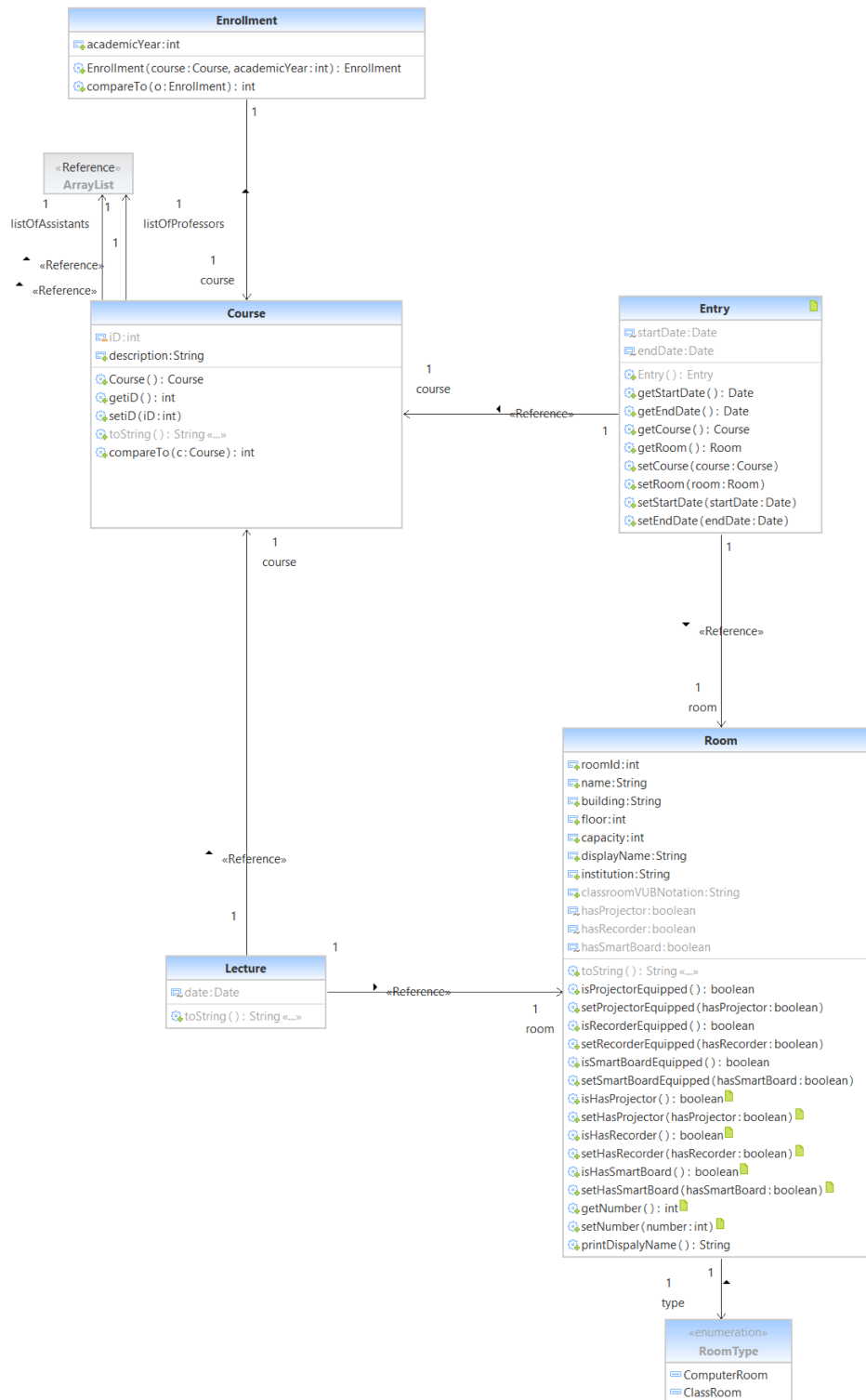
Figuur 3.9: Vierde deel van het database schema

3.4 Structuur

Deze sectie beschrijft de structuur van belangrijke en interessante onderdelen van het systeem. Deze beschrijvingen worden al dan niet ondersteund met een bijbehorend diagram indien dit van toepassing is.

3.4.1 Leslokalen en vakken

Volgend UML diagram beschrijft de structuur van de verscheidene klassen betreffende leslokalen en lessen die met elkaar gerelateerd zijn op het logisch niveau van het systeem. Deze klassen bezitten data vanuit de databank die nodig zullen zijn om lessenroosters en examenroosters te plannen.



Figuur 3.10: UML klassediagram voor leslokalen en vakken

3.4.2 Logging

Als logging framework werd gekozen voor SLF4J[6]. Dit framework is echter een API die het maken van logs versimpelt voor de programmeur en vereist dus nog een geconfigureerde backbone. Als

backbone werd gekozen voor logback [2].

In loggingsystemen kan men verschillende soorten logberichten produceren. Het systeem zorgt er dan voor dat deze berichten op de juiste plaats terecht komen. De belangrijkste soorten berichten zijn: info, warning, error en debug. In wat volgt wordt de configuratie van het loggingsysteem uitgeklaard.

Alle logberichten worden op de standaard output getoond. Verder worden er specifieke logs opgeslagen naar bestanden. Er zijn 2 folders voorzien voor logs: eentje voor debug logs en een folder met daarin een algemene log file. Deze algemene log file bevat log berichten vanaf het niveau 'INFO', wat betekent dat het dus INFO, WARNING en ERROR logs bewaart. Voor de debug logs wordt iedere dag een nieuwe file aangemaakt omdat het systeem een massa aan debug logs kan produceren op een dag.

Voor het in productie gaan van de applicatie is het sterk aangeraden de debug logs uit te schakelen om de lengte van de totale hoeveelheid logs drastisch te krimpen.

3.4.3 Scheduler

Scheduling

Volgend UML klassediagram beschrijft de structuur van de klassen betreffende lessenroosters en het maken van zulke roosters. Figuur 3.11 beschrijft hoe het planningsprobleem in onze applicatie geformuleerd wordt. Het framework OptaPlanner[10] maakt gebruik van deze klassen door middel van annotaties om lessenroosters te maken. Nu volgt een simpele beschrijving van de structuur.

Een *Schedule* is een planning. Zo een planning bestaat uit een sequentie van *Entries*. Een *Entry* kan men vergelijken met een afspraak in een agenda. In ons systeem wordt zo dus gevormd door een klaslokaal (Room), een vakonderdeel (CourseComponent) en een tijdstip (Date). Een initialisatieklasse, *SchedulerInitializer*, voorziet de beschikbare datums en tijdstippen. Het verwezenlijken van een planning wordt opgelost door de klasse *SchedulerSolver*. Deze klasse zal gebruik maken van OptaPlanner om een *Schedule* te genereren. Impliciet zal de *SchedulerSolver* gebruik maken van de klassen *SchedulerHelper*, *EntryDifficultyComparator*, *RoomStrengthComparator*, *DateStrengthComparator* in een poging een zo goed mogelijk lessenrooster te bekomen. De klasse *MovableSelectionEntry* zorgt ervoor dat OptaPlanner weet welke entries verschoven mogen worden door de solver en welke niet. Er is dus een notie van bevriezen van entries aanwezig in de schedule. Meer uitleg over hoe OptaPlanner werkt en geconfigureerd is wordt beschreven in 3.5.1.

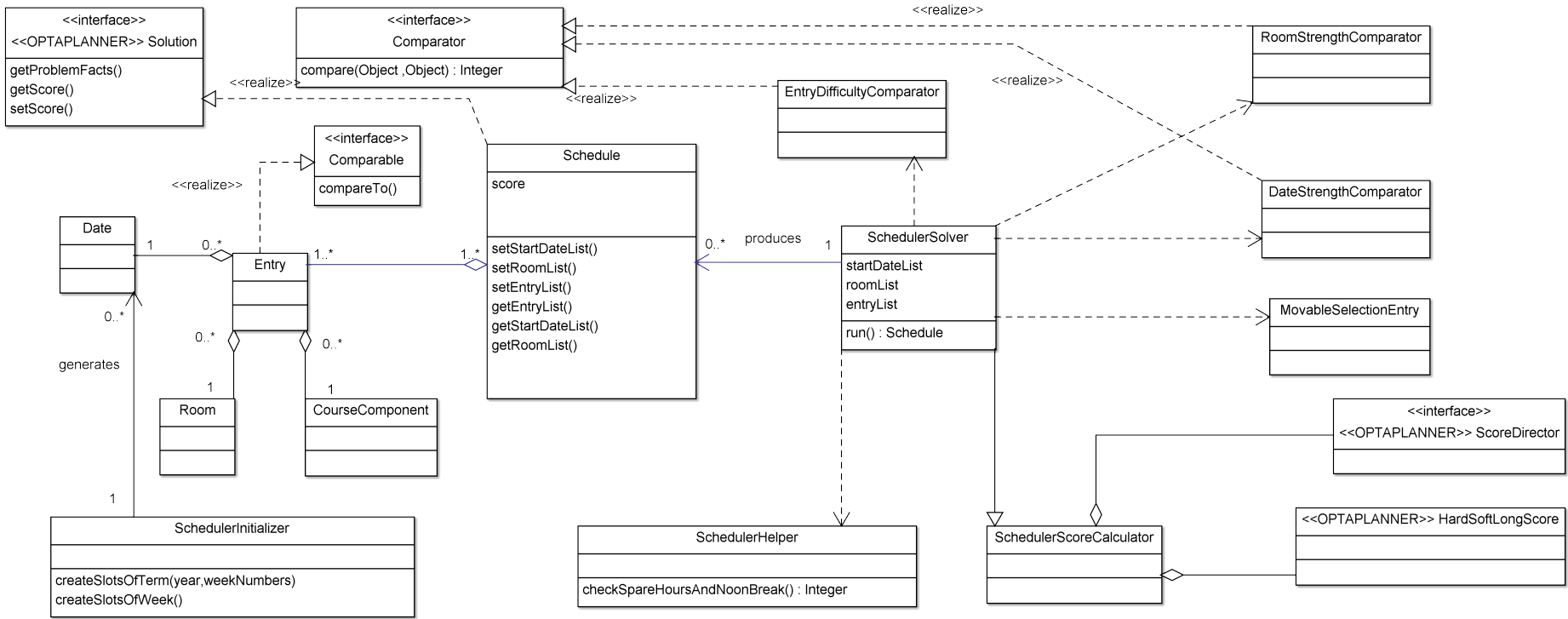
De klasse *SchedulerSolver* is een subtype van de klasse *SchedulerScoreCalculator*. Deze klasse is verantwoordelijk het berekenen van de score van een *Schedule* in zijn huidige staat. Via deze klasse kan men ook conflicten gaan detecteren.

Alle klassen in het klassenmodel, op de klasse *HardSoftLongScore* na, diende zelf geschreven te worden en geannoteerd te worden zodat OptaPlanner deze klassen kon herkennen en gebruiken. Bij het maken van deze klassen diende bepaalde interfaces van OptaPlanner geïmplementeerd te worden, *Solution*¹ en *ScoreDirector*² en werd de klasse *HardSoftLongScore* gebruikt. In het klassediagram

¹<https://docs.jboss.org/drools/release/6.0.1.Final/optaplanner-javadoc/org/optaplanner/core/impl/solution/Solution.html>

²<https://docs.jboss.org/drools/release/6.0.1.Final/optaplanner-javadoc/org/optaplanner/core/impl/score/director/ScoreDirector.html>

werden deze aangeduid met de annotatie $\langle\langle OPTAPLANNER \rangle\rangle$.



Figuur 3.11: UML klassediagram voor scheduling

Conflictdetectie in roosters

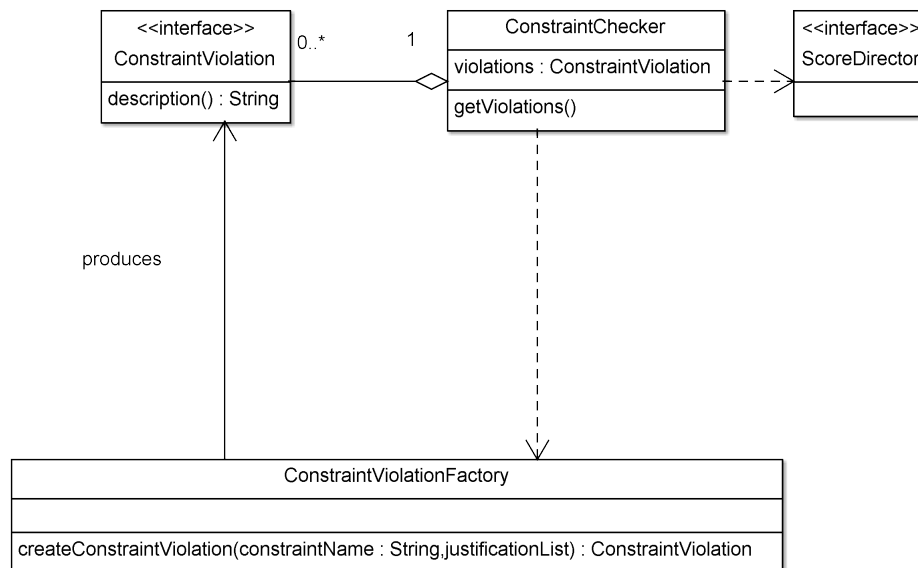
De interface *ScoreDirector*² van OptaPlanner is verantwoordelijk om de score van de staat van een planningoplossing te berekenen. Naast het berekenen van deze score, is de klasse ook in staat om te verklaren hoe hij aan die score komt. De *ScoreDirector* voorziet een *justificationList*, een lineaire structuur die informatie bevat over welke regels uit de Drools file afgevuurd zijn. Aangezien deze regels declaratief zijn en er dus gebruik wordt gemaakt van pattern matching, staat er ook in de *justificationList* welk patroon (de verzameling objecten die voldoet aan de match) er gevonden werd.

Dit stelt ons in staat deze informatie te parsen en te vertalen naar ons eigen design om conflicten in een schedule te melden aan de programmabeheerders en professoren die gebruik maken van CalZone. Figuren 3.12 en 3.13 beschrijven de structuur van de objecten die dit tot stand brengen.

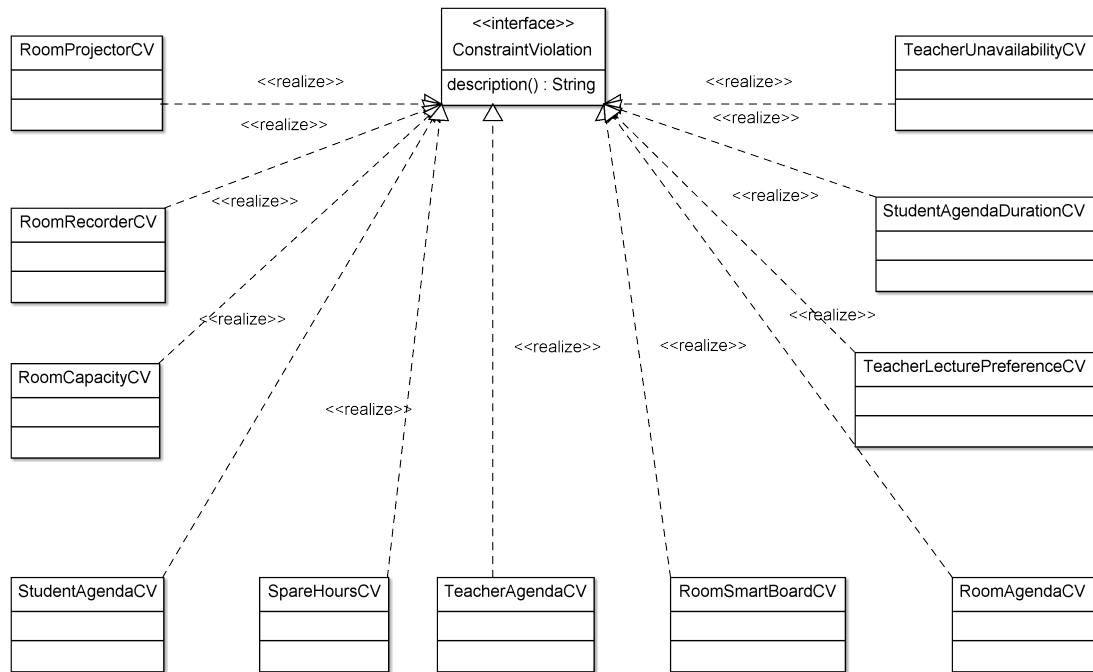
Een klasse *ConstraintChecker* zal de *justificationList* van de *ScoreDirector* van OptaPlanner analyseren en een lijst van schendingen teruggeven. Zo een schending wordt voorgesteld door de interface *ConstraintViolation*. Deze interface bezit een methode 'description'. Deze methode geeft een string terug die het gehele conflict uitlegt. De klasse *ConstraintChecker* is dus het aanspreekpunt voor de front-end om conflicten in een bestaand rooster weer te geven aan gebruikers.

Voor iedere constraint is er een klasse voorzien die de interface *ConstraintViolation* implementeert en dus een beschrijving ontwikkelt afhankelijk van wat voor constraint geschonden werd.

Het genereren van deze classes voor de *ConstraintChecker* wordt afgehandeld door een andere klasse: *ConstraintViolationFactory*.



Figuur 3.12: UML klassediagram voor conflictdetectie te realiseren



Figuur 3.13: UML klassediagram van de verschillende ConstraintViolations

3.5 Algoritmes

3.5.1 Scheduling

Voor het maken van lessenroosters wordt er gebruik gemaakt van de library OptaPlanner[10]. Dit is een constraint solver library met focus op het oplossen van planningen. Om gebruik te maken van OptaPlanner dien je eerst je planningsprobleem te modelleren. Dit werd besproken in 3.4.3.

De klasse `SchedulerSolver` representeert de agent die een lessenrooster zal construeren vanuit een lijst van lokalen, datums en vakonderdelen. Dit doet het door middel van local search algoritmes en construction heuristics die door de programmeur geconfigureerd dienen te worden. Initieel begint men de agent met een naïeve opstelling van een `Schedule`. De agent berekent dan de score van deze opstelling. Deze score wordt berekend door na te trekken welke constraints er geschonden werden. Deze constraints dienen door de programmeur ontwikkeld te worden. Indien deze score nog niet optimaal is, zal de agent een verschuiving toepassen in het rooster. Hiermee wordt getracht de score te verbeteren, met andere woorden minder constraints te schenden. De agent kiest de 'beste' verschuiving en het algoritme herhaalt zich.

De exacte manier van het maken van deze verschuivingen en het selecteren van de beste ver-

schuiving in het rooster hangt af van local search algoritmes die gebruikt worden voor het scheduleren. Om de configuratie zo optimaal mogelijk te maken bezit OptaPlanner een benchmarking systeem waarvoor de configuratie terug te vinden is in de package 'Scheduler'.

De huidige configuratie van SchedulerSolver gebruik van een construction heuristic, first fit decreasing, gevolgd door een local search algoritme, simulated annealing. Als scoresysteem wordt er gebruik gemaakt van een 2-laagse score, beter gekend als Hard-Soft score. Men kan de constraints dus opdelen in 2 groepen. Elke constraint bezorgt zijn score en de scores van constraints in dezelfde groep wordt bij mekaar opgeteld.

Volgende hard constraints werden ontwikkeld:

- Leraren mogen/kunnen geen 2 verschillende lessen op eenzelfde tijdstip geven.
- In een traject mogen geen lessen overlappen.
- Een les dient door te gaan in een lokaal met voldoende ruimte.
- Er kunnen niet meerdere lessen tegelijk doorgaan in eenzelfde lokaal
- Lessen worden gegeven in de daarvoor voorziene periode/weken.
- Een les wordt gehouden in een lokaal met de juiste accommodaties voor die les (projectoren, computers...)
- Lessen moeten wekelijks op hetzelfde moment gepland worden.

Volgende soft constraints werden ontwikkeld:

- Lessen worden gepland rekening houdend met de voorkeuren die de lesgever heeft opgegeven voor een vak.
- Lessen worden niet gepland op momenten dat de lesgever niet kan. (Opgegeven door de lesgever).
- Er worden geen 2 sessies (hoorcolleges of werkcolleges) van hetzelfde vak achter elkaar gepland.
- Een student heeft maximaal 9 uur les op een dag.
- Springuren worden vermeden, maar men tracht toch een middagpauze te behouden.
- Er worden geen lessen gepland die eindigen na 19u.

Deze constraints werden geschreven in de declaratieve taal Drools[13].

Bibliografie

- [1] Ben Alex, Luke Taylor en Rob Winch. *Spring Security Reference Documentation*. Versie 3.2.2.RELEASE. 2013. URL: <http://docs.spring.io/spring-security/site/docs/3.2.2.RELEASE/reference/htmlsingle/>.
- [2] Carl Harris Ceci Gülcü Sébastien Pennec. *The logback manual*. Versie 1.1.2. 2013. URL: <http://logback.qos.ch/manual/index.html>.
- [3] The Apache Software Foundation. *Apache Maven 3.x*. Versie 3.1.1. 2013. URL: <https://maven.apache.org/ref/3.1.1/>.
- [4] The Apache Software Foundation. *Apache Tomcat 7 Documentation*. Versie 7.047. 2013. URL: <https://tomcat.apache.org/tomcat-7.0-doc/index.html>.
- [5] Fernando Suarez Groen, Tim Witters en Youri Coppens. *CalZone: Software Requirements Specification*. Versie 1.0. 2013.
- [6] Ceci Gülcü. *SLF4J user manual*. Versie 1.7.6. 2013. URL: <http://www.slf4j.org/manual.html>.
- [7] Rod Johnson e.a. *Spring Framework Reference Documentation*. Versie 4.0.2.RELEASE. 2013. URL: <http://docs.spring.io/spring/docs/4.0.2.RELEASE/spring-framework-reference/htmlsingle/>.
- [8] Pivotal Software. *Spring*. 2013. URL: <http://spring.io/>.
- [9] The Hibernate Team en The JBoss Visual Design Team. *Hibernate Reference Documentation*. Versie 4.3.5.Final. 2014. URL: <https://docs.jboss.org/hibernate/orm/4.3/manual/en-US/html/>.
- [10] The OptaPlanner Team. *OptaPlanner User Guide*. Versie 6.0.1.Final. 2013. URL: https://docs.jboss.org/drools/release/6.0.1.Final/optaplanner-docs/html_single/index.html.
- [11] The OptaPlanner Team. *OptaPlanner core 6.0.1.Final API*. Versie 6.0.1.Final. 2013. URL: <https://docs.jboss.org/drools/release/6.0.1.Final/optaplanner-javadoc/index.html>.
- [12] Onbekende auteurs. *Model-view-controller*. 2013. URL: <https://en.wikipedia.org/wiki/Modelviewcontroller>.
- [13] The JBoss Drools team. *Drools Documentation*. Versie 6.0.1.Final. 2013. URL: https://docs.jboss.org/drools/release/6.0.1.Final/drools-docs/html_single/index.html.