Software Design Descriptions voor Schedule-Generator

Matthias Caenepeel Adam Cooman Alexander De Cock
Zjef Van de Poel

23 februari 2011 Versie 0.1

Aanpassingsgeschiedenis

. 23/2/2011 versie 0.1: Aanmaak document

. 27/2/2011 versie 0.2: Toevoeging delen over interfaces

. 28/2/2011 versie 0.3: Toevoeging hoofdstuk over algoritme en Logical

Inhoudsopgave

1 Introduction

TODO

1.1 Purpose

1.2 Scope

1.3 Context

1.4 Summary

2 Body

2.1 Identified stakeholders and design concerns

2.2 Design viewpoint 1: Compositie

ALEXANDER samenwerking van alle onderdelen + uitleg

2.2.1 Design concerns

Alle verschillende onderdelen opsommen en hun taken beschrijven.

2.2.2 Design elements

Entiteiten, verbanden en attribuut.

2.2.3 Function attributes

Legt het verband tussen de entiteiten. (da's eigenlijk de transferfunctie ma Rahnild weet toch wa da is; dus laat zitten jong)

2.2.4 Subordinates attributes

Entiteiten in entiteiten (overal UML als exampletaal)

2.3 Design viewpoint 2: Logical

2.3.1 Design concerns

Om de lessenrooster op te stellen en uit te lezen zijn er verschillende klasses nodig die alle verschillende participerende objecten voorstellen.

2.3.2 Elementen

2.3.2.1 Entiteiten

- Lessen structuur
  - *Course*: Les die gevolgd kan worden
  - *Subcourse*: Onderdeel van een les; bv hoorcollege, labo of oefeningenles
  - *Program*: Verzameling van lessen die in een pakket zitten. Bijvoorbeeld 1e bachelor ingenieurswetenschappen
- Gebouwen strucuur
  - *Building*: Gebouw
  - *Room*: Lokaal in een gebouw waar les gegeven kan worden
  - *Hardware*: Materiaal beschikbaar in een lokaal
- Personen structuur

    - *Student*: Persoon die lessen kan volgen
    - *Educator*: Persoon die lessen kan geven; zowel professoren, docenten en assistenten
- Andere
    - *Faculty*: Faculteit van de universiteit

2.3.2.2 Relaties

- Lessen structuur

  Elke *Course* bestaat uit een of meerdere *SubCourses* die in opgeteld het gehele vak weergeven.
  Courses zelf worden gegroepeerd in een *Program*.

- Gebouwen structuur

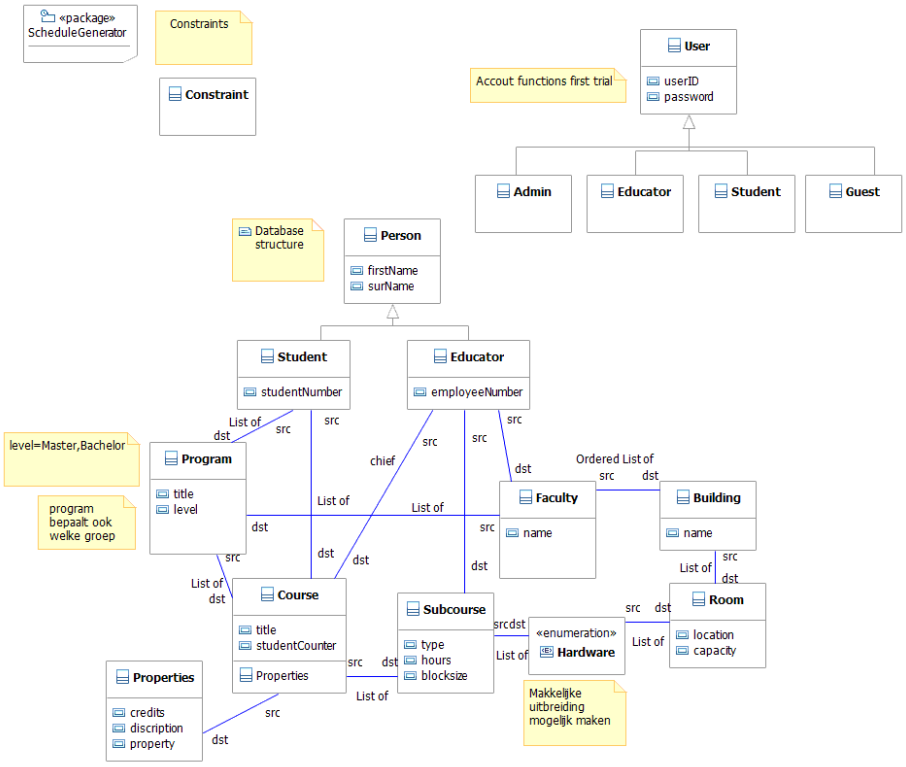
  Een gebouw beschikt over een lijst met *Rooms*, die op zich een lijst met beschikbare *Hardware* bijhoudt.
  Een gebouw kan ingedeeld worden onder een faculteit.

- Personen

  Elke *Student* is ingeschreven in een *Program* en een lijst met afzonderlijke *Courses* die ze volgen.
  Een *Educator* beschikt over een lijst *Courses* en *SubCourses* die ze geven.

2.3.2.3 UML Diagrammma

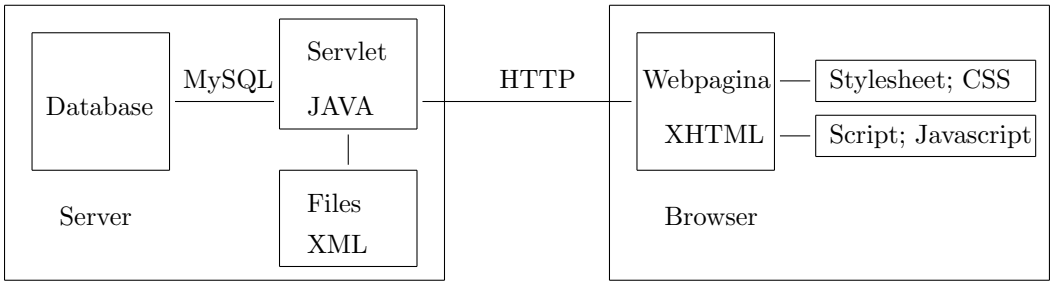
De klasse waarbij de pijl vertrekt erft van de klasse waar de pijl aankomt. De klasse waarbij er src staat zit in een lijst in de klasse waarbij er dst staat. Zo zit er bijvoorbeeld een lijst van educators in de klasse faculty.
Deze structuur zal nog worden uitgebreid met methode die de rooster opstellen en dan hieruit vertrekkende de fitnessvalue bepalen. De bovenstaande structuur zal met andere woorden nog worden uitgebreid.



Figuur 1: UML Diagramma

2.4 Design viewpoint 3: Interfaces

2.4.1 Design concerns, algemeen



Figuur 2: Overzicht van de verschillende delen van het programma

In de structuur van ons programma bestaan verschillende elementen met verschillende taken. Deze moeten met elkaar interageren volgens het bovenstaande schema. De lijnen tussen de verschillende blokken noemen we interfaces en zullen in het volgende deel van het Software Design Document besproken worden. Vooraleer daarmee te beginnen een kort overzicht van de taken die elk blok uit het diagramma uitvoert.

Webpagina, XHTML
In het XHTML bestand wordt de inhoud van de webpagina geplaatst.

Stylesheet, CSS
In het stylesheet wordt de lay-out van de webpagina beschreven.

Script, Javascript
In het script word beschreven wanneer welk deel van de webpagina weergegeven wordt. We gebruiken tabbladen om de functionaliteiten die een gebruiker krijgt duidelijk weer te geven. Ht beheer van die tabbladen gebeurt met het javascript "tabber"dat ontwikkeld werd door derden;

Servlet, JAVA
De servlets runnen op tomcat en genereren de XHTML code, afhankelijk van de eigenschappen van de gebruiker.

Database
In de database wordt de informatie van gebruikers en de kalender opgeslaan.

Files
De files bevatten wijzigbare parameters van het programma. Ze zijn in een XML bestand opgeslaan.

Het schema is geen volledig correcte weergave van de werkelijkheid omdat het stylesheet en het script zich ook op de server bevinden en door de browser opgehaald worden van de server via een HTTP protocol. Om volledig correct te zijn zouden die twee onderdelen van het schema zich dus ook op de server moeten bevinden. Om alles overzichtelijk te houden heeft de auteur beslist om ze bij de browser te plaatsen, omdat de browser het ophalen van de server voorziet en niet de gebruiker.

2.4.2 Interface: XHTML - CSS voor Layout

2.4.2.1 Initialiseren

Om het .css bestand aan de XHTML pagina te linken moet in de header van de XHTML code het volgende voorzien worden

```
<link rel="stylesheet" href="style.css" type="text/css">
```

Hierin zijn de volgende elementen te herkennen:

`rel="stylesheet"`: Duidelijk maken dat de link een link naar een stylesheet is. Deze tag verandert niet

`type="text/css"`: Duidelijk maken dat de stylesheet in css code geschreven is. Deze tag verandert ook niet

`href="style.css"`: De naam van het css bestand. Als die zich op een andere locatie bevindt dan de XHTML pagina moet het pad naar die map hierin toegevoegd worden

2.4.2.2 Gebruiken

In de XHTML code moet niet veel toegevoegd worden om de css op te roepen, enkel een id tag op de volgende manier om onderscheid te maken tussen verschillende gedefiniëerde stijlen in de css code. Als voorbeeld wordt het toevoegen van een id aan een rij van een tabel gegeven om aan te tonen hoe dit moet.

```
<tr id="MainBottom">
```

De naam van het id staat tussen de aanhalingstekens.

In het CSS bestand kan de code voor de stijl van hetzelfde id geschreven worden door gebruik te maken van hekje. Als voorbeeld de CSS code die de stijl beschrijfd van de tabelrij uit het vorige voorbeeld.

```
tr#MainBottom {
height:300px;
}
```

2.4.3 Interface: XHTML - Javascript voor tabbladen

2.4.3.1 Initialiseren

Het oproepen van het javascript bestand gebeurt in de header van het XHTML bestand dat de inhoud van de site beschrijft met de volgende code.

```
<script type="text/javascript" src="tabber-minimized.js"></script>
```

Hierin zijn de volgende elementen te herkennen:

> `type="text/javascript"`: Duidelijk maken dat het javascript is. Deze tag verandert niet
>
> `src="tabber-minimized.js"`: De naam van het javascript bestand. Als die zich op een andere locatie bevindt dan de HTML pagina moet het pad naar die map hierin toegevoegd worden

2.4.3.2 Gebruiken

Om dan een tabblad aan te maken en er inhoud in te plaatsen wordt gebruik gemaakt van div elementen in de XHTML code. dit gebeurd op de volgende manier:

```
<div class="tabber">
<div class="tabbertab" title="Tabblad 1">
Inhoud van het eerste tabblad
</div>
<div class="tabbertab" title="Tabblad 2">
Inhoud van het tweede tabblad
</div>
</div>
```

In de buitenste div staat het volgenden:

> `class="tabber"`: dit vertelt aan het javascript dat er tabbladen volgen. Het laat toe om geneste tabbladen te gebruiken

In de binnenste divs, één per tabblad:

> `class="tabbertab"`: dit vertelt aan het javascript dat er tabbladen volgen. Het laat toe om geneste tabbladen te gebruiken
>
> `title="Tabblad 1"`: In het title tag staat de titel van het tabblad die bovenaan weergegeven wordt

2.4.4 Interface: Javascript - CSS voor tabbladen

Het Javascript tabber dat gebruikt wordt bevat interne links naar het CSS bestand. De exacte werking van de interface is niet gekend omdat tabber een programma is dat geschreven is door derden. De nodige css code werd aan het stijlbestand toegevoegd zodat alles werk. Het is nu mogelijk om de layout van de tabbladen te definiëren in het stijlbestand. De nodige onderverdelingen die toegevoegd moeten worden zijn de volgende:

```
.tabberlive .tabbertabhide {}
.tabber {}
.tabberlive {}
ul.tabbernav{}
ul.tabbernav li {}
ul.tabbernav li a {}
ul.tabbernav li a:link {}
ul.tabbernav li a:visited {}
ul.tabbernav li a:hover{}
ul.tabbernav li.tabberactive a{}
ul.tabbernav li.tabberactive a:hover{}
.tabberlive .tabbertab {}
```

2.4.5 Interface: Browser - Server: HTTP

TODO

2.4.6 Interface: Database interface

2.4.6.1 Concerns

Wijzigbare parameters moeten opgeslagen worden in een file. Hiervoor wordt gebruik gemaakt van XML. Deze interface biedt de mogelijkheid om vanuit het programma eenvoudige XML bestanden uit te lezen en aan te maken.

2.4.6.2 Elementen

- Database
    - **Database**: Handle naar de database. Levert methodes om tabellen in te lezen, opzoekingen te doen, objecten weg te schrijven en dergelijke.
    - **Table**: Een enkele tabel uit de database
    - **Element**: Een element van een tabel
- Interface
    - **Databasable**: Inteface* die duidelijk maakt dat een object van een klasse die *Databasable* implementeert, in de database geschreven, of uit de database gelezen kan worden.
    - **InDatabase**: Annotation* die duidelijk maakt dat een bepaalde parameter van een object in de database bewaard moet worden.
    - **OutDatabase**: Annotation* die duidelijk maakt dat een bepaalde parameter van een object uit de database gelezen en naar het object gekopieerd moet worden.

*: Java terminologie

2.4.7 Interface: File interface

2.4.7.1 Concerns

Wijzigbare parameters moeten opgeslagen worden in een file. Hiervoor wordt gebruik gemaakt van XML. Deze interface biedt de mogelijkheid om vanuit het programma eenvoudige XML bestanden uit te lezen en aan te maken.

2.4.7.2 Elementen

- **XMLDocument**: Object verwijzende naar een XML bestand. Een XML-Document bevat een lijst van XMLElement
- **XMLElement**: Element van een XML bestand. Een element heeft een bepaalde waarde, of bevat zelf een aantal andere elementen.
  - **ElementWithValue**: Element met een waarde
  - **ElementWithChildren**: Element met lijst van andere elementen, zijn *Children.*(*Children* kunnen op hun beurt ook weer *ElementWith-Children* of *ElementWithValue* zijn)
- **XMLParser**: Levert methodes om een XML bestand, gecreerd met deze interface, weer in te laden naar een XMLDocument

2.4.7.3 Schema

2.5 Design viewpoint 4: Algoritme voor kalenderplanning

2.5.1 Design concerns

Het algoritme dient in staat te zijn om een lessenrooster te maken dat aan bepaalde voorwaarden (constraints) voldoet. Het is daarom belangrijk dat deze eerst bepaald worden. Men kan een indeling maken in deze voorwaarden. Zo zijn er de *fixed* constraints aan deze moeten zeker voldaan zijn, anders klopt het lessen rooster niet. *Hard constraints* deze moeten zo goed mogelijk vervuld zijn en dan zijn er nog *soft constraints* deze staan onderaan de ladder en hoeven dus niet noodzakelijk vervuld te zijn. Aan de hand van deze voorwaarden kan men dan nagaan hoe goed het lessenrooster is. Dit doet men aan de hand van een *fitnessvalue*. Zo krijgt een geplaatste les 3 punten als aan een *fixed constraint* voldaan is, 2 voor een hard en 1 voor een soft. Aangezien de *fixed constraints* zeker voldaan moeten zijn, moet een geplaatste les al zeker 12 punten hebben. Op die manier kan men het resultaat evalueren.

Fixed constraints

1. Er mag nooit meer dan 1 les gepland zijn in een leslokaal
2. Een docent (educator) kan niet meer dan 1 les geven
3. Een leslokaal (room) moet groot genoeg zijn voor het aantal studenten
4. Als een les bepaald materiaal vereist (zoals computer, laboratorium,...) moet hier aan worden tegemoet gekomen.

Hard constraints

1. Het lessenrooster van een student mag niet overlappen.
2. Er mag van een vak hoogstens 4 uur per dag gegeven worden.
3. Student mogen maximum 8 uur les per dag krijgen.

Soft constraints

1. De werkcolleges mogen pas na of gelijktijdig met de theoriecolleges beginnen.

Voor iedere week van het academisch jaar zal de lessenrooster een tabel bevatten voor elke student die bestaat uit 5 dagen (maandag t.e.m. vrijdag) waar voor elke dag de uren bijstaan (er kan les gegeven worden van 8u t.e.m. 18u).

Het algoritme zal deze tabel in een eerste fase zo opbouwen dat er aan de fixed constraints voldaan is en dan in een tweede fase proberen om de totale fitnessvalue op te drijven.

2.5.2 Design elements

Om het algoritme te kunnen uitvoeren is er natuurlijk de nodige informatie nodig om tot een lessenrooster te kunnen komen. De informatie die nodig is, zal uit een database worden gehaald en dan in klassenstructuur gegoten worden. Deze klassenstructuur is besproken in "Design Viewpoint 2: Logical". Het schema en bijbehorende uitleg kan daar gevonden worden.

2.6 Design rationale