Software Design Descriptions voor Schedule-Generator

Matthias Caenepeel Adam Cooman Alexander De Cock
Zjef Van de Poel

20 mei 2011 Versie 3.0

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

. 28/2/2011 versie 1.0: Verbeteringen doorgevoerd

. 17/3/2011 versie 1.1: Opmerkingen opdrachtgever in acht genomen en nodige aanpassingen gedaan. Gebruik van de GET en POST methode in de Site - Servlet interface grondiger uitgelegd

. 28/3/2011 versie 2.0: Volledige revisie van het document en aanpassingen/toevoegingen doorgevoerd waar nodig.

. 20/5/2011 versie 3.0: Volledige revisie van het document. Toevoegine van nieuw algoritme, toevoeging van uitleg over de werking van de servlets op de server, toevoeging van nieuwe klassediagrammas.

Inhoudsopgave

1 Compositie **4**
- 1.1 Design concerns . . . 4
- 1.2 Deployment Diagram . . . 4

2 Interfaces **6**
- 2.1 Design concerns, algemeen . . . 6
- 2.2 Interface: XHTML - CSS voor Layout . . . 7
- 2.3 Interface: XHTML - Javascript voor tabbladen . . . 8
- 2.4 Interface: Javascript - CSS voor tabbladen . . . 8
- 2.5 Interface: Browser - Server: HTTP . . . 9
- 2.6 Interface: Database interface . . . 11
- 2.7 Interface: XML interface . . . 13

3 Logical **16**
- 3.1 Design concerns . . . 16
- 3.2 Elementen . . . 16

4 Website/Server Communicatie **18**
- 4.1 Beknopte achtergrond informatie . . . 18
- 4.2 Inplementatie en Communicatie . . . 18

5 Algoritme voor kalenderplanning **20**
- 5.1 Inleiding . . . 20
- 5.2 Overzicht van de klasse SemesterScheduler . . . 20
- 5.3 De solver (methode solve()) . . . 22
- 5.4 Het backtrackingalgoritme per week (methode makeWeekSchedule) 23
- 5.5 Het aanmaken van Calendarfile (getCalendar()) . . . 27

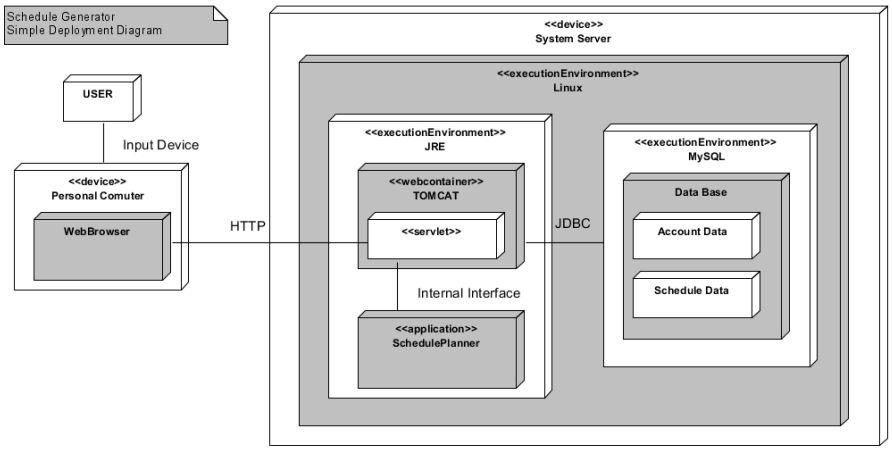
1 Compositie

1.1 Design concerns

Het doel van dit ontwerpstandpunt bestaat er in alle componenten van het systeem te identificeren, hun attributen te kenmerken en hun onderlinge verbanden te beschrijven. Deze beschrijving zal gebeuren aan de hand van een *component diagram* en het *deployment diagram*. Beide zijn een onderdeel van de UML 2.0 modelleertaal.

1.2 Deployment Diagram

Een deployement diagram geeft weer op welke manier de hard- en software wordt geconfigureerd tijdens de normale werking van het systeem. Hieronder volgt een beschrijving van de belangrijkste componenten in het diagram.



Figuur 1: Deployment Diagram

Gebruiker Een gebruiker die beschikt over een account kan zich aanmelden op de website via zijn gebruikersnaam en wachtwoord. Vervolgens zal hij beschikken over de functionaliteiten, eigen aan zijn gebruikertype. Als de gebruiker geen account heeft kan hij de site betreden als gast. Er wordt dan geen gebruikersgebonden informatie bijgehouden. Meer informatie over de verschillende gebruikertypes en functionaliteiten is voorzien in het SRS.

Persoonlijke Computer Toestel dat de hard- en software aan de gebruikerszijde bevat. Er worden geen onderstellingen gemaakt over de eigenschappen van dit toestel met uitzondering dat het instaat is een webbrowser

te draaien met eigenschappen gelijkaardig aan die van Internet Explorer, Google Chrome of FireFox.

Webbrowser Er worden geen specifieke onderstelling gemaakt met uitzondering van de reeds vermelde.

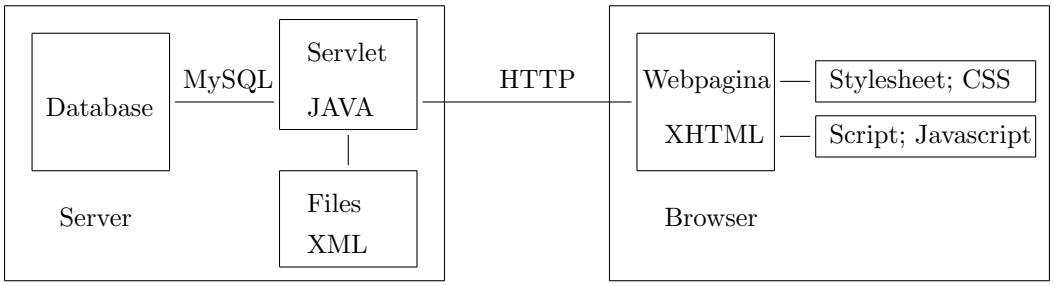
Systeemserver Tijdens de ontwikkeling van het project wordt Wilma als systeemserver gebruikt. De specificaties van de server worden volledig bepaald door de opdrachtgever en kunnen terug gevonden worden op http://wilma.vub.ac.be/.

Tomcat Dit is een webcontainer ontwikkeld door Apache die onder andere toelaat servlets te draaien op een Linux server. Deze servlets zullen de vragen van de gebruiker opvangen en op dynamische wijze webinhoud generen als antwoord. Deze webinhoud zal hoofdzakelijk worden beschreven via XHTML en CVS. Voor meer informatie over Tomcat kan men terecht op http://tomcat.apache.org/.

Database De database maakt onderdeel van de hardware van de systeemserver en bevat zowel de account informatie als de informatie die relevant is voor het opstellen van de lessenroosters. Om gegevens uit de database toegankelijk te maken voor elementen uit de Java runtime environment (JRE) zal gebruikt worden gemaakt van JDBC. Dit is een Java Application Programming Interface ontwikkeld door Sun. Een overzicht wordt gegeven op www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html

2 Interfaces

2.1 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.

2.2 Interface: XHTML - CSS voor Layout

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

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;
}
```

2.3 Interface: XHTML - Javascript voor tabbladen

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

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 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 {}
```

2.5 Interface: Browser - Server: HTTP

Concerns

Omdat de algemene kalender en andere gegevens in een database op de server zullen opgeslaan zijn en omdat de gebruiker vanop zijn computer thuis via de website die informatie te zien moet krijgen, moet er een communicatie tussen de server en de browser van de gebruiker zijn. Zoals bij de meeste websites werd gekozen om het het HTTP protocol te werken. We laten niet toe dat de gebruikers rechtstreeks met de database communiceren om te vermijden dat ze informatie te zien krijgen die ze niet mogen zien, en om de informatie op een overzichtelijke manier te structureren. Daarom werken we met Servlets die op server runnen en die de besproken taken verrichten. De volgende paragrafen beschrijven dus de communicatie tussen de browser van de gebruiker en de servlets.

Attributes

Uit het HTTP protocol zullen de GET en POST functies gebruikt worden door onze toepassing. Een HTTP pakket bestaat uit twee delen, de *header* en de *body*.

GET Deze methode vraagt een bepaalde pagina aan de server en zet alles in de *header*. Het is de methode die gebruikt wordt bij het klikken op een link van een site. Deze zal dus de meest gebruikte methode zijn om te communiceren met de server.

POST Deze methode stuurt informatie door in de *body* van het paket.

De manier waarop de informatie doorgestuurd zal worden hangt af van de plaats. Gevoelige informatie, zoals wachtwoorden zal via een POST verzonden worden, omdat de header gemakkelijk zichtbaar is voor gebruikers. Lange informatie,

zoals beschrijvingen van vakken zal ook via de POST methode verzonden worden, omdat de lengte van de *header* beperkt is.

Andere informatie, zoals het opvragen van het lessenrooster zal via de GET methode gebeuren. De waarden van de zoekactie worden dan in de link geplaatst door een script dat nog ontwikkeld moet worden en waarvan de interface met de XHTML pagina gelijkenissen zal vertonen met die voor de tabbladen.

GET

We gebruiken HttpServlets die de doGet methode bevatten om een Get aanvraag binnen te krijgen en die te beantwoorden. Beschouw als voorbeeld de volgende servlet. Hij stuurt het antwoord Hallo terug naar de gebruiker als die de juiste URL ingegeven heeft. Welke URL dat is wordt beschreven in een XML bestand dat zich op de server bevindt en een bepaalde URL aan een Servlet koppelt.

```
public class test2 extends HttpServlet {
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
  throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("Hallo");
  }
}
```

Om variabelen mee te geven naar de servlet met de GET methode wordt gerbuik gemaakt van een vraagteken. Als een variabele *ID* meegegeven wordt aan een URL gebeurt dit op de volgende manier:

```
servletnaam?ID=waarde
```

in de doGet methode kan de waarde van de variabele opgeroepen worden met de getParameter methode:

```
String ID = request.getParameter("ID");
```

POST

Bij het gebruik van een GET om variabelen aan de servlet door te geven staan de variabelen en hun waarden allemaal in de URL. Voor gevoelige informatie zoals wachtwoorden, of als er zeer veel informatie doorgegeven moet worden (de maximum lengte van een URL in internet explorer is 2083 tekens) is deze methode niet geschikt. Daarvoor zal de POST methode gebruikt worden.

Er zal dus enkel gebruik gemaakt worden van de POST methode bij het verzenden van een HTML Form. Deze HTML structuur bevat velden die ingevuld kunnen worden door de gebruiker en een knop waar de gebruiker op moet

klikken om de informatie door te sturen. Om een variabele PASS door te sturen met een POST naar de servlet zou de code voor de form er als volgt uitzien:

```
<FORM METHOD="POST" ACTION="servlet">
<P>
<INPUT TYPE="PASSWORD" NAME="PASS"><BR>
<INPUT TYPE="submit" VALUE="Send">
</P>
</FORM>
```

In de begintag van het form wordt de opstuurmethode gedefinieerd in het METHOD attribuut. deze kan de waarden POST en GET aannemen. In het ACTION attribuut wordt de locatie van de servlet waar de form naartoe gestuurd wordt gespecifieerd.
Het inputveld waar de gebruiker de data moet invoeren wordt gemaakt met een INPUT tag. Het attribuut NAME definieert de naam van de variabele en het attribuut TYPE definieert de vorm van het invoerveld[1].
Als laatste essentieel deel van de Form is er de knop waar de gebruiker moet op klikken om de gegevens door te sturen. Dit is opnieuw een INPUT tag, maar nu van het type submit. in het Value attribuut staat de tekst die op de knop moet weergegeven worden.

In de servlet kan de doPost methode geimplementeerd worden om de verschillende forms te verwerken. Opnieuw kunnen de variabelen opgevraagd worden met de getParameter methode:

```
String ID = request.getParameter("PASS");
```

Om meer variabelen mee te sturen met een form, is het mogelijk om onzichtbare INPUT tags te definiëren in het HTML Form. daarmee kan aan de Servlet bijvoorbeeld meegegeven worden welk Form ingevuld is.

2.6 Interface: Database interface

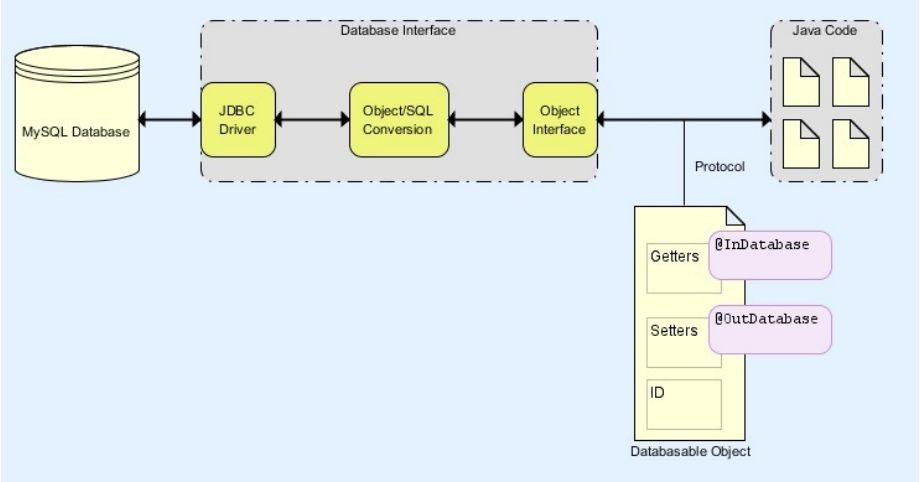
Concerns

De interface met de MySQL database wordt geleverd door de reeds bestaande java library MySQL Connector/J1 [2]. Deze biedt de mogelijkheid om queries uit te voeren vanuit Java code en de resultaten van deze query uit te lezen.

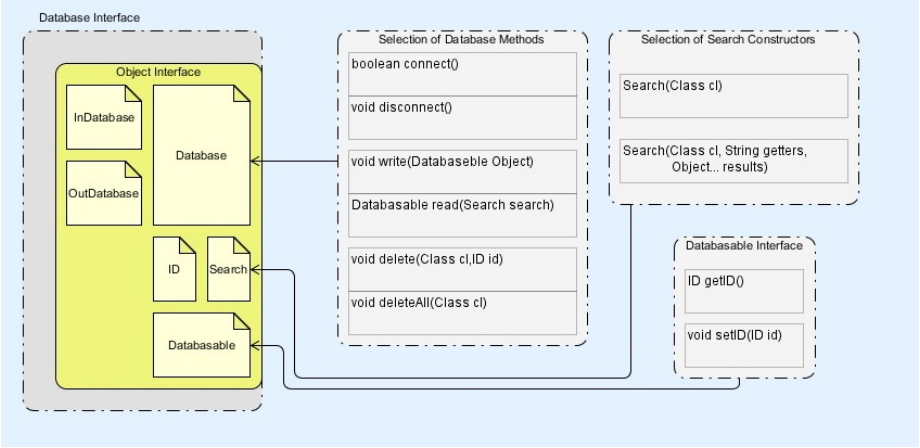
Om eenvoudig en flexibel te kunnen werken met de database vanuit de object georinteerde code is er een extra interface geschreven rond de MySQL Connector. Deze levert op een eenvoudig te implementeren manier de mogelijkheid om rechtstreeks Java objecten op te slaan en deze ook dusdanig als objecten uit te lezen, met inbegrip van alle onderlinge verbanden tussen de objecten.

[1] dit kan veel verschillende vormen aannemen, zie www.w3.org
[2] http://dev.mysql.com/downloads/connector/j/

design



Figuur 3: Overzicht



Figuur 4: Overzicht van belangrijkste klasses en methodes

De interface biedt communicatie met de database op basis van objecten die aangeboden of opgevraagd worden aan de interface. De interface zal dan zelf de nodige informatie extraheren vanuit het object om het in de database op te slaan, of zal de juiste informatie in het object steken om het uit de database te lezen. Om deze data extractie correct te laten gebeuren, moet het object aan een bepaald protocol voldoen. Dit protocol dient zich aan onder de vorm van de java interface klasse 'Databasable'. Het object dient deze klasse te implementeren.

De getters en setters van de parameters die in en uit de database gehaald moeten worden, moeten worden aangeduid met een Annotation. De interface zal tijdens het vertalen via Reflection in Java zo de juiste methodes kunnen selecteren uit de klasse. Meer bepaald:

- Getters annoteren met @InDatabase

- Setters annoteren met @OutDatabase

Tijdens het wegschrijven van een object worden alle parameters die teruggegeven worden door de @InDatabase getters opgeslagen in de database. Tijdens het uitlezen worden alle @OutDatabase setters opgeroepen met de waarde die uitgelezen werd uit de database.

Elk object krijgt een unieke ID (uniek binnen zijn klasse) wanneer hij naar de Database wordt geschreven. Hierdoor moet een parameter van de klasse 'ID' bijgehouden worden. De gebruiker dient deze slechts te voorzien samen met de methodes van de interface Databasable; de invulling van deze ID gebeurt automatisch.

Door dit protocol wordt het schrijven van een object naar de database vereenvoudigd tot het schrijven van:

```
myDatabase.write(myObj);
```

Het uitlezen gebeurt via een 'Search' object, waarmee een bepaald zoek criterium kan opgesteld worden:

```
myObj=myDatabase.read(mySearch);
```

De interface zorgt er ook voor dat de links tussen verschillende objecten ook mee afgehandeld worden. Zie de voorbeelden voor meer detail. Alle voorwaarden om de Databasable interface correct te implementeren zijn te vinden in de Javadoc commentaar van Databasable.

Voorbeelden van correcte implementatie zijn te vinden in de google-code repository bij: trunk/Database/Voorbeeld

2.7 Interface: XML interface

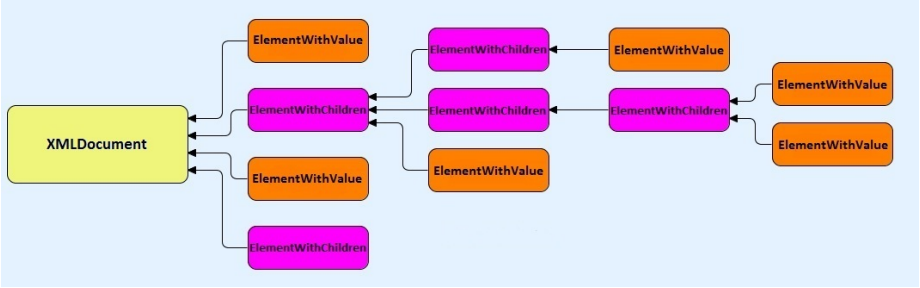
Concerns

Er bestaan reeds talloze XML parsers, maar omwille van flexibiliteit en modificeerbaarheid is er een eigen parser geschreven. Zo is het huidige format niet meer zuiver xml, maar zijn extra features toegevoegd die extra functionaliteiten toevoegen, maar op een manier dat 'echte' xml parsers de door deze parser gegenereerde xml nog steeds kan weergeven.

Design

Een boomstructuur van java objecten wordt opgesteld vanuit een xml bestand, of omgekeerd. Uitgedrukt in BNF:

```
<XMLDocument> ::= {<XMLElement>}
<XMLElement> ::= <ElementWithValue> | <ElementWithChildren>
<ElementWithChildren> ::= <name><value>
<ElementWithChildren> ::= <name>{<XMLElement>}
<name> ::= {<any_character_no_space>}
<value> ::= {<any_character>}
```



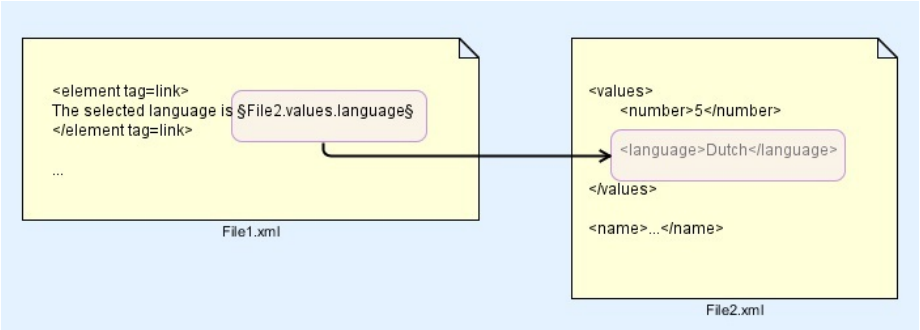
Figuur 5: Voorbeeld van XMLDocument

Features

Tags Specifieke tags kunnen toegevoegd worden aan de elementen:

```
<element tag=XXX> ... </element tag=XXX>
```

Code tag Door deze tag toe te voegen, wordt alle opmaak binnenin het element genegeerd. Dit kan gebruikt worden wanneer men bijvoorbeeld html code wil opslaan in een xml-element. Normaal gezien zouden de html tags interfereren met de xml tags, maar de code-tag zorgt ervoor dat alles wat binnen de tags van het element staat gewoon als tekst aanzien wordt **Link tag** Deze tag laat toe om links te genereren tussen verschillende xml bestanden. Links worden aangegeven door de link tag, waarna in de 'value' van het element een link kan worden toegevoegd tussen ' '. Zie voorbeeld hieronder:



Figuur 6: Link tussen xml files

Na het inladen van File1.xml heeft 'element' de waarde 'The selected language is Dutch'.

3 Logical

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.

3.2 Elementen

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

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*.

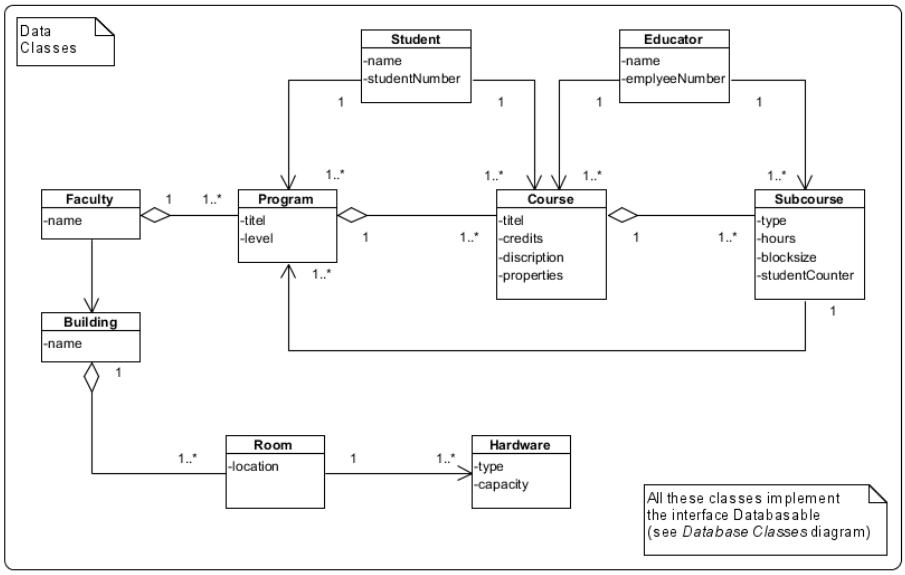
- Gebouwen structuur

  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.

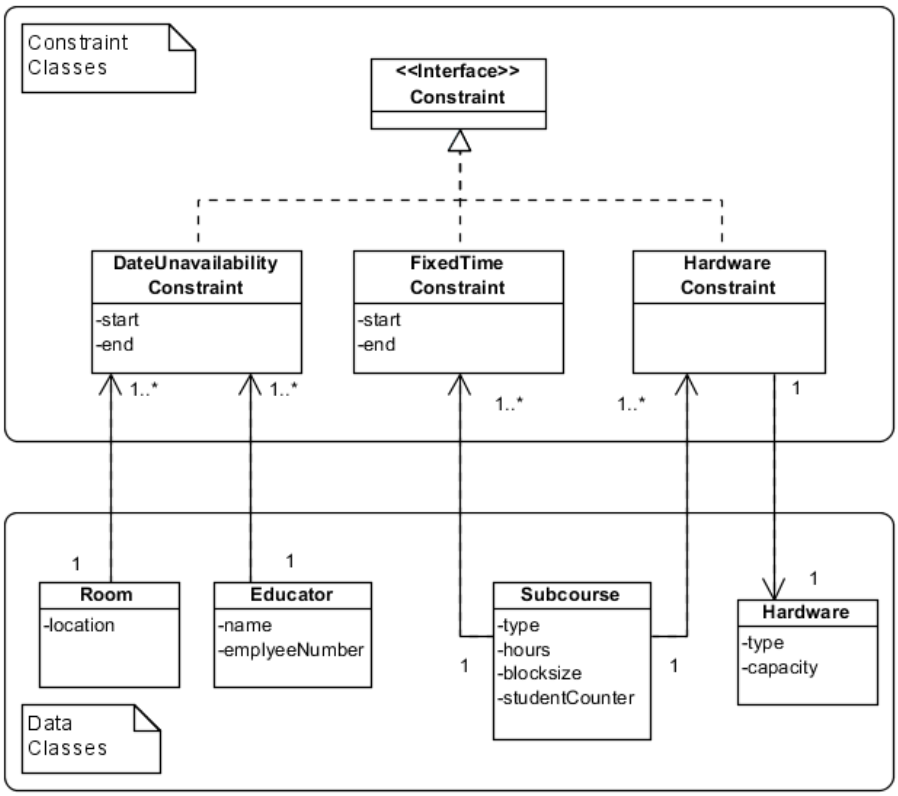
- Personen

  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.

UML Diagrammma



Figuur 7: Dataclasses



Figuur 8: Constraint classes

4 Website/Server Communicatie

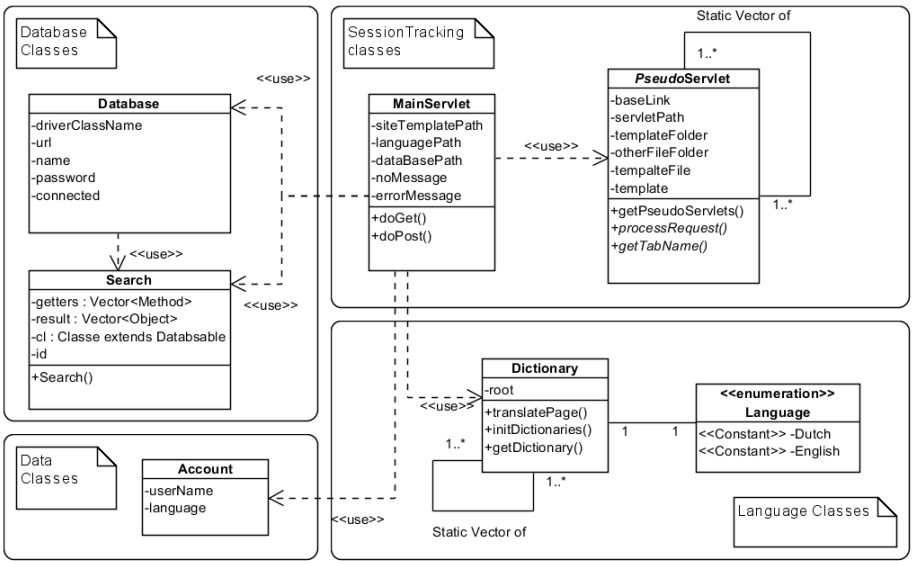
4.1 Beknopte achtergrond informatie

De webpaginas worden op de server gegenereerd door een Java Servlet. Deze ontvangt de requests van de website. De website is modulair opgebouwd door middel van tabbladen. Elk tabblad vervult een afgebakende functie (zoals het bekijken van de eigen kalender, het beheer van de cursussen voor de admins,). De tabbladen die een gebruiker te zien krijgt is afhankelijk van zijn gebruikerstype.

4.2 Inplementatie en Communicatie

Om de benodigde modulariteit te implementeren bestaat er een aparte java klasse/object voor elke functionaliteit/tabblad. Deze ontvangen de request van de browser wanneer het tabblad zijn pagina opvraagt en genereren aan de hand hiervan een pagina als antwoord. Deze pagina bestaat uit een statisch deel, die uit een template file gelezen wordt. Het dynamische deel wordt aangevuld aan

deze template.
Om de beveiliging te controleren (gebruikers die tabbladen opvragen waarvoor ze niet gemachtigd zijn) en om de site in verschillende talen te kunnen weergeven, is er een centraal punt nodig die deze taken, gemeenschappelijk aan alle functies, op zich neemt.
Hiervoor is er een centrale servlet, *MainServlet*, opgesteld die alle request van de browser ontvangt.
De taken zelf worden uitgevoerd door *PseudoServlets*. De MainServlet stuurt de binnengekregen request door naar de correcte PseudoServlet (*public String processRequest()*), op basis van een tag in de query string. Doordat alle requests gebundeld worden door de MainServlet, kan deze op basis van de account nagaan of deze de gevraagde PseudoServlet toegankelijk is voor de gebruiker; zo niet kan deze de request tegenhouden en niet doorsturen naar de desbetreffende PseudoServlet.
De PseudoServlet krijgt via de methode processRequest() de opdracht om een String te genereren die de html pagina voorstelt. Vooraleer deze String naar de browser wordt doorgestuurd door de MainServlet, zal deze eerst door een vertaler gestuurd worden, zodat elke gebruiker zijn eigen taal kan instellen, zonder dat elke PseudoServlet hier apart rekening mee moet houden.
Verder bestaan er ook *PseudoServletsForApplets* (subklasse van PseudoServlet). Deze genereren via processRequest() een html pagina waarin een Java applet geladen wordt. De applet zelf kan communicatie aangaan met de PseudoServlet om meer informatie in te laden of om zelf informatie door te sturen onder de vorm van geserialiseerde Java objecten. Dit delegeert de MainServlet naar de PseudoServlet via processAppletRequest().
De generatie van de totale pagina gebeurt nu eenvoudig. De MainServlet krijgt een request binnen waarvan de id niet is ingesteld. Dit triggert de creatie van een login pagina. Na het inloggen krijgt de gebruiker een id toegewezen die hij met alle requests zal doorsturen ter identificatie (de id wordt gekoppeld aan een account aan de server zijde). Na het inloggen genereert de MainServlet een pagina waarvoor er voor elke PseudoServlet van de gebruiker een tabblad voorzien is. In elk tabblad zit er een iframe waarbij de url naar de desbetreffende PseudoServlet verwijst. De webpagina zal nu alle correcte tabbladen kunnen inladen.



5 Algoritme voor kalenderplanning

5.1 Inleiding

Het doel is om op basis van informatie uit de Database een lessenrooster op te stellen dat aan bepaalde vereisten voldoet. Deze informatie zal via een webinterface kunnen worden ingevoerd door daartoe bevoegde personen. De lessenroosters zullen per semester worden opgesteld. Dit gebeurt in door een object van een klasse die SemesterScheduler heet. Dit object zal dan per semester CalendarFiles genereren die het lessenrooster beschrijven en deze zullen dan worden opgeslagen in de Database.

5.2 Overzicht van de klasse SemesterScheduler

Variabelen:

Vector<Educator> educators (lijst van de docenten)

Vector<Program> programs (lijst van de programmas: bv. 1 Ma EIT)

Vector<Room> rooms (lijst van de lokalen)

Vector<Course> courses (lijst van de vakken die moeten ingedeeld worden)

Bovenstaande vectoren bevatten variabelen uit het package DataStructure, deze klassen bevatten de informatie die nodig is om het lessenrooster op te stellen. Ze zullen uit de Database worden ingeladen met de methode loadData().

int numberOfweeks

int startingHour

int endingHour

Deze drie integers bepalen enkele tijdsdimensies van het lessenrooster. numberOfWeeks bepaalt het aantal weken van de semester. startingHour en endingHour resp. het begin- en einduur per dag (bv. van 8 tot 18 uur)

Hashtable<Educator,Vector<Integer>> unavailableHoursForEducator

Hashtable<Program,Vector<Integer>> unavailableHoursForProgram

Deze hashing tables zullen bijhouden welke uren er niet meer beschikbaar zijn voor een programma of een docent. Dit om dubbelboekingen te bekomen. Voor de docenten bestaat de mogelijkheid om via de webinterface data in te geven waarop hij of zij niet beschikbaar is. (Zie ook verder bij de klasse Constraint.)

Methodes:

SemesterScheduler(int startingHour, int endingHour, int numberOfweeks)
De constructor waaraan de tijdsvariabelen worden meegegeven.

solve()
Deze methode zal het lessenrooster voor heel de semester opstellen en dan de bekomen CalendarFiles wegschrijven in de Database.

De lijst van onderstaande methoden zullen verder in dit document worden aangehaald en dan daar worden uitgelegd.

generateCalendars()

loadData()

createBlocks()

initializeUnavailableHours(...)

calcNextNewSpace(...)

addUnavailableBlock(...)

removeUnavailableBlock(...)

makeWeekSchedule(...)

5.3 De solver (methode solve())

Deze methode zal uiteindelijk alle methoden bevatten die ervoor zorgen dat er een lessenrooster wordt opgesteld. In de constructor is de nodige data reeds ingeladen a.d.h.v. de methode *loadData()*.

De bedoeling is dat er nu per week een lessenrooster wordt opgesteld. Hiervoor is het dus nodig dat de data over het aantal weken worden verdeeld. Het algoritme dat het rooster per week opstelt is een backtrackingalgoritme.

De vector courses bevat alle vakken die moet ingedeeld worden per semester. De klasse course bevat echter nog eens een vector met daarin de klasse Subcourse. Deze klasse geeft aan over welk type les (hoorcollege, werkcollege, labo,...) gaat en bevat ook hoeveel uren er van dit type zijn en hoe lang een minimum blok moet zijn. Per Subcourse zullen er dan object van de klasse Subcourseblock gegenereerd worden. Deze klasse heeft een verwijzing naar zijn Subcourse en bevat de lengte van het blok. Deze blocks moeten dan over het lessenrooster verdeeld worden. Deze blocks moeten echter ook nog eens per week ingedeeld worden. Daarom heeft elk object van Subcourse een veld met daarin een beginweek (geeft aan wanneer deze lessen beginnen) en een minimum aantal uur per week. Er zal hiervoor een methode worden opgeroepen die *createBlocks()* heet. Deze geeft een Vector<Vector<Subcourseblock>> terug. Dit interpreteert men als volgt: elke week heeft een vector van Subcourseblocks. De index van de buitenste vector geeft de week aan.

Vector<Course>→ Vector<Subcourse>→ Vector<Vector<Subcourseblock>>

De methode *createBlocks()* roept een methode op uit de klasse Subcourseblock; *generateBlocksPerWeek(Vector<Course> courses, int NumberOfWeeks)*.

Pseudocode generateBlocksPerWeek

```
weeks =  new Vector<Vector<Subcourseblock>>
Maak voor elke week een Vector<Subcourseblock> (for lus)
  Vraag voor elke Course zijn Subcourses op. (for lus1)
    Ga elke Subcourse af. (for lus2)
      Genereer zijn Subcourseblocks (while lus)

        Verdeel ze over de weken a.d.h.v. zijn beginweek
        en minimum aantal uur per week.
        Voeg de Subcourseblock toe bij de juiste week

      End (while lus)
    End (for lus2)
  End (for lus1)
End (for lus)
Return weeks
```

Vervolgens zal in *solve()* voor elke week de daarbij horende vector van Sub-

courseblocks worden de methode *makeWeekSchedule(Vector<Subcourseblock> blocks, Vector<Room> rooms, int numberOfDays)* opgeroepen. Deze methode zal dan het backtrackingalgoritme uitvoeren en de bekomen resultaten omzetten in calendarfiles. (Meer over deze methode in de volgende paragraaf.)

Pseudocode solve

```
Maak de Subcourseblocks en verdeel ze over de weken.

(Vector<Vector<Subcourseblock>>  weeks  = createBlocks())

Ga de vector weeks af en haal er per week de blocks uit.
Los dan vervolgens het probleem op per week. (for lus)

    Vector<Subcourseblock> = blocks
    makeWeekSchedule(...)

End (for lus)
```

5.4 Het backtrackingalgoritme per week (methode makeWeekSchedule)

Deze methode zal een vector van Subcourseblocks invullen in lessenrooster. Men geeft de vector mee. Bovendien geeft men ook een vector mee met daarin het aantal beschikbare lokalen. Men geeft ook mee over hoeveel dagen het lessenrooster gespreid is.

De SpaceTimeMatrix:

De methode beschikt reeds over een lijst van blocks die moeten ingedeeld worden in het lessenrooster. Het lessenrooster zelf is echt nog niet voorgesteld. Om dit te doen bestaat er een klasse die SpaceTimeMatrix heeft. Deze klasse zal variabelen bevatten die de tijd-ruimte structuur van het rooster voorstellen, alsook methoden voor een geplaatste Subcourseblock het uur, de dag en het lokaal in de week teruggegeven. De belangrijktse variabele van deze klasse is de array Matrix met daarin booleans. De booleans geven weer of de plaats in het lessenrooster al dan niet is ingenomen (true == nog vrij, false == reeds ingenomen). De index van de array stelt de tijd-ruimte structuur voor.

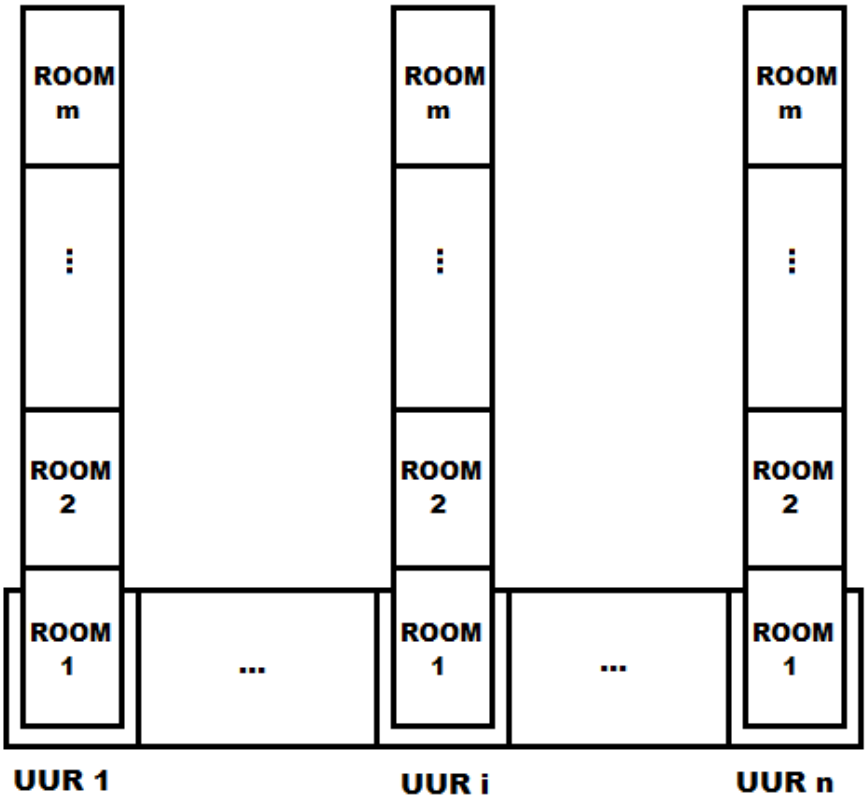
Dit gebeurt als volgt:

Hour loopt van 1 tot (numberOfDays*(endingHour-startingHour)-1): bv. 0 tot 39 (Dit stelt dan 40 uur voor. 5 werkdage en elke dag 8 uur)

numberOfRooms is het aantal lokalen (rooms.size())

roomNumber is de index van de room in rooms.

index = hour * numberOfRooms + roomNumber (1)



Figuur 9: De SpaceTimeMatrix

Uit (1) haalt men dan de informatie voor het lessenrooster als volgt:

roomNumber = index mod numberOfRooms

hour = (index (index mod numberOfRooms)) / numberOfRooms

hourInDay = hour mod (endingHour-startingHour) (bv. 4e uur van een dag)

Day = (hour hourInDay) / (endingHour-startingHour)

Deze klasse stelt dan het eigenlijke lessenrooster voor. Per Subcourseblock zullen dan de indices worden opgeslagen van de toegekende plaatsen en hieruit haalt men dan de tijdsinformatie.

Overzicht van de klasse SpaceTimeMatrix

Variabelen:

Boolean[] Matrix

int startingHour

int endingHour

int numberOfDays

int numberOfRooms

Methoden:

SpaceTimeMatrix(int startingHour, int endingHour, int numberOfDays, int numberOfRooms)

Methoden om dingen op te vragen of aan te passen in de SpaceTimeMatrix

changeBlockAt(int i, Boolean b, int blocksize)
checkBlockAt(int i, Boolean b, int blocksize)
isWithinRange(int i)

Methoden om de indices om te zetten.

giveHour(int i)
giveRoom(int i)
giveHourInDay(int i)
giveDay(int i)

Het algoritme:

Het algoritme om een weeklessenrooster is pure backtracking. Om dit op een efficinte manier te kunnen verwezenlijken heeft men de klasse Node ontwikkeld. Deze klasse bevat de toegekende Subcourseblock, de plaats in het lessenrooster (m.a.w. de index in de SpaceTimeMatrix) en een verwijzing naar de vorig toegekende Node. Aan de hand van deze verwijzing kan men dan teruggaan naar een vorige stap, als men op een doodlopend spoor terecht komt.

Het algoritme zal een vector van Subcourseblocks afgaan en dan steeds een plaats zoeken voor elke Subcourseblock in het lessenrooster; deze gegevens dan in een Node steken en dan naar de volgende Subcourseblock overgaan. Om een plaats te zoeken scant men de SpaceTimeMatrix af en gaat men telkens na of een bepaalde plaats voor die Subcourseblock in aanmerking komt. Om na te gaan of een bepaalde plaats in aanmerking komt, moet men de vereisten voor het lessenrooster nagaan. Deze vereisten worden in de onderstaande kader weergegeven.

Vereisten waaraan het rooster moet voldoen:

(1) Er mag geen dubbelboeking van een lokaal optreden.

(2) Het lokaal moet groot genoeg zijn voor het aantal studenten.

(3) Het lokaal moet het nodige materiaal voor een les bevatten.

(4) Programmas mogen niet dubbel geboekt zijn.

(5) Een docent mag niet dubbelgeboekt zijn.

(6) Een docent moet beschikbaar zijn.

(7) Op vakantiedagen mag er niets gepland zijn.

Om deze vereisten na te kunnen gaan, als men een Subcourseblock in een het lessenrooster wilt plaatsen, heeft men de klasse Constraint gecreerd. Deze klasse is een verzameling van staticsche methodes die steeds een Boolean terug gegeven om aan te geven of er al dan niet aan de vereisten voldaan is (false indien niet, true indien wel).

Overzicht van de klasse Constraint

int i
Dit is de index in de SpaceTimeMatrix die men evalueert. Hieruit kan men alle nodige informatie afleiden (zie ook SpaceTimeMatrix).

roomAvailableAtBlock(int i, Room room, Subcourseblock block) Deze methode checkt of het lokaal gedurende periode van het Subcourseblock nog beschikbaar is. (1)

roomSufficient(int i, Room room, Subcourseblock block)
Deze methode checkt of het lokaal groot genoeg is en het vereiste materiaal bevat. (2) en (3)

hourAvailable(int i, Subcourseblock block, Hashtable hoursAvailableForEducator, Hashtable hoursAvailableForProgram) Deze methode gaat na of er geen dubbelboekingen zijn voor de docent of voor het programma. hoursAvailableForEducator en hoursAvailableForProgram zijn objecten van de klasse Hashtable (hashingtables) waarin de vectoren zijn opgeslagen met daarin de reeds toegekende uren voor een docent of voor een programma. (4) en (5)

dateAvailable(int i, Subcourseblock block)
Deze methode gaat na of de dag waarop men iets wilt zetten geen verlofdag is en of de docent op die dag wel beschikbaar is. Dit gebeurt aan de hand van Calendarfiles. Elke docent heeft een persoonlijke Calendarfile en er is een algemene Calendarfile met daarin de verlofdagen voor een semester. Deze methode gaat overlappingen na. Indien er een overlapping is, is er niet aan deze vereiste voldaan. (6) en (7)

Tijdens het backtracken, zal er steeds vooraleer men een Subcourseblock kan plaatsen de methode calcNext(int i) worden opgeroepen. Deze methode zal steeds de volgende beschikbare plek in het lessenrooster teruggegeven. Ze gaat de indices van de SpaceTimeMatrix af en de eerst mogelijke die voldoet aan de vereisten wordt teruggegeven. De index die men meegeeft is de laatst toegekende plek in het rooster.

Het backtrack algoritme:

```
Zolang niet elke block is toegekend (while lus)

Zoek een plek in het rooster

    Plek gevonden:
    Vul deze plek in SpaceTimeMatrix
    Maak een Node aan
    Zorg dat de uren voor de docent en het programma onbeschikbaar zijn

End (while lus)
```

Aan elke Subcourseblock zal zijn plaats de SpaceTimeMatrix worden toegekend. Aan de hand van deze informatie zal hij dan de dag, het uur in die dag en het lokaal kennen. Zijn week kent hij ook. Op die manier zal elke Subcourseblock alle informatie bevatten om in een Calendarfile te plaatsen.

5.5 Het aanmaken van Calendarfile (getCalendar())

Elke Subcourseblock heeft nu een plek in het rooster. Nu moet dit nog omgezet worden in een bruikbare structuur die door gebruikers kan worden geraadpleegd. Hiervoor zal men per Subcourse een Calendarfile maken. De som van al deze Calendarfiles zal dan het hele lessenrooster voor de hele universiteit zijn.

De methode *getCalendar()* zal alle Subcourseblocks afscannen en dan steeds hun Subcourse opvragen. Hiervan vraagt men dan de Calendarfile op en het Subcourseblock zal dan hieraan worden toegekend.