Verslag Programmeerproject

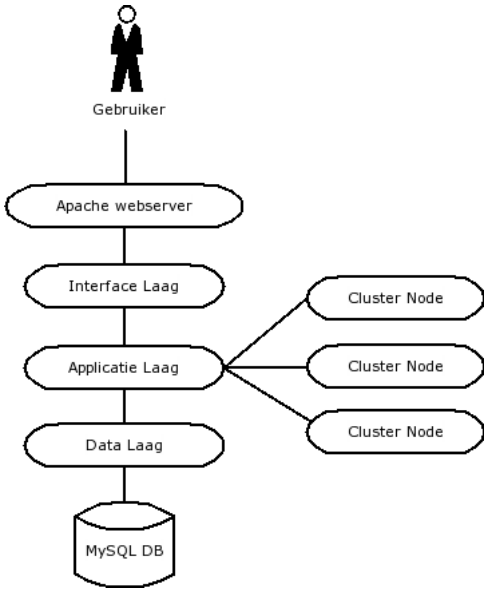
Thomas Raes
75972
traes@vub.ac.be

6 juni 2006

Samenvatting

Dit document beschrijft een geparallelliseerd scheduler programma gemaakt als programmeerproject voor de 1e licentie kerninformatica aan de Vrije Universiteit Brussel, academiejaar 2005-2006.

Inhoudsopgave



Figuur 1: Overzicht architectuur

1 Inleiding

Het doel van het project is het ontwerpen en implementeren van een geparallelliseerd scheduling-programma voor het departement Lerarenopleiding van de Erasmushogeschool Brussel. Concreet betekent dit dat er lokalen en uren worden toegekend aan lessen, rekening houdend met de voorkeuren van de gebruikers.

De belangrijkste prioriteit is de gebruikersvriendelijkheid. Via een web-interface kunnen gebruikers instellingen, voorkeuren en de pool van clusternodes beheren. De gegevens worden opgeslagen in een relationele database en zijn te raadplegen via de web-interface.

Het doel van dit document is het toelichten van de genomen beslissingen tijdens de uitvoer van het project. Eerst wordt de algemene architectuur besproken en vervolgens wordt er ingegaan op de technische details. Daarna worden de wijzigingen tegenover het origineel ontwerp besproken en verantwoord. Uiteindelijk worden de belangrijkste bevindingen samengevat in de conclusie.

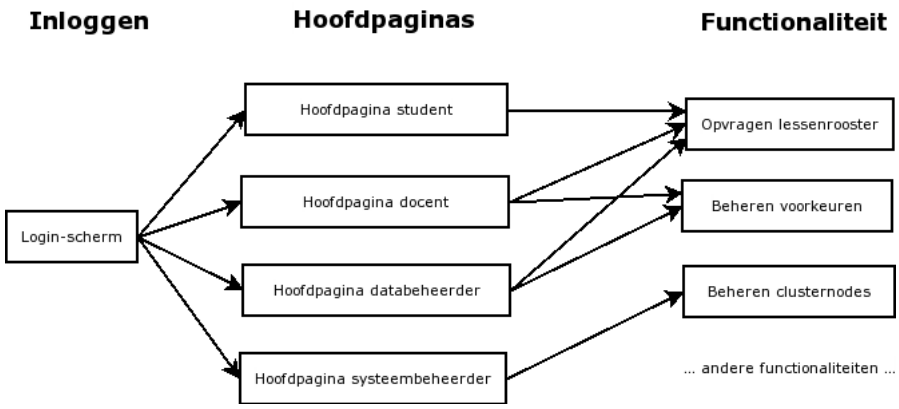
2 Architectuur

2.1 Overzicht

Het programma bestaat uit een 3-lagen architectuur.

- **Interfacelaag:** communicatie met gebruikers
- **Applicatielaag:** logica, schedulen
- **Datalaag:** gegevensbeheer

Deze lagen wisselen via een netwerk data uit in XML formaat. De verschillende lagen kunnen dus op afzonderlijke computers geïnstalleerd



Figuur 2: Web-pagina's

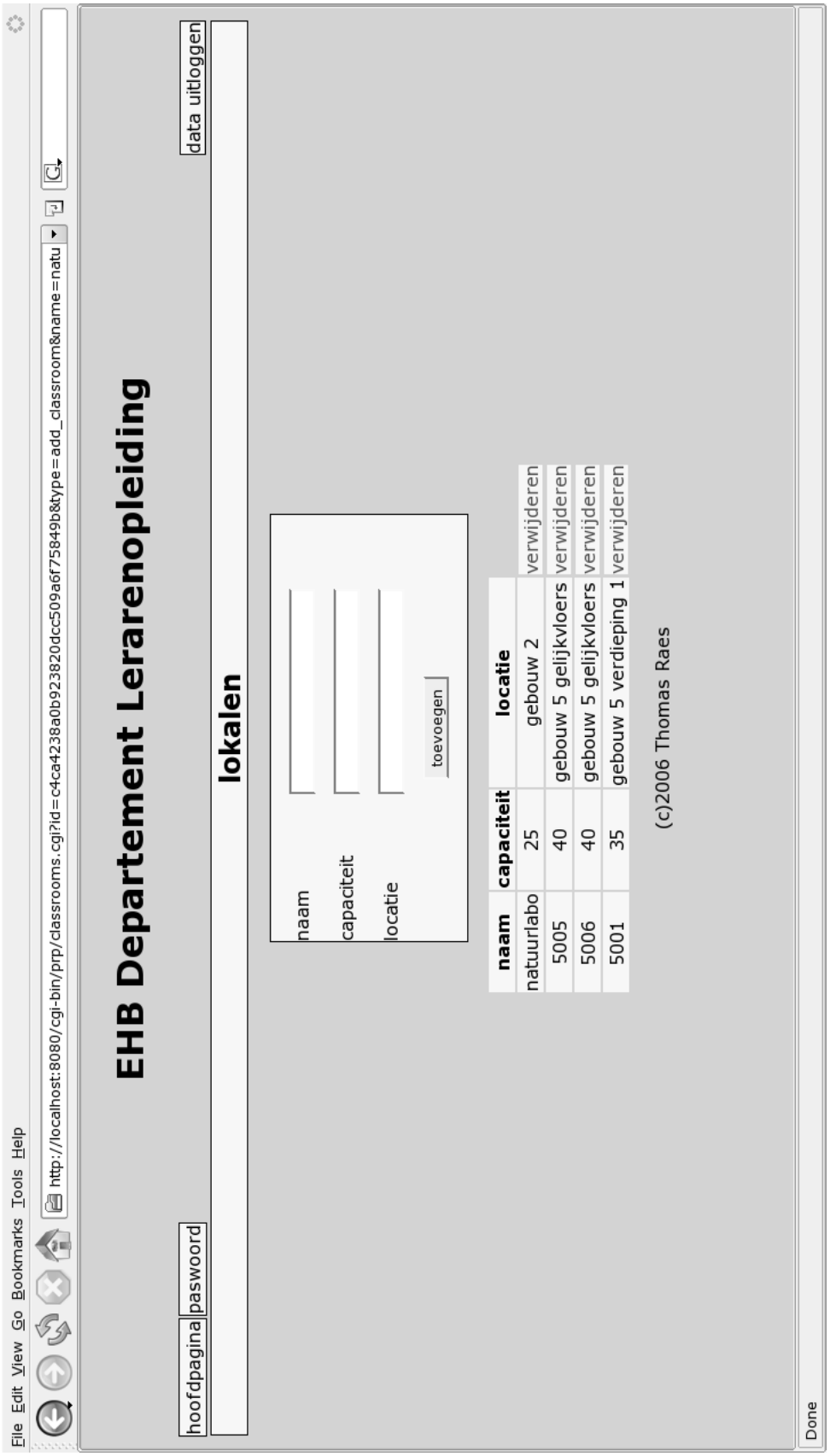
worden. Het systeem gebruikt een reeds bestaande web-server (Apache [3]) en relationele database (MySQL [4]). De andere componenten zijn zelf ontwikkeld en geschreven in de programmeertaal Ruby [1].

2.2 Interface

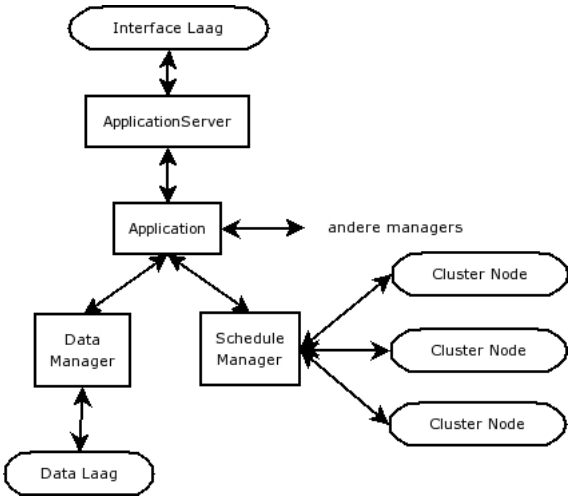
De interface is verantwoordelijk voor de communicatie met de gebruiker. Bij het ontwerpen van de interface was gebruiksvriendelijkheid de bepalende factor. De verschillende pagina's zijn zo eenvoudig, consistent en overzichtelijk mogelijk ontworpen. Hiervoor werden de volgende stappen ondernomen:

1. alle functionaliteit is bereikbaar vanaf de hoofdpagina
2. de hoofdpagina's zijn vanaf elke pagina bereikbaar
3. uitloggen en paswoord wijzigen is mogelijk vanaf elke pagina
4. de kleur-schema's zijn afhankelijk van het gebruikerstype
5. de keuzemenu's bevatten automatisch de mogelijke opties

De interface bestaat uit verschillende cgi-pagina's. De pagina's gebruiken in Ruby geschreven *PageGenerators* voor het weergeven van de pagina. Het PageGenerator object handelt de communicatie met de applicatielaag af en produceert HTML en CSS code. Op deze manier kan er met 1 regel code een hele web-pagina aangemaakt worden en hebben alle pagina's en consistent uiterlijk en gedrag.



Figuur 3: Webpagina lokalenbeheer



Figuur 4: Architectuur applicatie

2.3 Applicatie

De applicatie is verantwoordelijk voor het beheren van de sessies en het opstellen van de roosters.

De verschillende taken van de applicatie zijn verdeeld over managers. De *ScheduleManager* is verantwoordelijke voor het aanmaken van lessenroosters en coordineert de samenwerking tussen de verschillende nodes van de cluster. Deze manager is ook verantwoordelijk voor het afhandelen van de commando's van de gebruiker. Commando's zijn volgens het **Command** design pattern voorgesteld als afzonderlijke objecten. De *DataManager* is verantwoordelijk voor de communicatie met de datalaag. Nog andere managers zijn verantwoordelijk voor het afhandelen van sessies, gebruikers, configuratie, ...

2.4 Data

De datalaag is verantwoordelijk voor het opslagen van de informatie. Het communiceert in XML met de applicatielaag en stuurt SQL commando's naar de database. Deze laag is ook in staat om backups van de database te maken, en deze backups terug in te laden.

3 Ontwerp

3.1 Overzicht

Het systeem bestaat uit 3 lagen, elke laag gebruikt een ander formaat om informatie intern voor te stellen.

- **Interface:** HTML
- **Applicatie:** Ruby Objecten
- **Data:** Relationele tabellen

Daarom gebeurt de communicatie tussen de lagen in XML formaat, iedere laag weet hoe het data kan omzetten van XML naar het intern gebruikte formaat en omgekeerd. Vermits elke laag op een afzonderlijke computer kan werken, gebeurt alle communicatie via het netwerk.

3.2 Interface

De web-interface bestaat uit cgi pagina's, deze gebruiken *PageGenerator* objecten om HTML en CSS te genereren en staan in voor de communicatie met de applicatielaag. Hierdoor is het eenvoudig om alle pagina's en consistent uiterlijk en gedrag te geven en de communicatie met de applicatielaag ordelijk te laten verlopen. De code in de cgi pagina's zelf wordt ook compacter, de volgende regel code volstaat om de web-pagina van Figuur 3 te genereren:

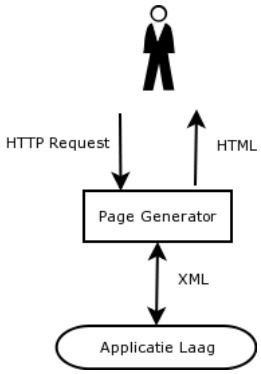
```
DataPageGenerator.new(CGI.new,'classrooms.cgi','lokalen',['data-admin'],
   'classroom',['name','capacity','location'],['naam','capaciteit','locatie'])
```

Communicatie De PageGenerators communiceren met de applicatie via een *Communication* object. Dit object vertaalt de aanvragen van de PageGenerator naar XML en stuurt ze door naar de applicatielaag. Vervolgens wordt het antwoord van de applicatielaag door de PageGenerator verwerkt. Met behulp van een *ConfigurationFile* object weet het Communication object hoe het de applicatielaag kan bereiken.

Authenticatie De PageGenerator communiceert met de applicatielaag en laat deze eerst controleren of het session id geldig. Vervolgens wordt het gebruikerstype aan de applicatielaag opgevraagd zodat de interface kan controleren of de gebruiker wel bevoegd is om de gevraagde pagina te bekijken.

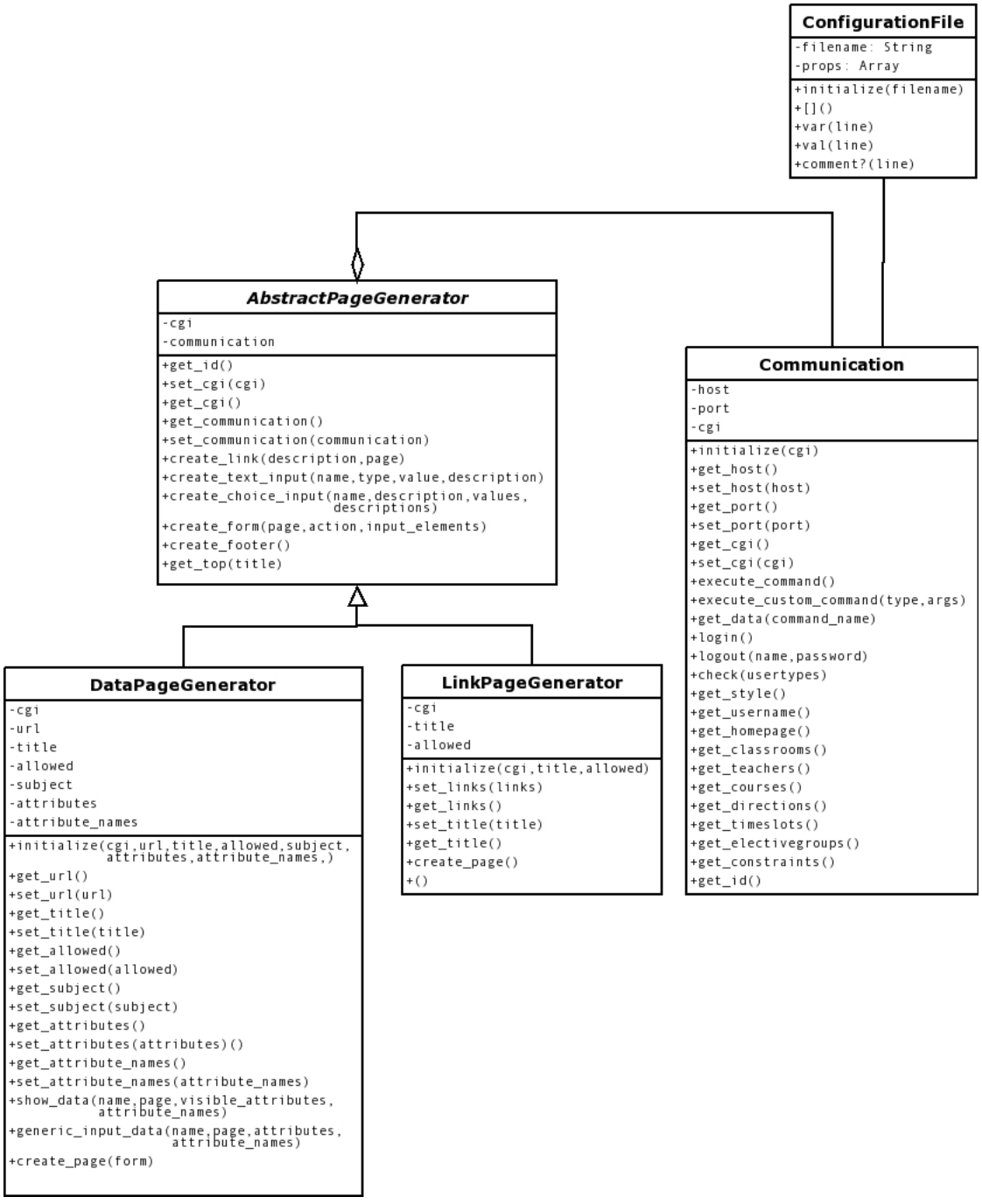
Personalisatie De PageGenerator vraagt aan de applicatielaag de stijlinformatie van de betreffende gebruiker. Dankzij deze CSS code kan de lay-out van de pagina's eenvoudig aangepast worden. In de applicatielaag wordt deze CSS code gegenereerd door de **Viewer** van de sessie.

Databeheer Het PageGenerator object hoeft enkel het onderwerp van de pagina (bv. *classroom*) te kennen om zelf de nodige commando's naar de applicatielaag te sturen (*add_classroom*, *remove_classroom*, *view_classroom*). Het afhandelen van deze commando's is de verantwoordelijkheid van de **Viewer** (view) en **Controller** (add,remove) van de gebruiker.



Figuur 5: werking PageGenerator

Zoals te zien is op Figuur 2 bestaan er 2 soorten pagina's: pagina's die de een bepaalde functionaliteit aanbieden en pagina's die links bevatten. Voor beide soorten is er een aangepaste PageGenerator, namelijk *DataPageGenerator* en *LinkPageGenerator*. Beide generators erven algemene functionaliteit over van *AbstractPageGenerator* Men kan aan een DataPageGenerator een aangepast invulformulier laten aanmaken, bijvoorbeeld met een keuzemenu met alle leerkrachten in de database. Indien er geen speciaal formulier aangevraagd wordt, gaat de PageGenerator er van uit dat alle attributen met een textveld ingevuld kunnen worden.



Figuur 6: PageGenerator

3.3 Applicatie

De applicatielaag bevat de eigenlijke logica van het programma. De verschillende taken zijn verdeeld over managers die ieder hun specifieke verantwoordelijkheid hebben.

Alle communicatie met de applicatielaag gebeurt in XML. De commando's van de interfacelaag worden ontvangen als XML en de antwoorden teruggestuurd als XML. Als de applicatielaag informatie van de datalaag nodig heeft, stuurt deze een aanvraag voor informatie in XML formaat naar de datalaag en krijgt vervolgens een antwoord terug dat ook in XML staat.

Vermits alle communicatie via het netwerk verloopt, kunnen de applicatielaag, interfacelaag en datalaag op een afzonderlijk computer werken.

3.3.1 Managers

Er zijn verschillende managers, die elk een deel van de verantwoordelijkheid van applicatielaag op zich nemen. Van elke manager bestaat er juist 1 instantie (**Singleton** design pattern).

ConfigManager De ConfigManager is verantwoordelijk voor de configuratie, zoals het poort-nummer waarop de applicatielaag naar commando's van de interfacelaag luister, het netwerk-adres waarop de datalaag bereikt kan worden, de locatie van de log-bestanden, ...

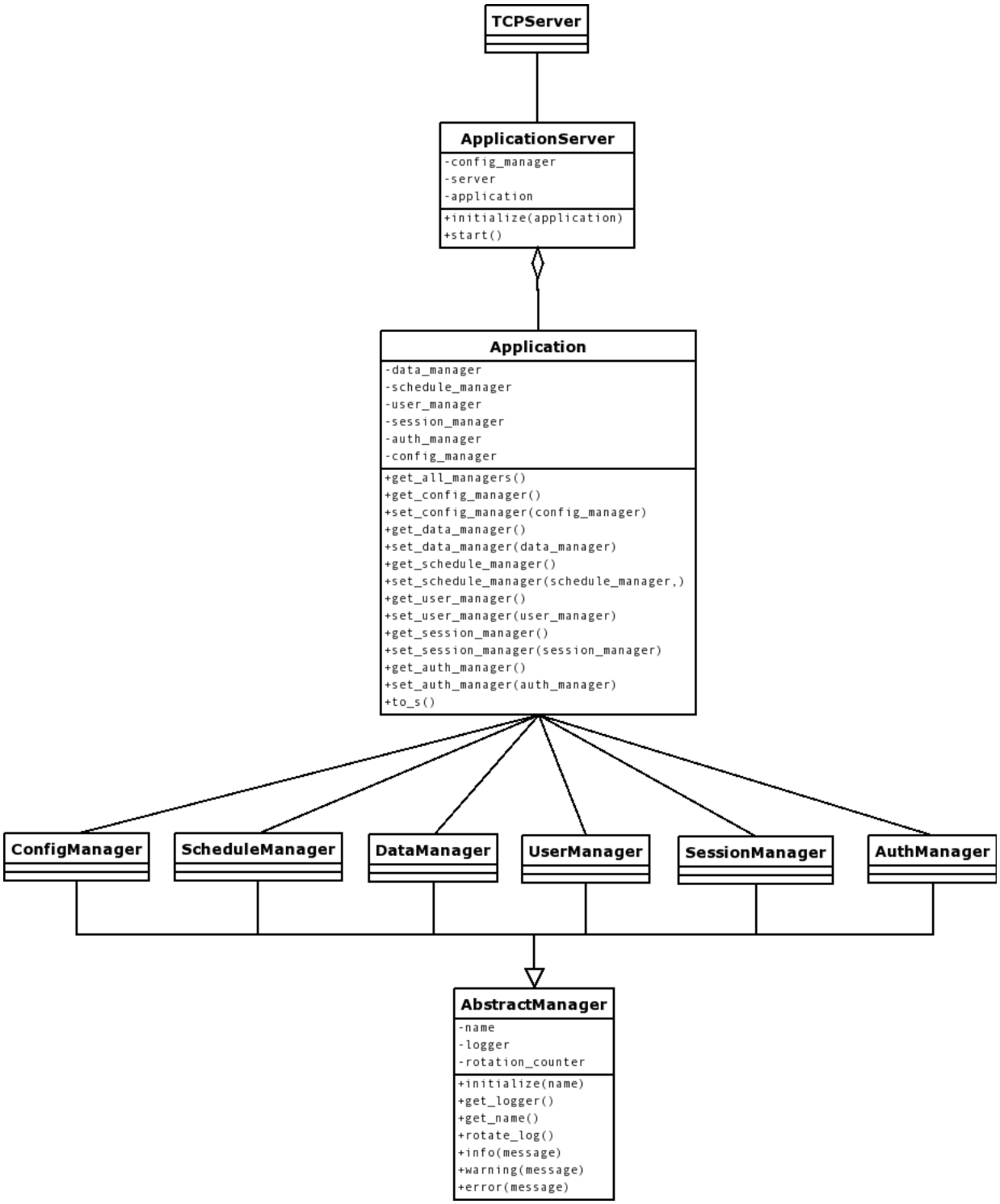
SessionManager De SessionManager is verantwoordelijk voor het beheren van de sessies. Het behandelt aanvragen voor nieuwe sessies, het stopzetten van reeds bestaande sessies en het doorsturen van commando's naar de sessies zelf. Voor het aanmaken van de sessies wordt samengewerkt met de *AuthManager* en de *UserManager*. De uit te voeren commando's worden volgens het **Command** design pattern voorgesteld door een Command object.

AuthManager De AuthManager is verantwoordelijk voor het controleren van de paswoorden. Om veiligheidsredenen zijn deze paswoorden gehashed met het MD5 algoritme. Voor dezelfde reden wordt het paswoord van de systeembeheerder in de applicatielaag zelf opgeslagen en niet in de datalaag.

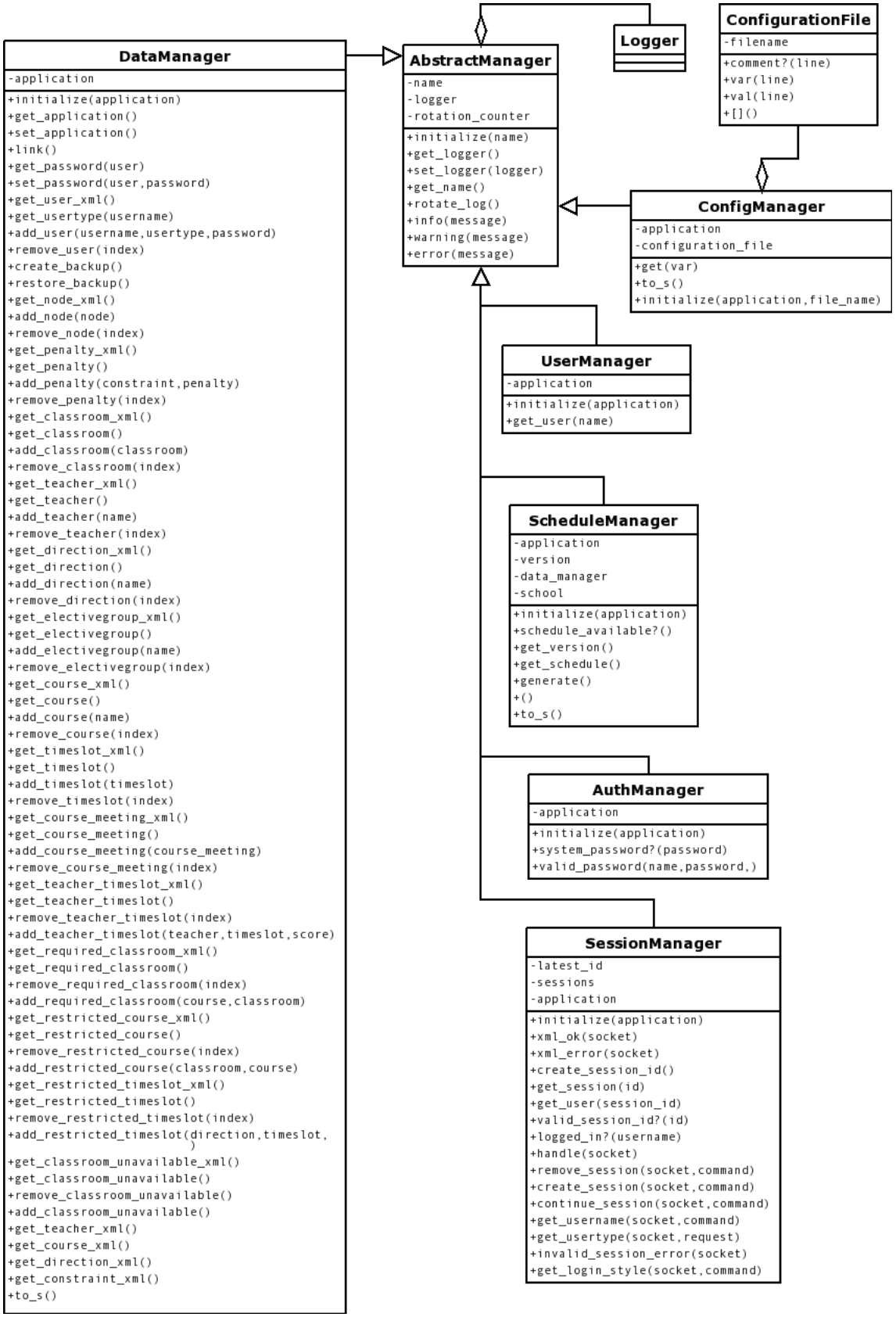
UserManager De UserManager beheert de gebruikers-objecten. Deze zijn nodig bij het inloggen.

ScheduleManager De ScheduleManager staat in voor het aanmaken van nieuwe lessenrooster. Deze manager is ook verantwoordelijk voor werkverdeling over de cluster nodes.

DataManager De DataManager is verantwoordelijk voor de communicatie met de datalaag. Het vertaalt aanvragen voor informatie naar XML en vertaalt de antwoorden van de datalaag van XML naar gewone Ruby objecten.



Figuur 7: Applicatielaag (details managers op Figuur 8)



Figuur 8: Ontwerp managers

3.3.2 Gebruikers en sessies

Er zijn 4 soorten gebruikers en overeenkomstige sessies.

Student Studenten volgen de opleiding tot leerkracht, ze kunnen lessenroosters opvragen volgens hun studierichting.

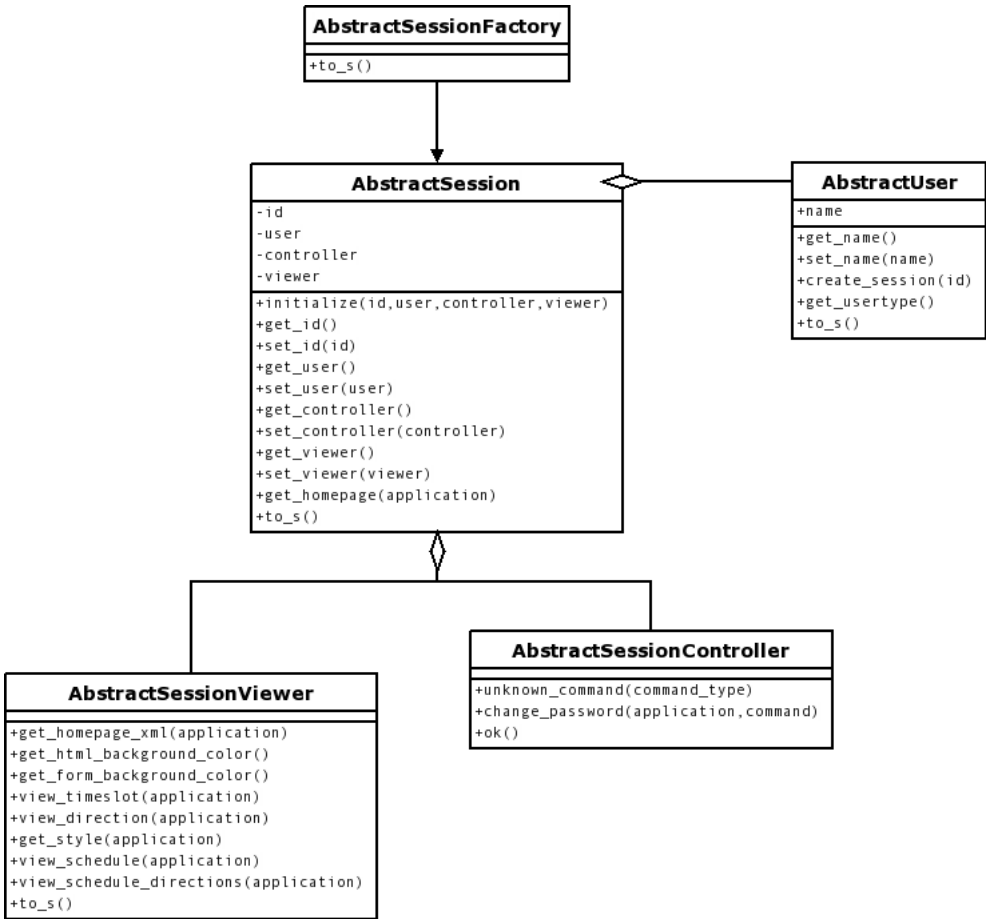
Docent Docenten geven les aan studenten. Ze kunnen een rooster opvragen met alle lessen die ze geven en ze kunnen hun persoonlijke voorkeuren voor het aan te maken lessenrooster ingeven.

Databeheerder De databeheerder kan algemene informatie over de school (lokalen, leerkrachten, ...) beheren. Hij kan ook de algemene voorkeuren van het rooster instellen (bv. onbeschikbare lokalen), alsmede hun prioriteit. De databeheerder is ook in staat om nieuwe lessenroosters te genereren. Er is hoogstens 1 databeheerder.

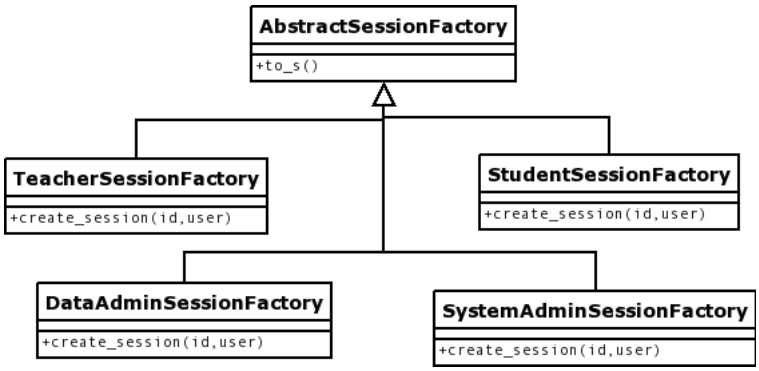
Systeembeheerder De systeembeheerder beheert de technische aspecten van het systeem. Deze zijn:

- accounts aanmaken
- accounts verwijderen
- log-bestanden bekijken
- log-bestanden roteren
- backups maken
- backup inladen

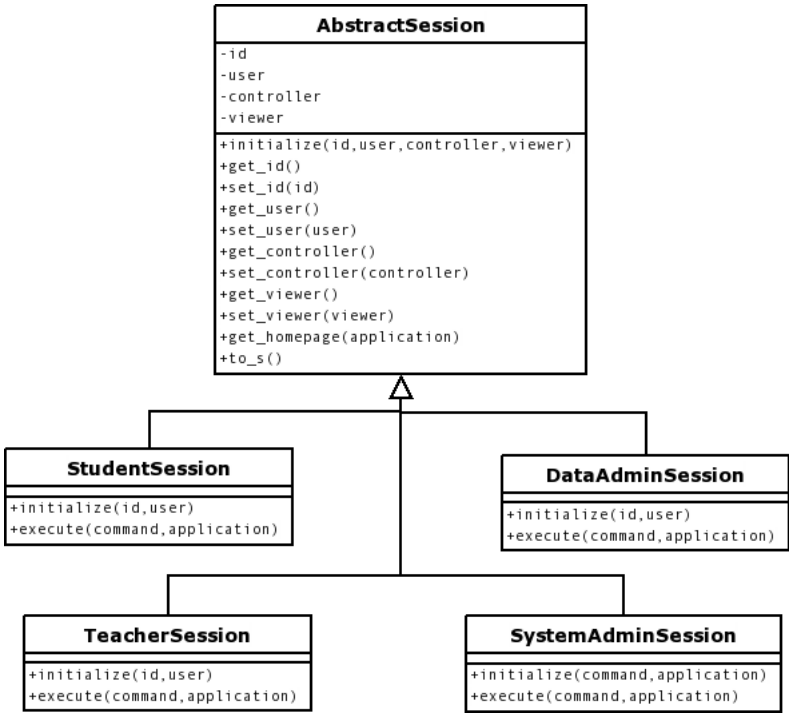
Elke gebruikerstype heeft een afzonderlijk sessie-type met een aangepaste **Viewer** (Figuur 13) en **Controller** (Figuur 12). Om sessies eenduidig aan te maken wordt er gebruik gemaakt van een **AbstractFactory** (Figuur 9). Voor elk soort gebruiker bestaat er een aangepaste **Factory** die de overeenkomstige sessie aanmaakt. Elk User object (Figuur 14) bevat het juiste SessionFactory object, zodat de SessionManager aan elke gebruiker de method *create_session* kan sturen en er automatisch het juiste session object aangemaakt wordt. Het User object maakt dus gebruik van het **Factory Method** design pattern.



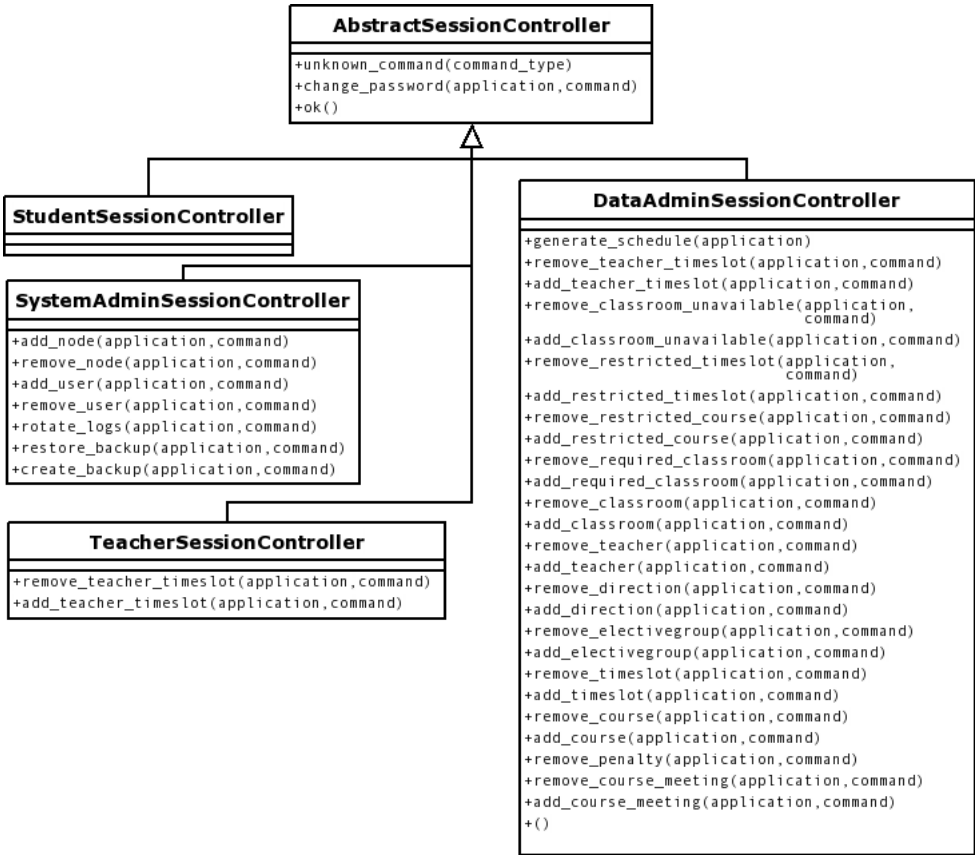
Figuur 9: AbstractSessionFactory



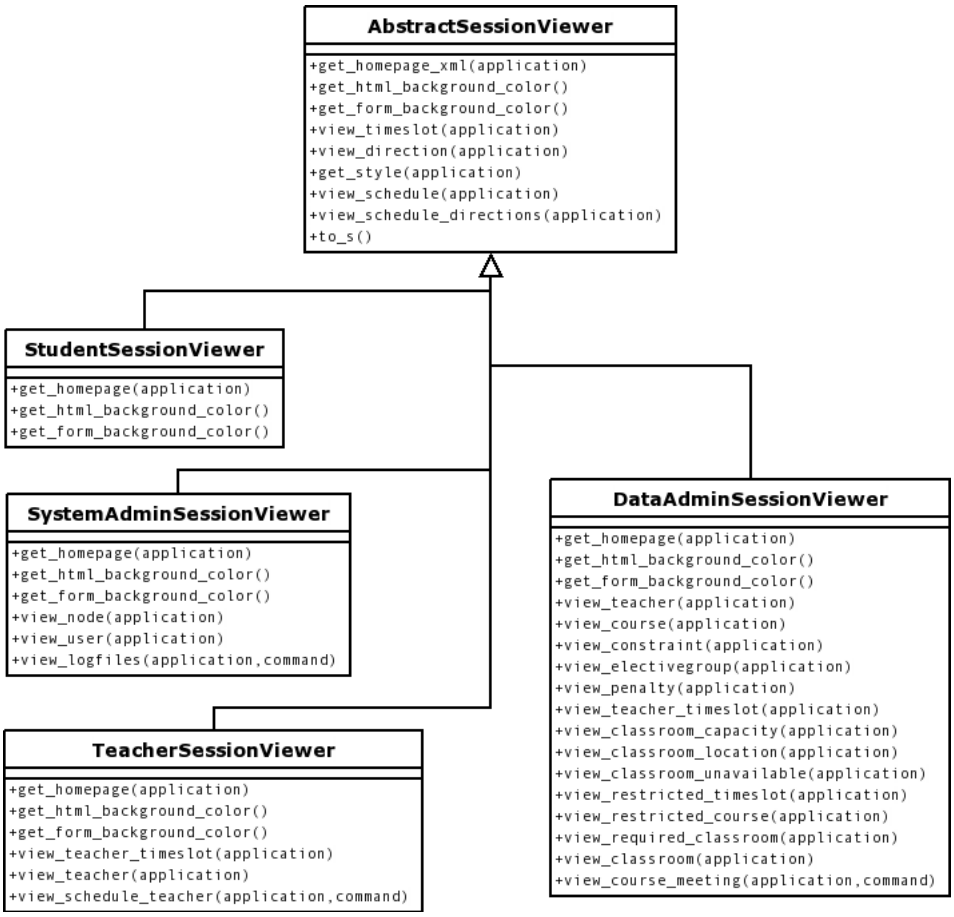
Figuur 10: SessionFactory



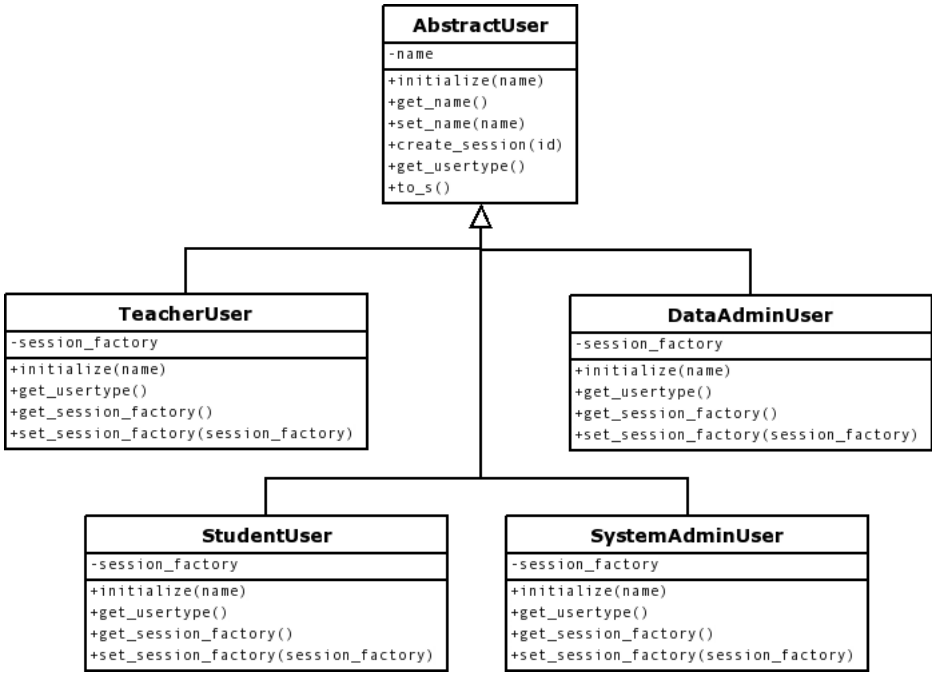
Figuur 11: Session



Figuur 12: SessionController



Figuur 13: SessionViewer



Figuur 14: Ontwerp User

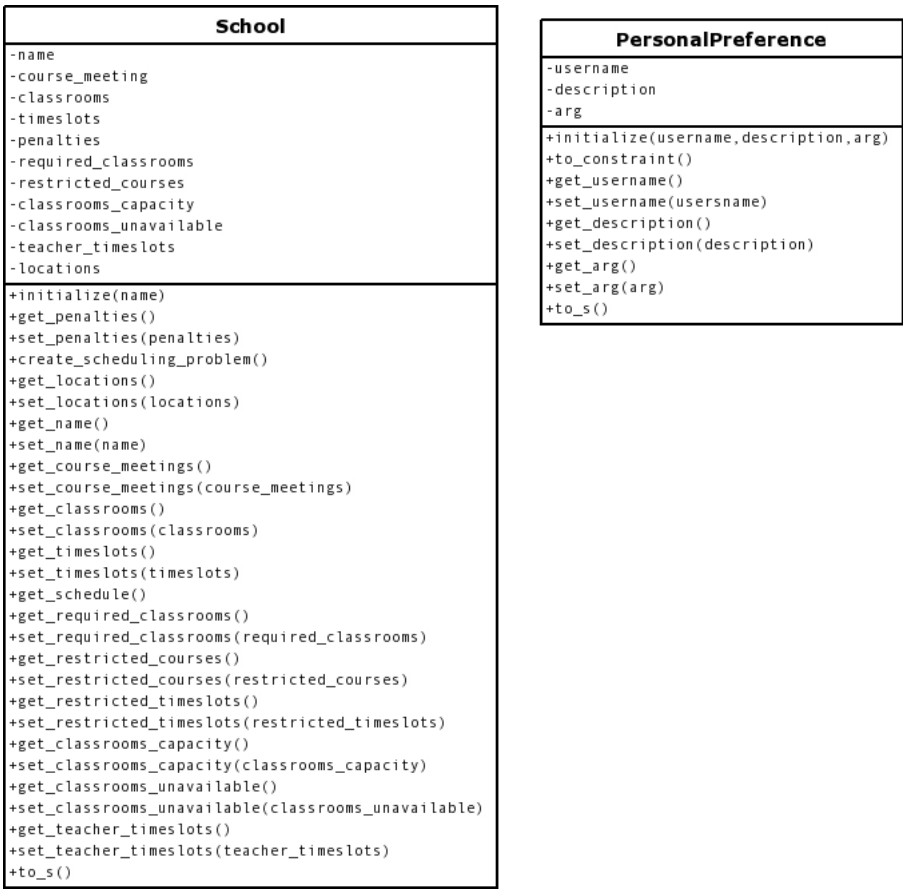
3.3.3 Overzicht schedulen

Het voorstellen van het lessenrooster probleem gebeurt op 3 niveaus. Dit is een kort overzicht, verder wordt op elk niveau dieper ingegaan.

Domeinspecifiek schedulingprobleem Dit niveau is specifiek voor het opstellen van lessenroosters voor scholen. In dit geval gaat het over lokalen, vakken, leerkrachten, voorkeuren, enz.

Algemeen schedulingprobleem Op dit niveau wordt het probleem voorgesteld als een abstract scheduling probleem. Niet enkel het opstellen van lessenroosters, maar ook andere scheduling problemen (bv het plaatsen van containers in een haven) kunnen hiermee voorgesteld worden. Een algemeen scheduling probleem wordt voorgesteld mbv resources,consumers,constraints en improvements. Deze klassen geven een uniforme toegang tot de domein-specifieke klassen (**Adapter/Wrapper** design pattern).

Evolutie van populaties Op dit niveau wordt het probleem voorgesteld als het zoeken van het maximum van een wiskundige functie. Dit gebeurt aan de hand van een genetisch algoritme dat een populatie met verschillende chromosomen laat evolueren volgens een fitheidsfunctie. Het eigenlijke scheduling algoritme kan eenvoudig gewijzigd worden (**Strategy** design pattern).



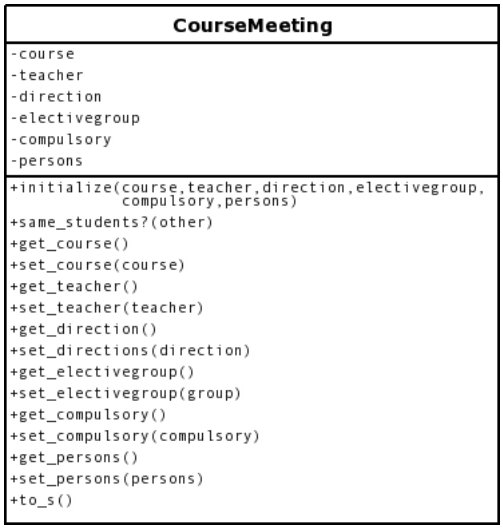
Figuur 15: Ontwerp School

3.3.4 Domeinspecifiek schedulen

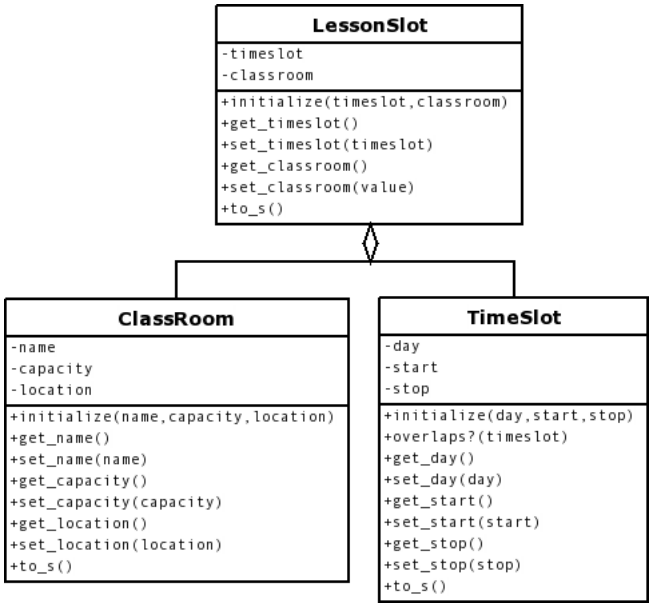
Al de domein-specifieke informatie in verband met het op te stellen lessenrooster zit in een *School* object. Dit object bevat informatie over alle lokalen, lesuren, vakken, vakbijeenkomsten, enz. Bij het aanmaken van een lessenrooster wordt eerst alle data uit de database ingelezen en aan het School object toegevoegd.

Op dit niveau bestaat het scheduling probleem er uit om een *LessonSlot* object toe te kennen aan elk *CourseMeeting* object. Een LessonSlot object bestaat uit een lokaal en een tijdsspanne waar een les kan doorgaan. Een CourseMeeting object bevat informatie over het vak, de leerkracht, het aantal studenten, de studierichting, enz.

Het School object kan al zijn informatie omzetten naar een algemeen *SchedulingProblem*.



Figuur 16: Ontwerp CourseMeeting



Figuur 17: Ontwerp Lessonslot

3.3.5 Algemeen schedulen

Om het programma zo herbruikbaar mogelijk te maken wordt het lessenrooster probleem omgezet naar een algemeen scheduling probleem. Een *SchedulingProblem* bestaat uit *Consumer* objecten die elk een *Resource* object moeten toegewezen krijgen. Afhankelijk van het soort probleem kunnen Resources al dan niet hergebruikt worden, dit is instelbaar in de Scheduler. Een *SchedulingSolution* bestaat uit *Combinations* die ieder een Consumer en een Resource bevatten.

Constraints Eigenschappen waaraan de oplossing moet voldoen kunnen worden ingesteld met behulp van *Constraint* objecten (Figuur 18) Een constraint bevat een naamloze procedure die een score (getal) geeft aan een mogelijke oplossing. Als de constraint de voorgestelde oplossing goed vindt, geeft deze een positief getal terug. Indien de constraint een harde eis voorstelt die geschonden wordt, geeft deze een negatief getal terug. Indien een constraint geen mening over een oplossing heeft geeft deze 0 terug. De score van een oplossing is de som van alle scores van de verschillende constraints.

Resources Een gewone scheduler kan zich enkel baseren op de score van oplossingen om deze te verbeteren. Men kan echter domein-specifieke kennis toevoegen aan de scheduler met *Improvement* objecten (Figuur 19). Deze objecten onderzoeken een voorgestelde oplossing en kunnen er gerichte verbeteringen aan maken. Het voordeel van deze methode is dat de gevonden oplossingen van hogere kwaliteit zijn. Het nadeel is dat het uitvoeren van Improvements extra tijd kost. Daarom worden deze best enkel gebruikt om verbeteringen aan te brengen die anders niet snel gevonden kunnen worden, bijvoorbeeld een les die slechts in 1 lokaal kan plaatsvinden.

School De informatie nodig om een lessenrooster op te stellen zit in een *School* object (Figuur 19). Dit object bevat informatie over de leerkrachten, vakken, lokalen, voorkeuren, ... Wanneer een lessenrooster moet aangemaakt worden, wordt aan het School object gevraagd om een algemeen scheduling probleem aan te maken. Dit SchedulingProblem bevat dan de nodige Resources, Consumers, Constraints en Improvements.

In het geval van het lessenrooster probleem is de overeenkomst als volgt:

- **Resource** Lokaal met lesuur (LessonSlot)
- **Consumer** Vakbijeenkomst (CourseMeeting)

Werking Constraints Sommige constraints kunnen automatisch opgelost worden door de keuze van Resources en Consumers. Vermits een *LessonSlot* een combinatie van lokaal met lesuur bevat, kan in een gevonden oplossing nooit een lokaal door meerdere lessen tegelijk gebruikt worden. Het gebruik van LessonSlot objecten als Resources maakt het ook mogelijk om de beperking dat sommige lokalen op bepaalde momenten onbeschikbaar zijn, eenvoudig op te lossen: men haalt de overeenkomstige Resources uit het SchedulingProblem alvorens men het aan de Scheduler geeft.

Deze methode werkt echter niet voor alle constraints. Daardoor worden er Constraint objecten gebruikt die tijdens het scheduling een score

toekennen aan een voorlopige oplossing. Voor elk soort beperking wordt er een nieuw type constraint aangemaakt dat het *AbstractConstraint* als ouder heeft. Bij het aanmaken van een probleem-specifieke constraint hoeft men enkel de prioriteit en eventuele extra parameters mee te geven. Het Constraint object maakt dan zelf een *Procedure* object aan (analoog aan *Lambda* in Lisp en Scheme) dat aan het ouder-object wordt meegegeven. Door deze aanpak kunnen constraints gebruik maken van informatie die de scope van het School object zit.

Een overzicht van de verschillende beperkingen en hun overeenkomstige Constraints (zie ook Figuur 18).

CapacityConstraint Er kunnen slechts een beperkt aan studenten in een klaslokaal.

ClassroomDistanceConstraint Als een leerkracht/student direct na elkaar meerdere vakken heeft, liggen de klaslokalen liefst niet te ver uit elkaar. De campus is opgedeeld in discrete stukken (bv verdiepingen van gebouwen), lokalen die in een zelfde zone liggen worden verondersteld om dicht bij elkaar te liggen.

CompulsoryConstraint De verplichte vakken van een student mogen niet overlappen.

ElectiveConstraint Keuzevakken uit dezelfde keuzevakgroep worden tegelijk gegeven.

LastTimeslotsConstraint De laatste lesuren worden liefst niet gebruikt, het laatste gewenste lesuur is instelbaar.

LunchBreakConstraint Elke studierichting/docent moet rond de middag een lunch-pauze hebben van minstens 1 uur.

OneCourseADayConstraint Een student heeft liefst van elke vak slechts 1 les per dag.

RequiredClassroomConstraint Sommige vakken kunnen enkel in bepaalde lokalen doorgaan (bv. natuurkunde in natuurkunde-labo).

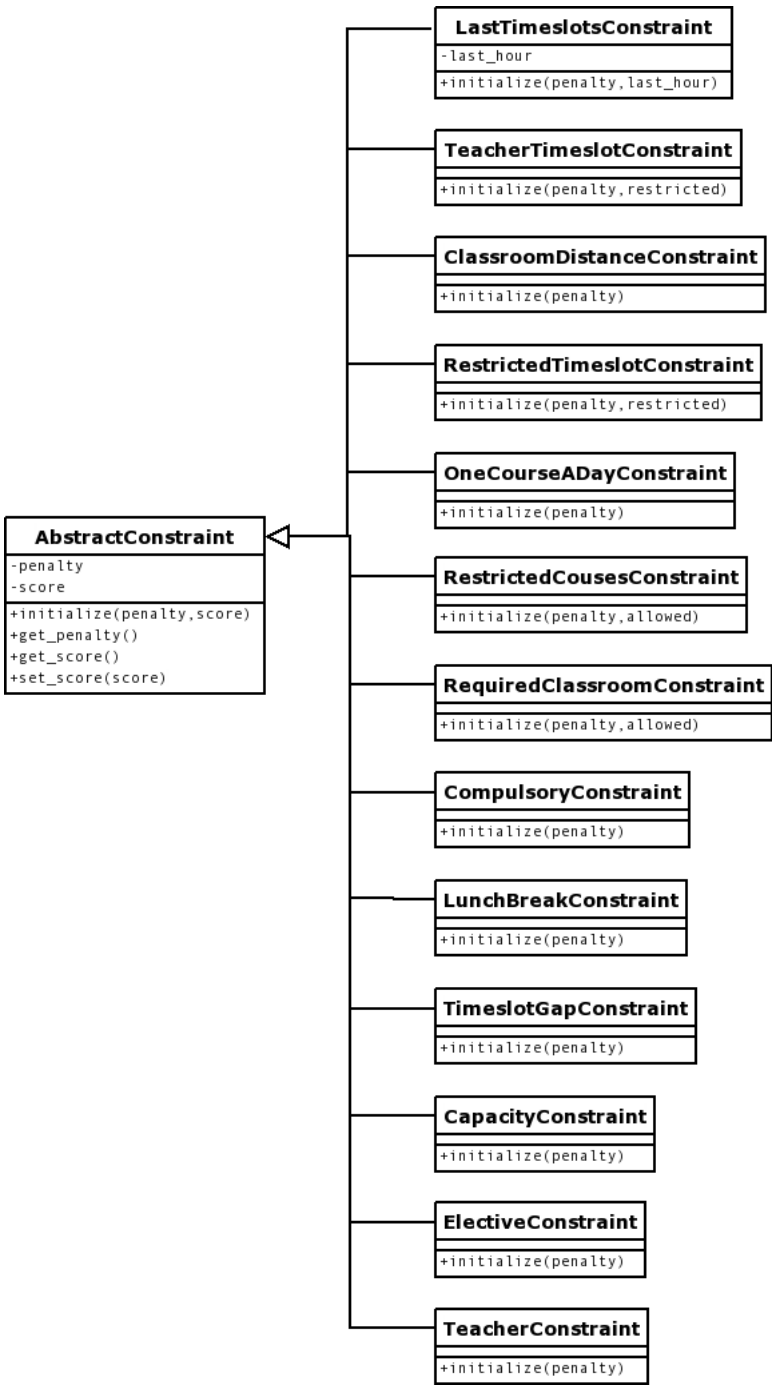
RestrictedCoursesConstraint Sommige lokalen kunnen enkel voor bepaalde vakken gebruikt worden (bv. zwembad)

RestrictedTimeslotConstraint Sommige richtingen kunnen op bepaalde momenten geen les krijgen.

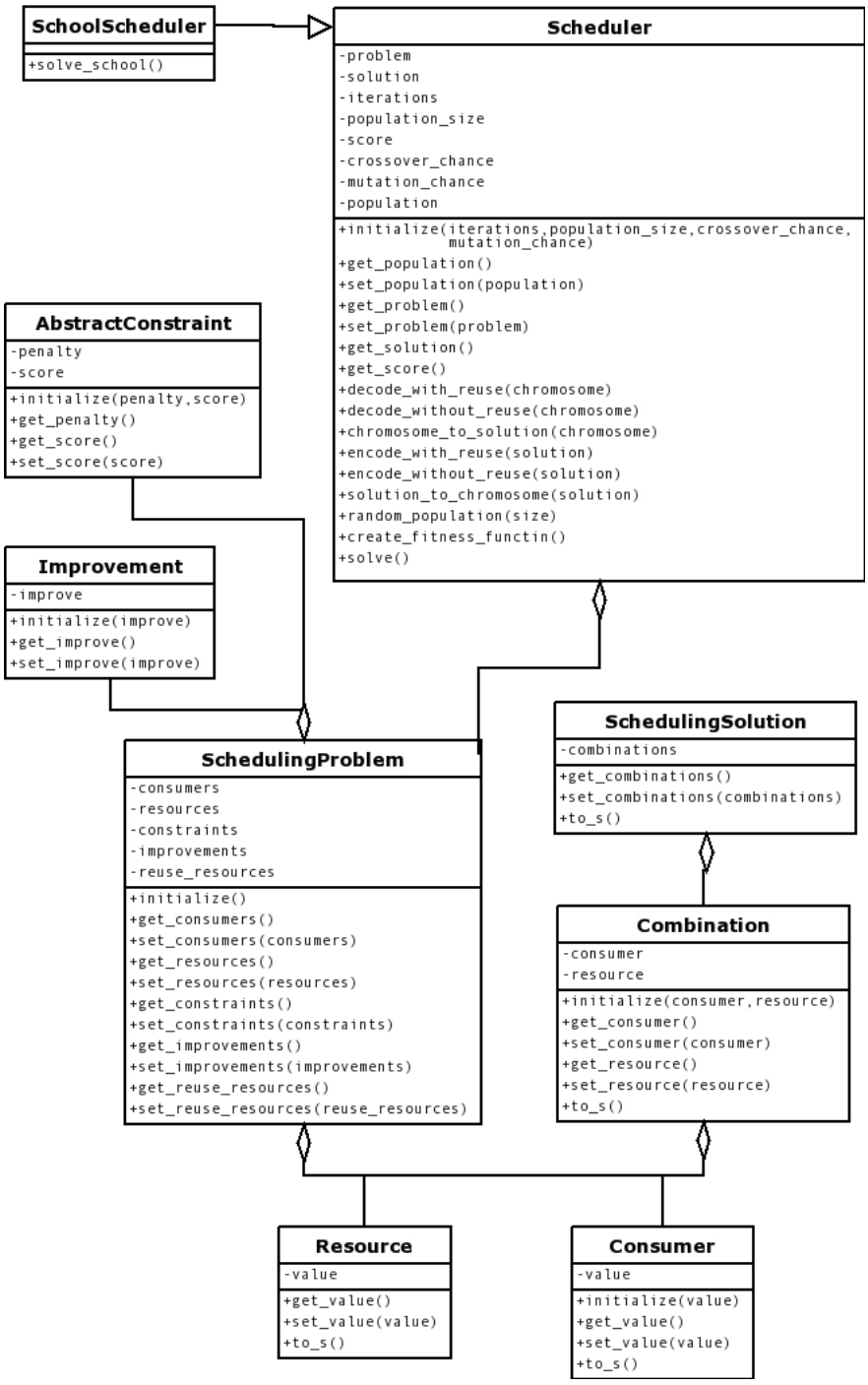
TeacherTimeslotConstraint Sommige docenten willen (voorkeur) of kunnen (eis) op bepaalde momenten geen les geven.

TeacherConstraint Een docent kan slechts 1 les tegelijk geven.

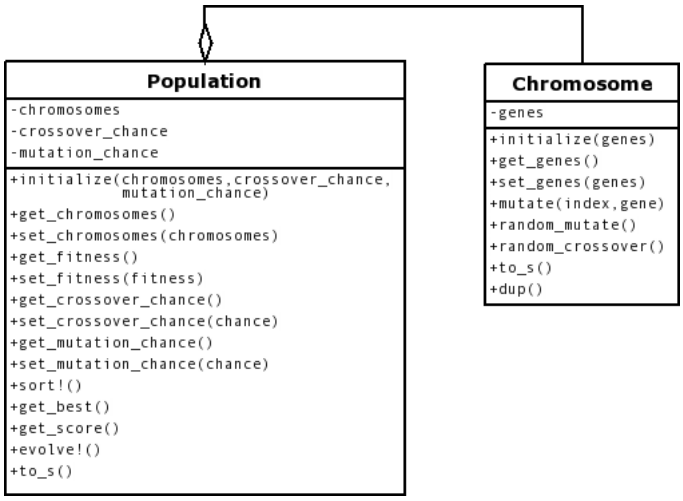
TimeslotGapConstraint Spring-uren dienen vermeden te worden



Figuur 18: Ontwerp Constraints



Figuur 19: Ontwerp Scheduler



Figuur 20: Ontwerp Population en Chromosome

3.3.6 Evolutie

Een Scheduler stelt een object voor dat een SchedulingProblem kan oplossen. Resources, Consumers, Constraints en Improvements zijn niet afhankelijk van het gebruikt algoritme.

Wanneer een scheduler een probleem begint op te lossen, moet alle probleem-informatie dus eerst omgezet worden naar datastructuren die door het scheduling algoritme gebruikt kunnen worden. In dit geval betekent dit dat de informatie van een SchedulingProblem gecodeerd moet worden als een *Chromosome*. Met elk Chromosome object komt een SchedulingSolution overeen.

Een Chromosome stelt een levend individu voor. Afhankelijk van een fitheidsfunctie krijgt het een score toegewezen. Deze fitheidsfunctie wordt aangemaakt met de Constraints van het SchedulingProblem.

Een *Population* bestaat uit meerdere individuen. Elke generatie sterven de individuen met de slechtste chromosomen. De beste individuen planten zich met elkaar voort waardoor er een kruising (cross-over) van genetisch materiaal ontstaat. Dit betekent dat er chromosomen ontstaan die voor de ene helft bestaan uit het chromosoom van hun vader en voor de andere helft uit het chromosoom van hun moeder. Hoe beter een individu is, hoe meer kans het heeft zich voort te planten. Om de diversiteit van het genetisch materiaal te waarborgen treden er ook mutaties op. Deze mutaties wijzigen een willekeurig allel op het chromosoom van een individu.

Door de evolutie verbeteren de scores van de individuen. Na een aantal generaties stopt het algoritme en wordt het beste Chromosoom uit de Populatie als antwoord teruggegeven. Het aantal generaties, de kans op mutatie en de kans op cross-over kunnen ingesteld worden. Deze instellingen staat in een configuratie-bestand en kunnen opgevraagd worden bij de ConfigManager.

3.3.7 Parallellisatie

Vermits het oplossen van een schedulingprobleem reken-intensief is, kan het systeem een cluster van nodes gebruiken als extra rekenkracht. Deze nodes kunnen op verschillende computers staan die met een netwerk verbonden, bv. een computerzaal van een school. Men kan ook meerdere nodes op eenzelfde computer starten. Vermits een node een afzonderlijk proces is, heeft dit als voordeel dat er op multiprocessor computers gebruik kan worden gemaakt van meerdere processoren. Dit gebeurt immers niet bij normale threads in een Ruby programma, omdat Ruby geen native threads ondersteunt.

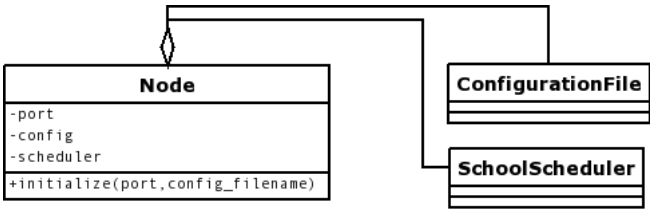
Het grootste probleem bij het parallelliseren van een programma is de communicatie en de synchronisatie tussen de verschillende nodes. We nemen aan dat de netwerkverbindingen van de cluster onbetrouwbaar en traag zijn. Hierdoor moeten we dus rekening houden met nodes die opeens onbereikbaar kunnen worden. Bij het ontwerpen en implementeren van de parallelisatiemogelijkheden van het programma had robuustheid de belangrijkste prioriteit.

Als men gebruik wil maken van de parallellisatie, moet men op elke computer die men wil gebruiken een node starten. Dit kan met doen door het commando *ruby run-node poort-nummer* uit te voeren op alle computers van de cluster. Vervolgens kan de systeembeheerder via de web-interface de netwerk-adressen van de nodes instellen. Als men nu een nieuw rooster aanmaakt worden deze nodes ook gebruikt om een oplossing te zoeken. Tijdens elk moment mogen er nodes en netwerkverbindingen uitvallen in de cluster zonder dat de master crasht.

Wanneer de ScheduleManager opdracht krijgt om een nieuw rooster op te stellen start hij eerst een Distributed Ruby service. Daarna stuurt hij een copy van het School object naar elke node. Om de nodes afzonderlijk te laten werken, wordt er dus gebruik gemaakt van het **Prototype** design pattern. Omdat de netwerk-topologie niet gekend is kan men deze informatie niet broadcasten. Vervolgens krijgt elke node de opdracht om te beginnen rekenen en start de master zelf ook lokaal een scheduler. Na een ingestelde tijd vraagt de master aan elke node de score van zijn beste oplossing. Dit vergt weinig communicatie vermits de score slechts een getal is. De oplossing met de hoogste score wordt doorgestuurd naar de master, die deze als resultaat teruggeeft.

Om de communicatie tussen de nodes en de master zo klein mogelijk te houden wordt het School object doorgestuurd en niet het door het School object gegeneerde SchedulingProblem object. Dit komt omdat het SchedulingProblem Constraints bevat, de Constraints bevatten een score-functie die aangeroepen wordt bij het beoordelen van voorgesteld oplossingen. Deze score-functies gebruiken informatie uit het School object (bv. een hash-tabel met voorkeur-uren van leerkrachten). Omdat de score-functie aangemaakt worden in het School object kunnen ze aan deze informatie in de scope van het School object. Wanneer enkel het SchedulingProblem object doorgestuurd wordt naar de nodes, verwijst de scope van de score-functies naar de scope van het School object dat op de master is achtergebleven. Bij het aanroepen van de score-functie op de nodes detecteert het Distributed Ruby mechanisme dit en gaat de node de informatie opvragen bij de master. Dit zorgt voor veel netwerkverkeer en veroorzaakt een bottleneck bij de master.

De oplossing voor dit probleem is om heel het School object door te sturen van de master naar de nodes, zodat elke node lokaal zijn eigen



Figuur 21: Ontwerp Node

SchedulingProblem kan genereren en niet meer moet communiceren met de master tijdens het rekenen. De oorzaak van dit probleem was niet eenvoudig te achterhalen en moest gebeuren door met een packetsniffer (Ethereal [10]) de communicatie tussen de master en de client af te luisteren en manueel de geserialiseerde data te analyseren.

3.4 Data

De datalaag is verantwoordelijk voor het beheren van de data. Ze krijgt aanvragen binnen van de applicatielaag in XML formaat. Na deze te verwerken en uit te voeren wordt de gevraagde informatie in XML formaat terug gestuurd naar de applicatielaag.

Intern gebruikt de datalaag een SQL server om de data op te slagen in relationele tabellen. De datalaag zorgt voor de joins en andere operaties op de tabellen zodat de applicatielaag het database schema niet moet kennen.

Bij de naamgeving van de tabellen is de volgende conventie gehanteerd:

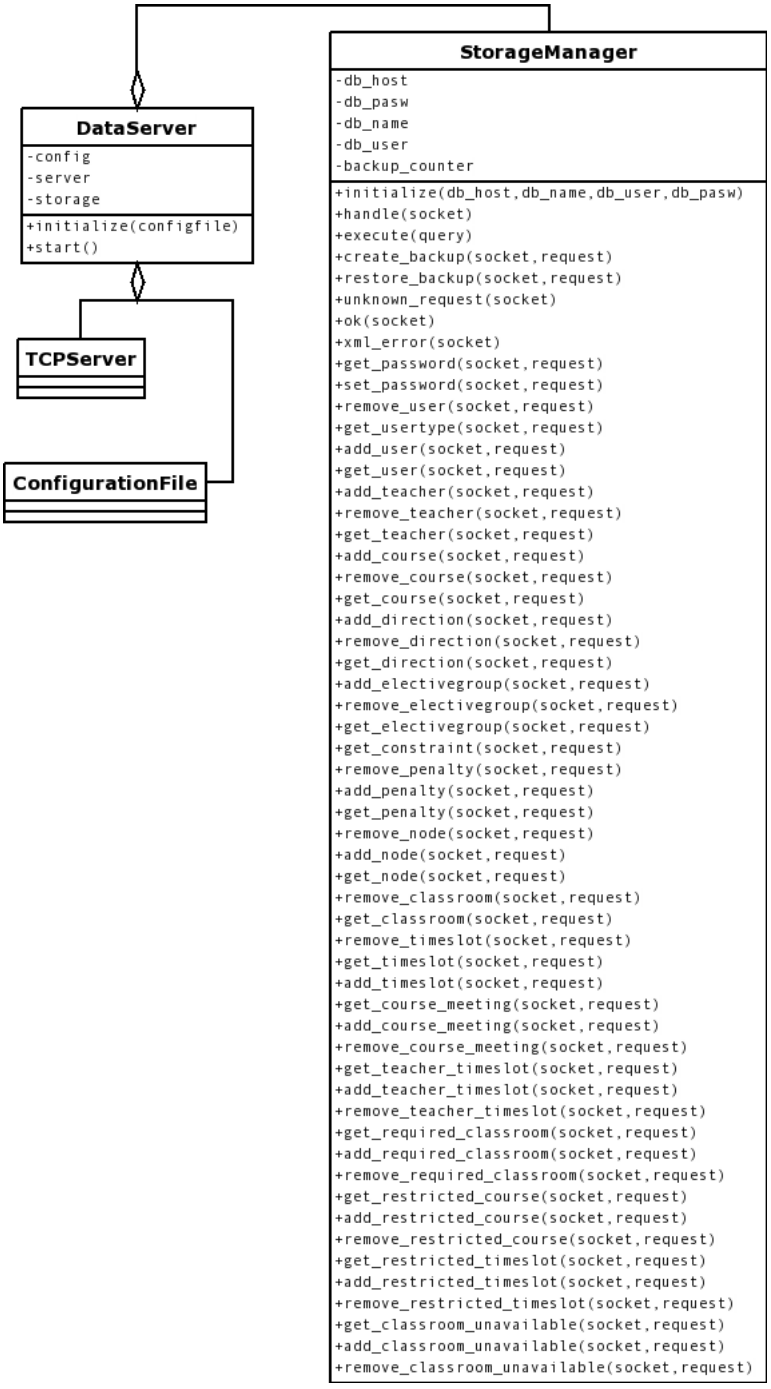
- tabellen met informatie over de toepassing hebben prefix **Data**
- tabellen met informatie over de school hebben prefix **School**
- tabellen met informatie over constraints hebben prefix **Constraint**
- tabellen met informatie over het systeem hebben prefix **System**

De database bestaat uit de volgende tabellen:

- **DataConstraintsClassroomsUnavailable**
  (*Id,ClassroomId,TimeslotId*)
- **DataConstraintsPenalties**
  (*Id, Constraint, Penalty*)
- **DataConstraintsRequiredClassrooms**
  (*Id, CourseId, ClassroomId*)
- **DataConstraintsRestrictedCourses**
  (*Id, ClassroomId, CourseId*)
- **DataConstraintsRestrictedTimeslots**
  (*Id, DirectionId, TimeslotId*)
- **DataConstraintsTeacherTimeslots**
  (*Id, TeacherId, TimeslotId, Score*)
- **DataSchoolClassrooms**
  (*Id,Name,Capacity,Location*)
- **DataSchoolCoursemeetings**
  (*Id, CourseId, TeacherId, DirectionId, ElectivegroupId, Compulsory, Persons*)

- **DataSchoolCourses**
  (*Id, Name*)
- **DataSchoolDirections**
  (*Id, Name*)
- **DataSchoolElectivegroups**
  (*Id, Name*)
- **DataSchoolTeachers**
  (*Id, Name*)
- **DataSchoolTimeslots**
  (*Id,Day,Start,Stop*)
- **SystemAccounts**
  (*Id, Username, Usertype, Password*)
- **SystemNodes**
  (*Id, Name*)

De datalaag is ook in staat om backups van de database te maken en deze terug in te laden. Hiervoor gebruikt het de functionaliteit van mysqldump [5].



Figuur 22: Ontwerp Datalaag

4 Wijzigingen design

Tijdens het implementeren zijn er wijzigingen aan het oorspronkelijk design uitgevoerd. Dit is een overzicht van de aanpassingen en hun verantwoording.

4.1 Genetisch algoritme

Oorspronkelijk Lokalen en lesuren apart toekennen.

Huidig Combinatie (lokaal,lesuur) als 1 resource beschouwen.

Verantwoording Door een aangepaste codering te gebruiken, kan er aan bepaalde constraints automatisch voldaan worden. Dit idee is afkomstig uit [7, MIC99].

4.2 Database schema

Oorspronkelijk Volgens het oorspronkelijk design werden de gegevens opgeslagen in de database als geserialiseerde objecten.

Huidig Er wordt een conventioneel schema gebruikt.

Verantwoording Het oorspronkelijk ontwerp had volgende nadelen:

- moeilijk leesbaar/debugbaar voor mensen
- er kan geen andere programmeertaal gebruikt worden

4.3 PageGenerators

Oorspronkelijk Voor elke pagina afzonderlijk code schrijven.

Huidig PageGenerators geschreven

Verantwoording

- bij wijzigingen moesten alle pagina's afzonderlijk aangepast worden
- uiterlijk en gedrag van pagina's is altijd consistent

4.4 Session id

Oorspronkelijk Volgnummers voor session idś.

Huidig MD5 hashes

Verantwoording veiligheid

5 Conclusie

Tijdens het uitvoeren van dit project ben ik tot de conclusie gekomen dat het schrijven van geparallelliseerde code vereenvoudigd wordt door de Distributed Ruby bibliotheek. Het debuggen daarentegen vergt meer tijd vermits men gebruik moet maken van packet sniffers om fouten te achterhalen. Automatisatie heeft dus als nadeel dat men de controle over de details verliest. Het Prototype design pattern kan gebruikt worden om de communicatie tussen nodes te minimaliseren.

Vermits alle vereisten uit het SRS geïmplementeerd zijn voor het einde van de deadline, kunnen we besluiten dat het project gelukt is.

Referenties

[1] Ruby, The Object Oriented Scripting Language, http://www.ruby-lang.org/en/

[2] Design Patterns, E. Gamme, R. Helm, R. Johnson, J. Vlissides, 1995, Addison Wesley, Indianapolis

[3] Apache HTTP Server Project, http://httpd.apache.org

[4] MySQL, http://www.mysql.com

[5] mysqldump, A Database Backup Program, http://dev.mysql.com/doc/refman/5.0/en/mysqldump.html

[6] Distributed Ruby, http://www.rubycentral.com/articles/drb.html

[7] Genetic Algorithms + Data Structures = Evolution Program, Z. Michalewicz, 1999, Springer Verlag, New York

[8] An Introduction to Genetic Algorithms, M. Mitchell, 2002, The MIT Press, Massachusetts

[9] Artificial Intelligence, A Modern Approach, S. Russel, P. Norvig, 2003, Pearson Education, New Jersey

[10] Ethereal, A Network Protocol Analyzer, http://www.ethereal.com