

Software Design Descriptions voor Schedule-Generator

Matthias Caenepeel Adam Cooman Alexander De Cock
Zjef Van de Poel

23 februari 2011 Versie 1.0

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

Inhoudsopgave

1	Body	4
1.1	Identified stakeholders and design concerns	4
1.2	Design viewpoint 1: Compositie	4
1.2.1	Design concerns	4
1.2.2	Deployment Diagram	4
1.2.3	Component Diagram	5
1.3	Design viewpoint 2: Logical	5
1.3.1	Design concerns	5
1.3.2	Elementen	6
1.4	Design viewpoint 3: Interfaces	8
1.4.1	Design concerns, algemeen	8
1.4.2	Interface: XHTML - CSS voor Layout	9
1.4.3	Interface: XHTML - Javascript voor tabbladen	9
1.4.4	Interface: Javascript - CSS voor tabbladen	10
1.4.5	Interface: Browser - Server: HTTP	11
1.4.6	Interface: Database interface	12
1.4.7	Interface: File interface	13
1.5	Design viewpoint 4: Algoritme voor kalenderplanning	14
1.5.1	Design concerns	14
1.5.2	Design elements	15

1 Body

1.1 Identified stakeholders and design concerns

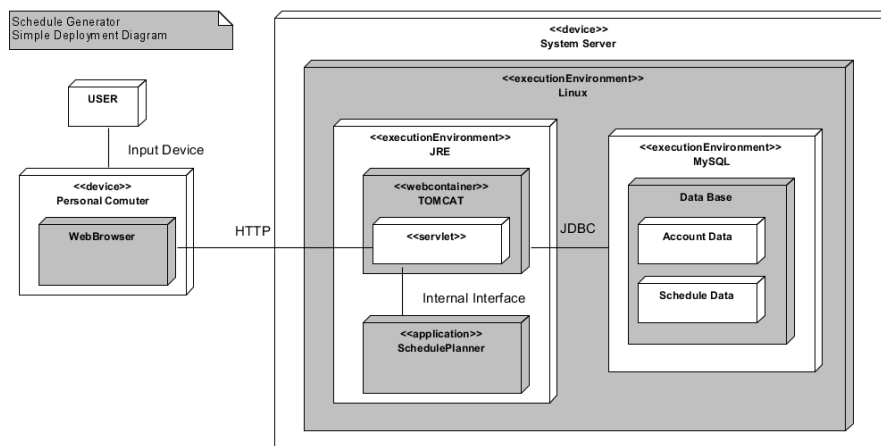
1.2 Design viewpoint 1: Compositie

1.2.1 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.

1.2.2 Deployment Diagram

Een deployment 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 gebruikerzijde 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

1.2.3 Component Diagram

Zal later worden toegevoegd

1.3 Design viewpoint 2: Logical

1.3.1 Design concerns

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

1.3.2 Elementen

1.3.2.1 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 structuur
 - *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

1.3.2.2 Relaties

- Lessen structuur

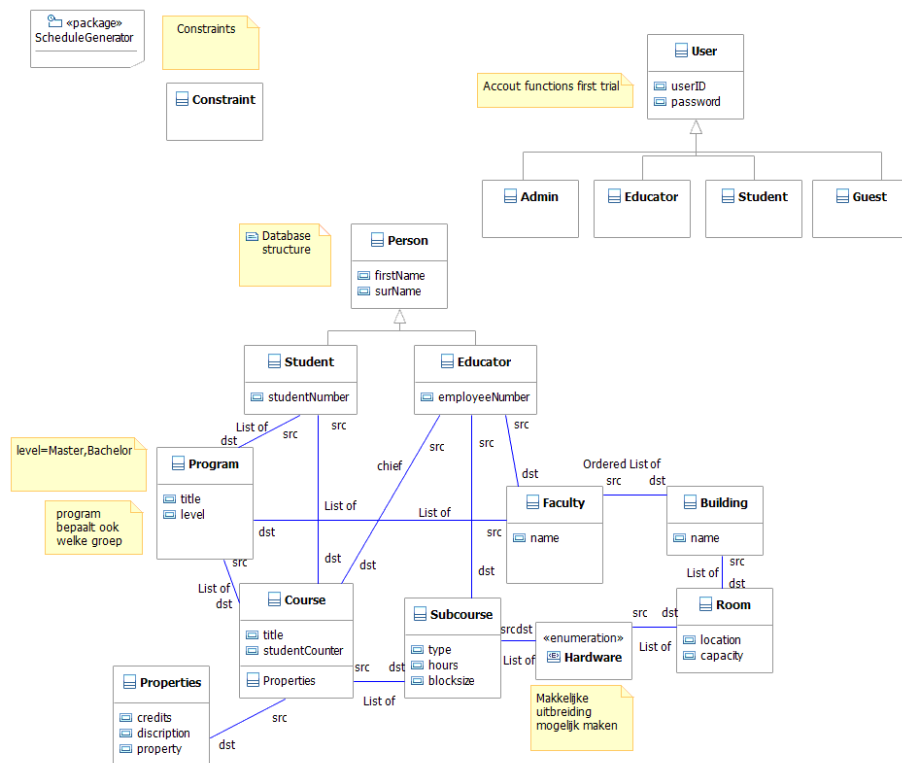
Elke *Course* bestaat uit een of meerdere *SubCourses* die in opgeteld het gehele vak weergeven.
Courses zelf worden gegroepeerd in een *Program*.
- Gebouwen structuur

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.

1.3.2.3 UML Diagrammma

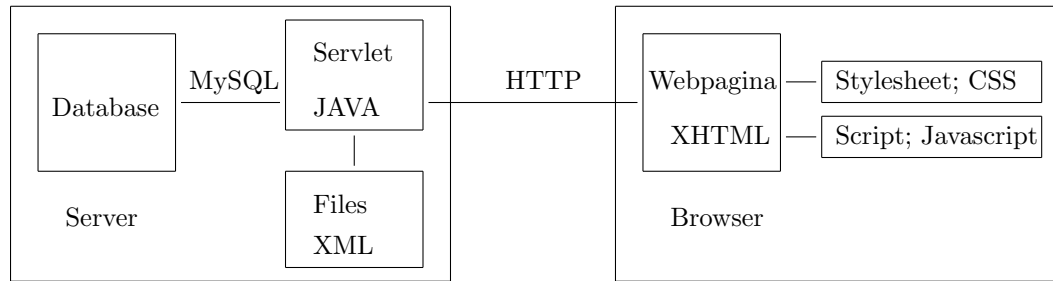
De klasse waarbij de pijl vertrekt erft van de klasse waar de pijl aankomt. De klasse waarbij er src staat zit in een lijst in de klasse waarbij er dst staat. Zo zit er bijvoorbeeld een lijst van educators in de klasse faculty.



Figuur 2: UML Diagramma

1.4 Design viewpoint 3: Interfaces

1.4.1 Design concerns, algemeen



Figuur 3: 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 diagram 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.

1.4.2 Interface: XHTML - CSS voor Layout

1.4.2.1 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

1.4.2.2 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 beschrijft van de tabelrij uit het vorige voorbeeld.

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

1.4.3 Interface: XHTML - Javascript voor tabbladen

1.4.3.1 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

1.4.3.2 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 gebeurt 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 volgende:

`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

1.4.4 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 werkt. 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 {}
```

1.4.5 Interface: Browser - Server: HTTP

1.4.5.1 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.

1.4.5.2 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 pakket.

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.

We gebruiken `HttpServlet`s 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");
    }
}
```

Het gebruik van de `POST` methode wordt op dit moment nog onderzocht door het team en zal binnenkort verduidelijkt worden.

1.4.6 Interface: Database interface

1.4.6.1 Concerns

De interface met de MySQL database, bevattende alle objecten beschreven in 2.1, dient op een algemene manier aangesproken te kunnen worden vanuit de andere onderdelen van het project.

1.4.6.2 Elementen

- Database
 - **Database**: Handle naar de database. Levert methodes om tabellen in te lezen, opzoeken te doen, objecten weg te schrijven en dergelijke.
 - **Table**: Een enkele tabel uit de database
 - **Element**: Een element van een tabel
- Interface
 - **Databasable**: Interface* 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

1.4.7 Interface: File interface

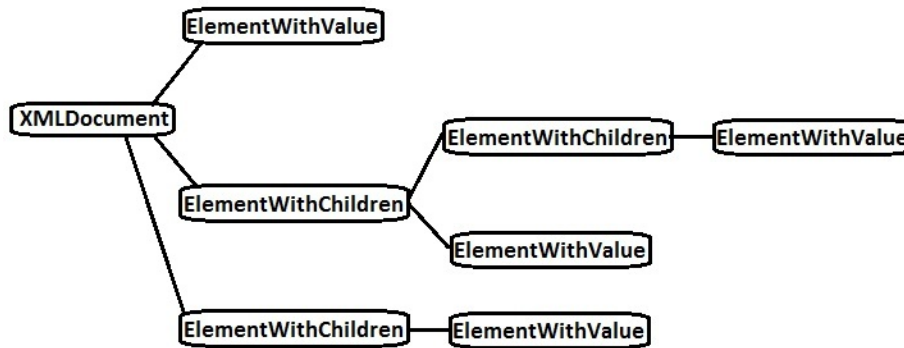
1.4.7.1 Concerns

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.

1.4.7.2 Elementen

- **XMLDocument**: Object verwijzende naar een XML bestand. Een XML-Document bevat een lijst van XML-Element
- **XML-Element**: 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 *ElementWithChildren* of *ElementWithValue* zijn)
- **XMLParser**: Levert methodes om een XML bestand, gecreerd met deze interface, weer in te laden naar een XMLDocument

1.4.7.3 Schema



Figuur 4: Voorbeeld van een *XMLDocument*

1.5 Design viewpoint 4: Algoritme voor kalenderplanning

1.5.1 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 bijvoorbeeld hoogstens 4 uur per dag gegeven worden.
3. Student mogen bijvoorbeeld 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.

1.5.2 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. 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.