De ontwikkeling van wijkrouter software

Afstudeerverslag
In opdracht van

Door	: Erik Koster
Studentnr	: 20020326
Versie	: 1.0
Datum	: 10-06-2005
Afstudeerperiode	: 2005-1.1
Opleiding	: I&I / IVIT 3
Richting	: Infrastructuur Ontwikkeling
Startdatum	: 7 februari 2005
Einddatum	: 10 juni 2005

Deze pagina is opzettelijk leeg gelaten.

I Referaat

Afstudeeropdracht van E.R.Koster. Eindverslag betreffende de ontwikkeling van een Linux router applicatie.

Dit verslag beschrijft de totstandkoming van een wijkrouter in opdracht van Lijbrandt Telecom Nederland B.V. Het verschaft inzicht in de keuzes die gemaakt zijn en de manier waarop de keuzes zijn vormgegeven.

Descriptoren:

- Lijbrandt Telecom
- Wijkrouter
- Perl
- PostgreSQL
- 3-Tier model

II Voorwoord

Dit is het eindverslag dat de afstudeerperiode vanaf 7 Februari tot en met 10 Juni bestrijkt. Hierin is beschreven hoe ik te werk ben gegaan om het project tot een goed einde te brengen en zal ik mijn activiteiten beschrijven om zo een duidelijk beeld te scheppen van de denkwijze die geleid heeft tot het eindproduct.

Tevens wil ik van deze gelegenheid gebruik maken om mijn begeleider Dhr M.J. Schreuder bedanken voor zijn hulp bij de obstakels die ik ben tegen gekomen tijdens dit project en de oplossingen die hij hiervoor heeft aangedragen. Ook heb ik veel ondersteuning gekregen van Dhr J. Vermeer. Hij was voornamelijk mijn technisch aanspreekpunt voor PERL specifieke problemen die ik heb ondervonden.

Hillegom, Juni 2005

Erik Koster

Inhoudsopgave

1. Inleiding

Dit verslag heb ik geschreven voor de examinatoren en de gecommitteerde van de Haagse Hoge School zodat mijn afstudeerproject beoordeeld kan worden. Verder wil ik door middel van dit verslag inzicht geven in de afwegingen en keuzes die ik heb gemaakt om dit project te kunnen voltooien.

De hoofdstukken in dit verslag zijn zodanig ingedeeld om de verschillende fases inzichtelijk te maken die dit project hebben gekend, daarnaast zijn de verschillende fases onderverdeeld om de specifieke werkzaamheden binnen deze fases te verduidelijken en het proces te beschrijven.

Hoofdstuk 2:
- Dit hoofdstuk geeft een beschrijving van het bedrijf waar ik dit afstudeerproject uitvoerde, een beschrijving van mijn positie binnen het bedrijf en de inhoud van het project.

Hoofdstuk 3:
- Dit hoofdstuk beschrijft de probleemstelling en de doelstelling die geleid heeft tot de opdracht die ik uitvoer binnen dit project.

Hoofdstuk 4:
- Dit hoofdstuk gaat dieper in op de keuzes die ik heb gemaakt tijdens de aanvang van deze projectperiode. Ook zal ik aangeven wanneer en welke fundamentele wijzigingen hebben plaatsgevonden binnen het project.

Hoofdstuk 5:
- Dit hoofdstuk beschrijft hoe ik mij georiënteerd heb zodat het doel van dit project mij duidelijk werd. Hiermee wordt de omvang en de diepgang van dit project vastgesteld.

Hoofdstuk 6:
- Dit hoofdstuk geeft een beschrijving van de definitiefase van het project. Hierin zal ik beschrijven hoe ik aan de informatie ben gekomen die heeft geleid tot het ontwerp.

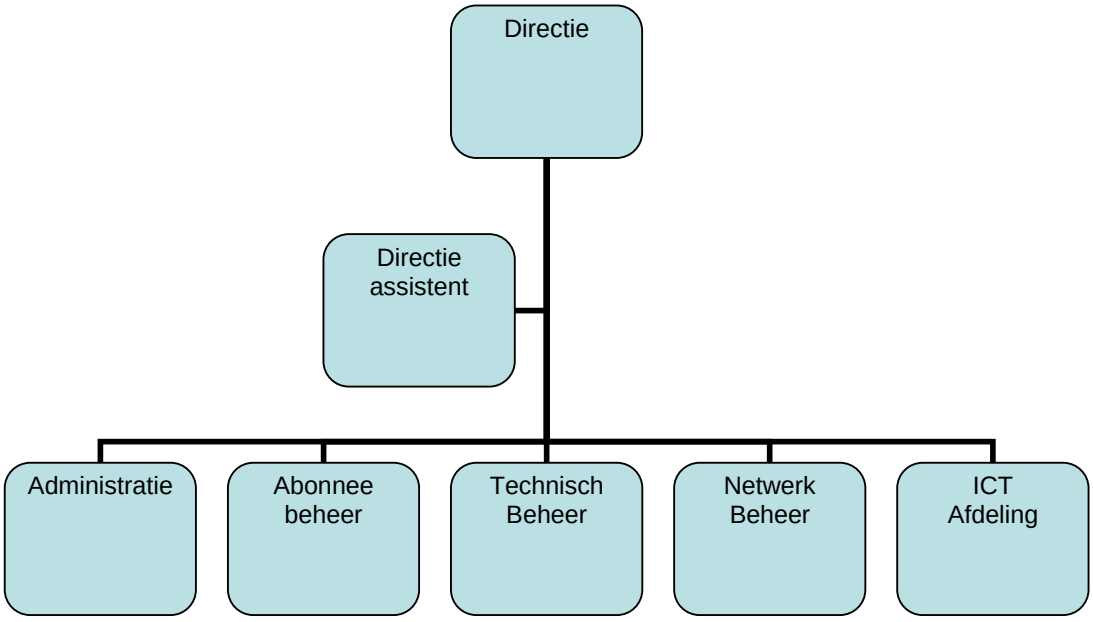
Hoofdstuk 7:
- Dit hoofdstuk beschrijft de ontwerpfase van het project. Ik zal hierin het ontwerp van de wijkrouter software en het proces dat daartoe geleid heeft uitleggen.

Hoofdstuk 8:
- Dit hoofdstuk beschrijft de ontwikkelfase van het project. Hierin zal voornamelijk worden beschreven hoe de software tot stand is gekomen en hoe deze getest zal gaan worden.

Hoofdstuk 9:
- In dit hoofdstuk wordt het gehele project geëvalueerd. Dit is onderverdeeld in een procesevaluatie om gedetailleerd terug te kijken op het proces en een productevaluatie om de balans op te maken van de opgeleverde producten.

2. Lijbrandt Telecom

Lijbrandt Telecom is een telecommunicatiebedrijf dat zich richt op de exploitatie van telecommunicatiediensten en internet via haar eigen glasvezel netwerk door heel Nederland. Lijbrandt Telecom is gevestigd in Hillegom en heeft 30 werknemers. De leiding bestaat uit twee directeuren en een directie assistent die de dagelijkse leiding onder haar hoede heeft. Daaronder vallen de afdelingen: Administratie, Abonnee beheer, Technisch beheer, Netwerk beheer en ICT. Dit heb ik voor de duidelijkheid weergegeven in het onderstaande organigram.



Figuur 2.1: Organigram Lijbrandt Telecom Nederland B.V.

2.1 De historie van Lijbrandt Telecom

Lijbrandt Telecom Nederland B.V. is ooit opgezet door twee mensen. Het bedrijf heette destijds Technotel Installatie B.V. Dit bedrijf verzorgde telecommunicatie installaties in zorginstellingen door heel Nederland. Omdat er toen veel vraag naar was om ook de exploitatie van het aangelegde netwerk op zich te nemen is Technotel Exploitatie opgericht.

De vraag naar exploitatie groeide waardoor het bedrijf doorgroeide naar een volwaardige telecommunicatiediensten aanbieder, die Lijbrandt Telecom Nederland B.V. is gaan heten. Hiernaast is er ook Lijbrandt Telecom Infrastructuren B.V. Dit bedrijf is een zuster organisatie van Lijbrandt Telecom Nederland B.V. en zorgt voor de aanleg en onderhoud van glasvezel netwerken. Het netwerk van Lijbrandt Telecom Nederland B.V. is sinds 1997 uitgegroeid tot 14.000 abonnees en heeft volgens een glasvezel dekkingskaart die Lijbrandt Telecom hanteert een potentieel om 26% van alle huishoudens van Nederland te bereiken.

2.2 De plek van de afstudeerder binnen de organisatie

Ik heb zelf gewerkt binnen de afdeling ICT op het hoofdkantoor van Lijbrandt Telecom in Hillegom. Mijn opdrachtgever binnen het bedrijf is Ing. M.J. Schreuder. Hij is verantwoordelijk voor de implementatie van glasvezelnetwerken binnen Lijbrandt Telecom. Dhr. Schreuder heeft een extensieve ervaring binnen de telecom sector. Dit hielp mij om goed te kunnen inschatten wat voor competenties nodig zijn bij het ontwikkelen van de wijkrouter en hoe het gedrag van abonnees zich verhoudt tot de praktische vertaling hiervan bij het aanbieden van diensten.

Ik heb mij, volgens voorschrift, aan de aanwezigheidsregels gehouden. Waarbij ik om 8.30 uur aanwezig moest zijn tot 17.00 uur. Wanneer ik niet aanwezig kon zijn dan moest ik dat melden bij de directie assistente S. Hofstra. Wat betreft vragen kon ik voor organisatorische zaken terecht bij Dhr M.J. Schreuder, en voor specifieke technische vragen bij Dhr J. Vermeer. Ik rapporteerde aan Dhr M.J. Schreuder en hij is ook voor mijn project eindverantwoordelijk. Verder kon ik van alle faciliteiten gebruik maken die voorhanden waren binnen Lijbrandt Telecom.

3. Opdrachtomschrijving

Tijdens het maken van de opdrachtomschrijving leek het voor het grootste gedeelte al duidelijk te zijn wat er gemaakt moest worden. Het ging om software geschreven in PERL dat taken van een wijkrouter over zou gaan nemen. Echter na gesprekken gevoerd te hebben met de opdrachtgever kwam ik er achter dat er vanaf nu in plaats van de NKF 2700 switches, Cisco 2950 Catalyst switches gebruikt gaan worden. Dit is toch wel een vrij ingrijpend besluit aangezien de interface naar deze switches anders is dan het was.

Dit betekende dat ik een andere interface moest schrijven om de switches te programmeren, in plaats van de manier van configureren die al gebruikt werd.

Nadat Dhr Sikkema en Dhr van Dinter van de Haagse Hoge School langs waren geweest bij Lijbrandt Telecom is er besloten om een aantal zaken te veranderen in de opdrachtomschrijving. Zo zijn de doelstelling en werkzaamheden iets aangepast.

3.1 De probleemstelling

Binnen Lijbrandt Telecom is de kennis over de werking van de wijkrouter bij één persoon belegd. Deze software is ooit gemaakt onder een hoge tijdsdruk en is daardoor ook niet gedocumenteerd. Dit heeft als gevolg dat de software niet overdraagbaar is. De wijkrouter verzorgt essentiële functies voor het leveren van Internet aan abonnees. Verder zijn er functionaliteiten die Lijbrandt Telecom graag in de wijkrouter wil zien, en er nog niet in zitten. Als laatste speelt ook de performance van de wijkrouter een rol. Op dit moment is deze niet voldoende om aan de steeds groter wordende capaciteitsvraag te voldoen. Tevens is de wijkrouter lastig te installeren.

3.2 De doelstelling

Het doel van deze afstudeeropdracht is het ontwikkelen van een nieuwe Linux router waar een eigen applicatie op draait. Hierbij is van belang dat de router en de bijbehorende software makkelijk te implementeren is binnen het netwerk van Lijbrandt Telecom. De wijzigingen die gemaakt worden in de administratieve systemen moeten worden doorgevoerd en de benodigde gegevens dienen over een beveiligde verbinding te worden verstuurd. Ook patches voor de applicatie dienen op een gedistribueerde manier worden toegepast zonder dat iemand elke locatie af moet om deze patch toe te passen. Verder moet de ontwikkeling van de router overdraagbaar worden gemaakt door middel van documentatie.

3.3 Resultaten voor de opdrachtgever

- Plan van aanpak
- Wijkrouter software
- Documentatie over de werking van de wijkrouter software

4. Plan van aanpak

4.1 De projectorganisatie

Bedrijfs informatie:

Lijbrandt Telecom Nederland B.V.

Postadres:
Postbus 308
2180 AH Hillegom

Bezoekadres:
Satellietbaan 20
2181 MH Hillegom

Telefoon : 023-8910000
Fax : 023-8911000

Student:

Naam : Erik Koster
Studentnr : 20020326
Telefoonnr : 0654-268215
E-mail : e.r.koster@student.hhs.nl

Bedrijfsmentor:

Naam : Ing. M.J. Schreuder
Telefoonnr : 023-8910005
E-mail : j.schreuder@lijbrandt-telecom.nl

4.2 De projectmethode Extreme Programming

De keuze voor Extreme Programming was niet erg lastig. Aangezien ik software ging schrijven was het een goed idee om voor een methode te kiezen die het makkelijk maakt om vooral veel door middel van trial and error op te lossen. Extreme Programming voorziet in een groot aantal zaken die voor een klein projectteam(en in dit geval één persoon) handig zijn. Bijvoorbeeld de denkwijze om niet veel vooraf te plannen en om alles simpel en overzichtelijk te houden. Ook al is dit makkelijker gezegd dan gedaan, biedt deze aanpak een beter perspectief voor dit project dan een methode als SDM of DSDM.

Wanneer ik Extreme Programming bijvoorbeeld vergelijk met een methode als I.A.D. of SDM, dan vond ik de methode van Extreme Programming veel efficiënter.
Dit omdat een methode als I.A.D. of SDM veel documentatie oplevert die niet zo relevant is voor dit project. Het gaat hier om pilotontwikkelplannen en fases die over invoering gaan waar ik niets aan had. Ook omdat het doel van mijn project het schrijven van software is, en dus de nadruk vooral op de implementatie ligt.

Daarnaast zijn sommige methodes niet geschikt voor implementatie en bieden zij oplossingen voor andere gebieden in een project. Denk bijvoorbeeld aan een uitgebreide definitiestudie zoals die terug te vinden is in de IAD methode.

Wat mij vooral aansprak is de manier waarop de methode is samengesteld om het programmeurs zo makkelijk mogelijk te maken, om aan de ene kant zich volledig te concentreren op het creatief oplossen van algoritmische problemen en toch zorgt dat er voldaan wordt aan de controleerbaarheid en duidelijkheid die bij een project zo belangrijk zijn. Het voorziet onder andere aan een testfase en betrekt hierbij op een vrij aparte manier de opdrachtgever. Geen ingewikkelde metaplan methodes en workshops die bij grote projecten uiteraard wel handig zijn. Maar een uitermate efficiënte manier om snel resultaten te boeken waarmee tijdens het gehele proces wordt gestreefd naar de beste oplossing voor het probleem. Mede hierdoor en de parate kennis die ik al had opgedaan tijdens de opleiding was dit voor mij de beste keuze voor dit project.

De produkten uit extreme programming die ik zal opleveren zijn onderverdeeld in een aantal fasen. Deze zien er als volgt uit:

- **Definitie fase**
  - Planning
    - Het inplannen van de keuze momenten en deelproducten en het eindproduct.
  - Release plan
    - Hier vertelt de opdrachtgever wat er in de software verwerkt moet worden.
  - User stories
    - Onderdeel van het vaststellen van de eisen voor de wijkrouter software. Door scenario's voor het gebruik vast te stellen.
  - Spikes(optioneel)
    - Kleine simpele oplossingen om te kijken of dit werkt en meer werk waard is.
- **Ontwerp fase**
  - Het ontwerp in Extreme programming dient zo simpel mogelijk te worden gehouden en continu te worden bijgewerkt wanneer dit nodig is. In deze ontwerp fase zal ik producten opleveren die gebruik maken van de UML techniek.
- **Ontwikkel fase**
  - Code (ook wel 'small release' of 'iteratie' genoemd)
    - Dit is het iteratieve deel waarbij er geprogrammeerd wordt (het liefst in de door XP voorgeschreven 'pair programming' opstelling. Dit is echter niet mogelijk aangezien ik dit project alleen uitvoer.). Dit levert een *'small release'* op die feedback levert voor een volgende iteratie.
  - Acceptatie tests
    - Hier test de opdrachtgever of het prototype naar wens is geïmplementeerd.

4.3 Technieken

Om dit project uit te voeren en deelproducten van de methode Extreme Programming op te leveren zal ik gebruik maken van een aantal technieken. De techniek die ik voor een groot gedeelte voor het ontwerp zal gebruiken is UML(Unified Modelling Language). Deze techniek houdt er voor het grootste gedeelte een object georiënteerde benadering op na. Of de software die ik ga schrijven ook object georiënteerd is, is nog niet duidelijk. Echter om op een abstract niveau de software en data modellen te modelleren zal ik gebruik maken van UML.

Een aantal producten van UML zijn:

- Usecasediagram (dit diagram zal gebruikt worden in de definitie fase)
- Usecase scenario's
- Klassendiagram
- Collaboratiediagram

Er zijn nog veel meer modellen die UML voorschrijft maar ik zal alleen van deze modellen gebruik maken.

De methode Extreme Programming schrijft voor dat er veel overleg zal zijn met de opdrachtgever(s). Om dit overleg zo effectief mogelijk te benaderen zal ik gebruik maken van interview technieken om de benodigde informatie te achterhalen.

4.4 Risico's

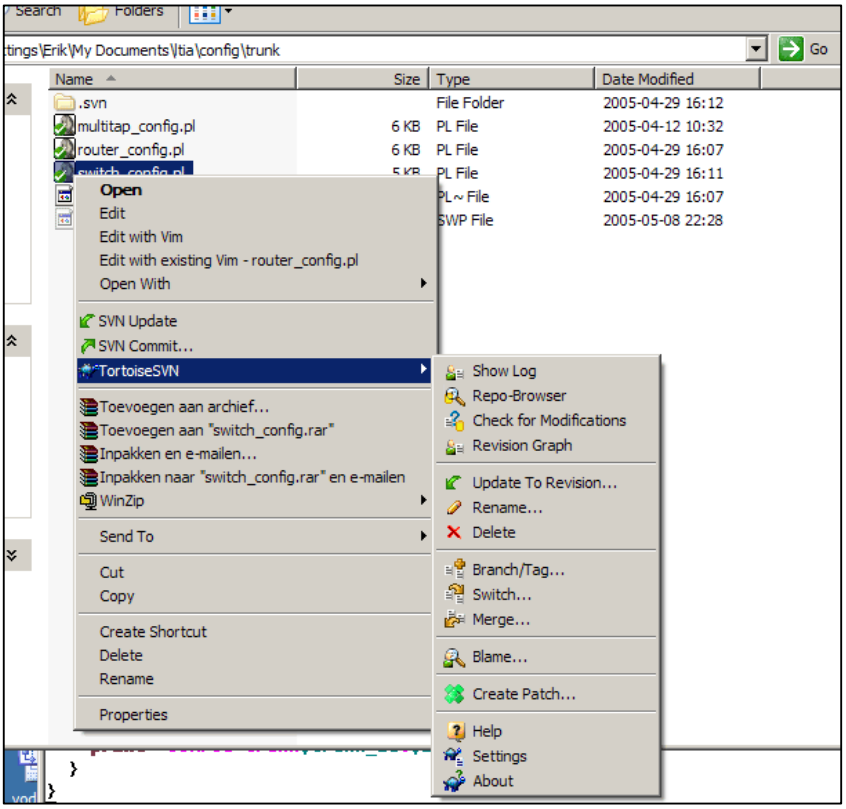
De kans bestond dat ik de gestelde werkzaamheden in de opdrachtomschrijving niet af kreeg wat voor mij een potentieel risico was. Ook bestond de kans dat de opdracht anders ging verlopen dan in eerste instantie was vastgesteld in de opdrachtomschrijving. Naast de omvang was ook mijn parate kennis op het gebied van netwerken niet genoeg om een goede inschatting te kunnen maken of ik dit project binnen de gestelde tijd goed kon afronden. Ik moest mijn kennis daarom uitbreiden en veel communiceren met de opdrachtgever om zo een transparant mogelijke situatie te creëren, waarop beslissingen gebaseerd kunnen worden. Daarnaast heb ik in de loop van dit project prioriteiten gesteld aan de functionaliteit van de software om zo de duidelijkheid binnen het project te verbeteren.

4.5 Versiebeheer

In mijn plan van aanpak heb ik aangegeven dat ik gebruik ging maken van CVS. Dit versie management systeem was ook aanwezig bij Lijbrandt Telecom. Echter aangezien er een aantal projecten lopen bij Lijbrandt Telecom werd dit systeem omgezet in een andere manier van versie management dan CVS. Deze software heet 'Subversion'. Dit is ongeveer hetzelfde als CVS alleen dan met een aantal extra mogelijkheden die het gehele versie management binnen Lijbrandt Telecom een stuk gemakkelijker maken.

Zelf hoefde ik me over de opzet van subversion geen zorgen te maken, dit was al gebeurd en ik hoefde slechts mijn project aan te melden op de SVN(subversion) server om er gebruik van te maken. Als client maakte ik gebruik van TurtoiseSVN. Deze client is een integraal deel van Microsoft Windows Explorer. Hierdoor is het voor mij enorm makkelijk om aan versie management deel te nemen zonder dat hiervoor allemaal aparte programma's moeten worden geïnstalleerd, dat overigens bij het gebruik van CVS wel het geval zou zijn.

Ook voor CVS bestaat een client genaamd WinCVS alleen is deze een stuk onduidelijker dan de Subversion client die ik gebruikte tijdens dit project. De onderstaande afbeelding geeft een impressie van de werking van TurtoiseSVN en Subversion.



Figuur 4.1: Screenshot 'Subversion' user interface

Zoals zichtbaar in deze afbeelding zijn exact dezelfde mogelijkheden als CVS geïntegreerd in de Windows Explorer. Aangezien de software die ik ging schrijven niet met behulp van een complexe IDE[1] werd geschreven was het geen probleem om af te wijken van de CVS[2] standaard en wogen alle andere voordelen van dit Subversion systeem op tegen de nadelen van het afwijken van de standaard manier van werken. Dat was dan ook de enige afweging die ik kon maken wat betreft het versie beheer.

De omzetting van CVS naar Subversion was volledig gedaan door andere mensen binnen Lijbrandt Telecom. Hier heb ik mij niet mee bezig gehouden. Uiteraard heb ik me er genoeg in verdiept om het op de juiste manier te kunnen gebruiken zodat ik me verder op de implementatie van de wijkrouter software kon richten.

4.6 Kwaliteitseisen

De kwaliteitseisen die aan dit project werden gesteld zijn geformuleerd op basis van het **SMART** principe. Dit houdt in:

- **S:** *specifiek* - Is het duidelijk wat er moet gebeuren? En is dit specifiek gedefinieerd.
- **M:** *meetbaar* - Kan het al dan niet behalen van de doelstelling geëvalueerd worden?
- **A:** *acceptabel* - Is de doelstelling acceptabel als zodanig of moet er aangegeven worden dat bepaalde delen van de doelstelling een ambitie zijn. In mijn geval wordt deze factor afgeschermd met het stellen van prioriteiten door middel van het **MoSCoW** principe welke na de definitie studie zullen worden bepaald.
- **R:** *realistisch* - Is de doelstelling haalbaar?
- **T:** *tijdgebonden* - Is het duidelijk wanneer de doelstelling behaald moet zijn? Dit komt vooral terug in de planning welke in dit document is bijgesloten.

In dit document is een invulling gegeven aan de wijze waarop ik aan de beschreven kwaliteitseisen heb voldaan.

4.7 De faciliteiten

De benodigdheden om dit project uit te voeren waren bij Lijbrandt Telecom Nederland B.V. ook al aanwezig. Dit waren:

4.7.1 Hardware

- Een switch
- 1 Server
- 1 Werkstation

[1] Integrated Development Enviroment – Een programmeeromgeving zoals bijvoorbeeld Visual Studio .NET
[2] Concurrent Version System – Het meest gebruikte versiebeheer systeem

4.7.2 Software

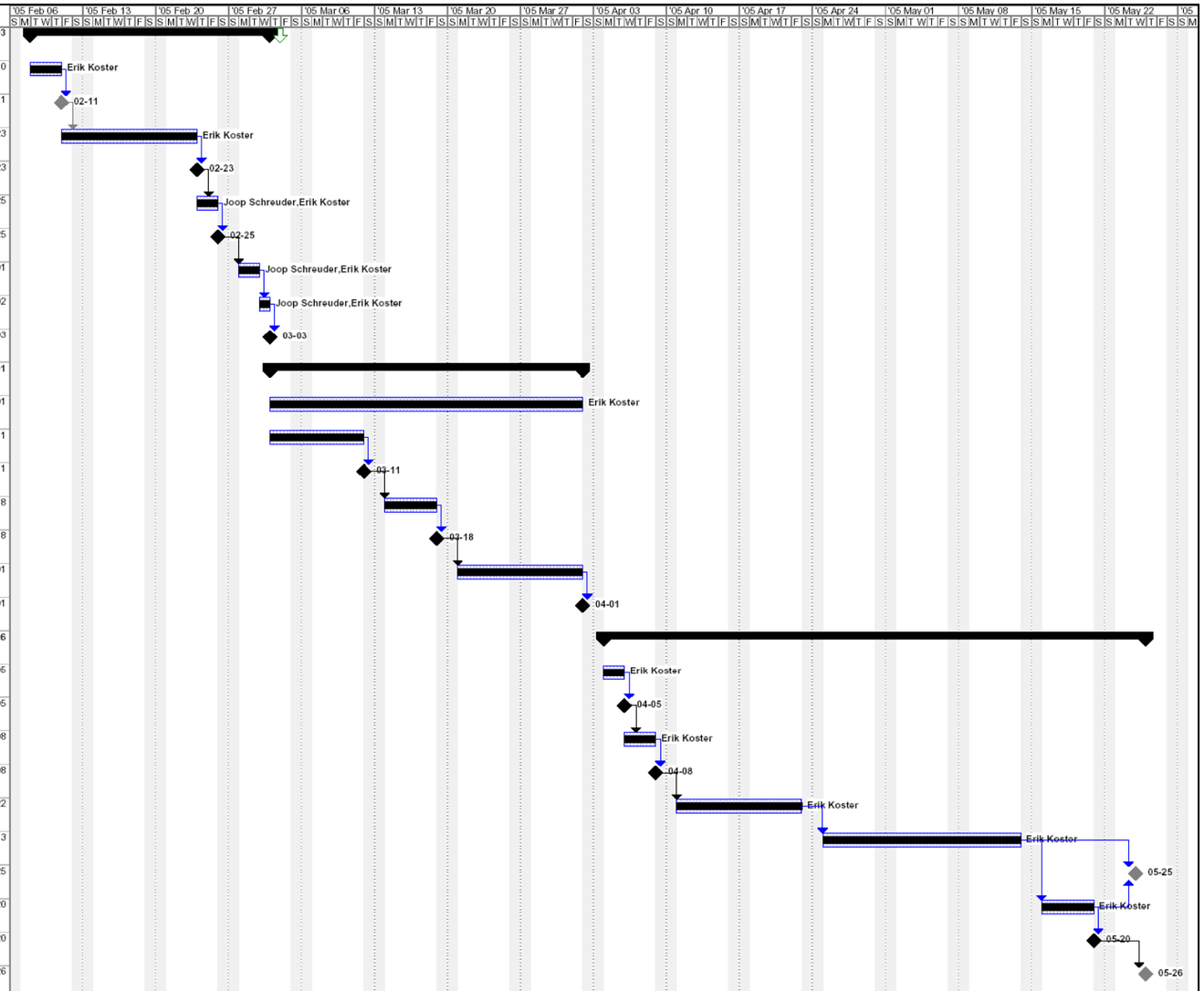
- PostgreSQL(DBMS)
- Fedora Core 3(of een andere linux distributie)
- Microsoft Visio
- Microsoft Powerpoint
- Microsoft Project
- Microsoft Word
- Microsoft Excell

4.8 De planning

De planning zal worden uitgevoerd in de vorm van een Gantt chart. Om deze te generen gebruik ik de software Microsoft Project 2003. In deze Gantt chart zullen ook ‘milestones’ voorkomen om aan te geven welke producten er aan het einde van een periode opgeleverd worden.

Omdat de gebruikte methode een iteratief proces en ook adviseert niet alles van te voren te anticiperen en te plannen, zal de planning continu worden bijgewerkt door middel van recente ontwikkelingen. De planning ziet er als volgt uit.

ID	Task Name	Duration	Start	Finish
1	**Definitiefase**	**17 days**	**Tue 05-02-08**	**Thu 05-03-03**
2	Het schrijven van het plan van aanpak	3 days	Tue 05-02-08	Thu 05-02-10
3	Opleveren plan van aanpak	0 days	Fri 05-02-11	Fri 05-02-11
4	Het onderzoek naar de werking van de huidige wijkrouter software	9 days	Fri 05-02-11	Wed 05-02-23
5	Opleveren rapport werkwijze huidige wijkrouter software	0 days	Wed 05-02-23	Wed 05-02-23
6	Het vaststellen van de user requirements met behulp van user stories	2 days	Thu 05-02-24	Fri 05-02-25
7	Opleveren user stories	0 days	Fri 05-02-25	Fri 05-02-25
8	Het stellen van prioriteiten voor de functionaliteit van de software	2 days	Mon 05-02-28	Tue 05-03-01
9	Het opstellen van een release plan	1 day	Wed 05-03-02	Wed 05-03-02
10	Opleveren releaseplan	0 days	Thu 05-03-03	Thu 05-03-03
11	**Ontwerpfase**	**22 days?**	**Thu 05-03-03**	**Fri 05-04-01**
12	Het ontwerpen van de wijkrouter software	22 days	Thu 05-03-03	Fri 05-04-01
13	Opzetten van use-case scenario's	7 days	Thu 05-03-03	Fri 05-03-11
14	Opleveren use-case scenario's	0 days	Fri 05-03-11	Fri 05-03-11
15	Opzetten klassendiagram	5 days?	Mon 05-03-14	Fri 05-03-18
16	Opleveren klassendiagram	0 days	Fri 05-03-18	Fri 05-03-18
17	Opzetten van een collaboratiediagram	10 days	Mon 05-03-21	Fri 05-04-01
18	Opleveren collaboratiediagram	0 days	Fri 05-04-01	Fri 05-04-01
19	**Ontwikkelfase**	**38 days**	**Mon 05-04-04**	**Thu 05-05-26**
20	Het opzetten van een omgeving waar een werkend prototype getest kan worden	2 days	Mon 05-04-04	Tue 05-04-05
21	Opleveren test omgeving	0 days	Tue 05-04-05	Tue 05-04-05
22	Het opzetten van SVN voor de wijkrouter software	3 days	Wed 05-04-06	Fri 05-04-08
23	Opleveren versiebeheer procedure(SVN) voor de wijkrouter software	0 days	Fri 05-04-08	Fri 05-04-08
24	Iteratie 1: Het programmeren van de DatabaseConnector	10 days	Mon 05-04-11	Fri 05-04-22
25	Iteratie 2: Het programmeren van de linux router applicatie	15 days	Mon 05-04-25	Fri 05-05-13
26	Opleveren wijkrouter documentatie	0 days	Wed 05-05-25	Wed 05-05-25
27	Het uitvoeren van een acceptatie tests	5 days	Mon 05-05-16	Fri 05-05-20
28	Opleveren acceptatie test documentatie	0 days	Fri 05-05-20	Fri 05-05-20
29	Opleveren Wijkrouter software	0 days	Thu 05-05-26	Thu 05-05-26



Project: afstud-planning-0-1
Date: Thu 05-06-09

Task | Split | Progress | Milestone | Summary | Project Summary | External Tasks | External Milestone | Deadline

Page 1

5. Beschrijving van de wijkrouter

De afdeling ICT heeft in 2002 een systeem ontwikkeld genaamd de wijkrouter. Dit systeem wordt door Lijbrandt Telecom gebruikt op locaties waar internet gedistribueerd wordt. Het is een server met hierop eigen software welke mutaties doorvoert op de controlerende onderdelen van het routerende verkeer.

Deze wijkrouter is direct gekoppeld aan L.T.I.A.(Lijbrandt Telecom Internet Administratie) waar een aantal eigenschappen van een internet verbinding ingesteld kunnen worden(snelheid, mailbox, wachtwoorden etc…).

Deze wijkrouter is ooit mede ontworpen door iemand die niet meer werkzaam is bij Lijbrandt Telecom. Daarom zijn ook bepaalde onderdelen erg onduidelijk. In deze afstudeerperiode heb ik mij bezig gehouden met het opnieuw ontwerpen van de wijkrouter. Voor sommige problemen die ik tegenkwam heb ik terug gekeken naar hoe het voorheen was opgelost.

Het gebruik van de wijkrouter varieert van kleine tot middelmatige projecten. Bij grote projecten wordt overgegaan op een volledig hardwarematige benadering. Waarbij grote routers worden gebruikt om de functies van de wijkrouter op zich te nemen. Bij kleinere projecten wordt deze wijkrouter ingezet. Het grootste voordeel hiervan is dat dit een enorm verschil oplevert in de kosten die een project met zich meebrengt. Ook komt het bij deze kleinere projecten vaak voor dat de bandbreedte leverancier niet Lijbrandt Telecom zelf is. Deze Linux router maakt het mogelijk door middel van een kleine investering(alleen een PC, switch en bekabeling) toch mensen internet te bieden en aangesloten te zijn op het Lijbrandt Telecom netwerk.

Om de kosten te drukken en de mogelijkheden van het netwerk op professioneel niveau aan te bieden is de wijkrouter uitstekend geschikt. Het gebruik van een hardwarematige oplossing voor de Linux router is ongeveer 20 maal duurder dan een simpele PC met een Linux distributie. Om dit project dan ook zo rendabel mogelijk te maken wordt hier al snel over gegaan op een Linux router.

Toen ik begon met het onderzoek naar de wijkrouter was er vrijwel niets bekend. Het enige dat er was, was het datamodel van de database waar de wijkrouter informatie uit moest halen. Om een onderzoek te doen naar de werking van de wijkrouter moest ik eerst uitvinden waar ik de benodigde informatie kon verkrijgen.

Als eerste heb ik gesproken met Joost Vermeer. Dit is één van de twee mensen die verantwoordelijk waren voor het ontwerp van de vorige wijkrouter. Ik kwam er al snel achter dat Dhr Vermeer zelf ook niet meer precies wist hoe de wijkrouter werkte. Samen met Dhr. Vermeer heb ik broncode bekeken van de wijkrouter die op dat moment gebruikt werd. Hier was goed te zien dat ook in de broncode geen documentatie was aangelegd over wat er in de software precies gebeurd.

Vervolgens heb ik na veel navraag kunnen vaststellen dat de wijkrouter zelf maar een klein onderdeel is van een heel groot geheel dat er voor zorgt dat internet verkeer op de juiste manier gerouteerd wordt.

In eerste instantie dacht ik dat de wijkrouter informatie uit de administratieve systemen haalde om op basis van die criteria poorten op switches te beheren. Dit bleek niet het geval. De wijkrouter heeft als hoofdtaken:

- Het verstrekken van IP adressen.
- Het toewijzen van een VLAN aan elke verbinding die de wijkrouter routeert.
- Het instellen van de snelheid waarmee elke verbinding verkeer kan doorvoeren(shaping).
- Het bijhouden van enkele QOS[3] services.

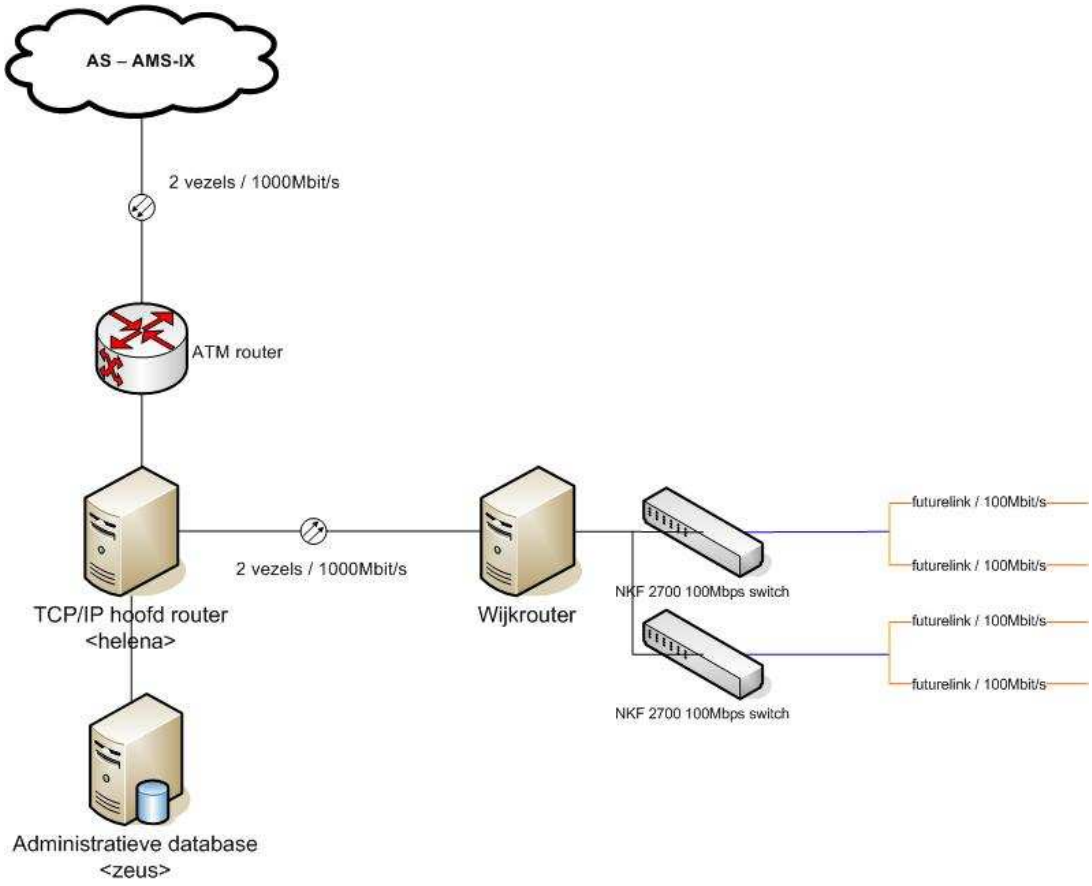
Deze functies worden allemaal door middel van software uitgevoerd. Nu had ik de functies van de wijkrouter. Echter om de bestaande software op waarde te kunnen taxeren moest ik weten waarom de keuzes gemaakt waren om bijvoorbeeld alles in software in te stellen, en niet als instelling van bijvoorbeeld een switch of router.

Om dit te kunnen bepalen moest ik eerst een idee hebben waar in het netwerk van Lijbrandt Telecom de wijkrouter zich bevond.

[3] Quality of services – maatregelen op die de kwaliteit van een dienst beïnvloeden.

5.1 De context van de wijkrouter software

Om goed te kunnen inzien hoe het netwerk in elkaar zat, heb ik met Joost Vermeer gesproken om zoveel mogelijk informatie te verkrijgen. De onderstaande afbeelding laat zien hoe het netwerk in elkaar zit:



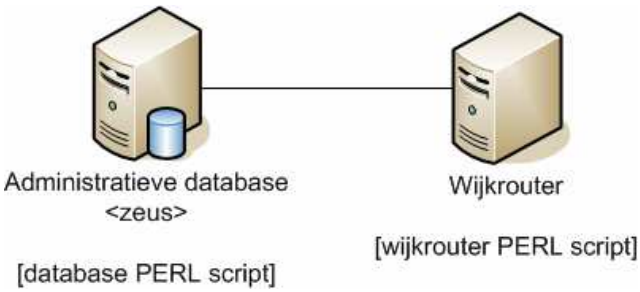
Figuur 5.1: Netwerk indeling Lijbrandt Telecom

Verder heb ik onderzocht of er documenten waren die beschreven hoe de oude software in elkaar zat. Dit was ook niet het geval, er was slechts een beschrijving van de administratieve database die de wijkrouter software hanteert.

In deze tekening is ook de naam ‘Futurelink’ terug te vinden. Dit is een op maat gemaakte kabel die alleen bij Lijbrandt Telecom gebruikt wordt. Deze kabel is door LG ontwikkeld en is een verzameling van: Glasvezelkabel, UTP Cat5E, Coax kabel en een 2 aderige telefoon kabel gewikkeld in een harde mantel. Deze kabel wordt afgemonteerd in de huiskamers van de abonnees in één kastje.

5.2 De werking van de oude wijkrouter software

De oude wijkrouter software werkt door middel van twee PERL scripts die samenwerken op twee aparte servers. Deze servers draaien een Linux Debian distributie. Het eerste script is in staat om door middel van SQL queries de benodigde informatie uit de database te halen. Dit script is actief op dezelfde server als waar de database op draait. Het tweede script is actief op de wijkrouter server zelf. Dit script roept het 'databasescript' aan waardoor het in staat is de switches die aan de wijkrouter zijn gekoppeld in te stellen. Om dit duidelijk weer te geven heb ik dit in een tekening gezet.



Figuur 5.2: Werking: ophalen administratieve gegevens

5.3 Beveiligde verbinding van de wijkrouter

Allereerst wordt de beveiliging gewaarborgd doordat alle communicatie op administratief niveau gedaan op een apart netwerk door middel van VLAN's[4]. Daarnaast worden gegevens door middel van een RSA beveiligde SSH verbinding opgehaald bij de database. De authenticatie gebeurd door middel van een gebruiker en een wachtwoord.

Wanneer gegevens direct vanaf de ene server worden opgehaald uit de database(wat heel goed mogelijk is) dan zou alle informatie die nodig is onbeveiligd over het internet verstuurd worden.

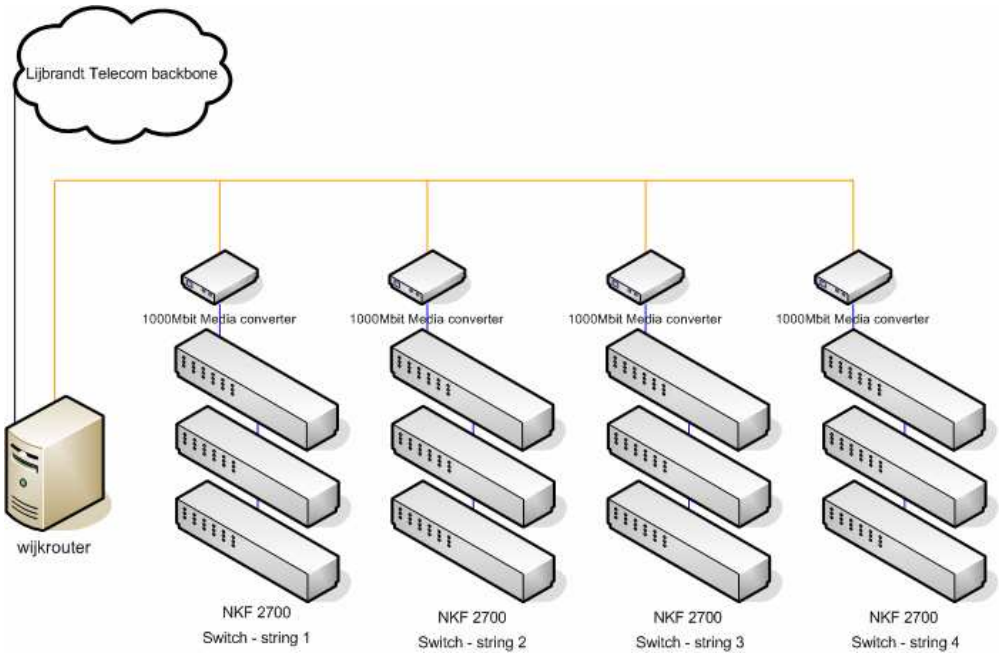
Normaal gesproken zou hiervoor ook een zeer ingewikkelde beveiligingsprocedure voor geschreven kunnen zijn. Echter omdat de oude wijkrouter software geschreven is in PERL en hierdoor makkelijk programma's uit kan voeren op het systeem, is het onnodig om hier zelf iets voor te schrijven volgens Dhr. Vermeer.

[4] Virtual LAN – Een verzameling technieken om een netwerk onder te verdelen in verschillende kleine netwerken.

5.4 Implementatie van de wijkrouter

Om de gegevens die de wijkrouter opgevraagd heeft ook daadwerkelijk te implementeren, was het nodig om controle eenheden te creëren die er voor zorgen dat er per abonnee één set attributen kan worden toegepast. Om dit voor elkaar te krijgen gebruikt men bij Lijbrandt Telecom een apart VLAN per abonnee. Deze VLAN's worden dan ook ingesteld op de aangesloten switches en het verkeer wordt dan door middel van zogenaamde 'VLAN tagging' onderscheiden en gerouteerd. Dit betekend dus dat er niet alleen een bestuurlijke functie is vanuit de router gezien, maar dat er ook attributen van een port op een switch moeten worden aangepast. Deze aanpassingen gebeuren door middel van systeem commando's en op basis van reeds bestaande software die voor het Linux operating system geschreven is, en door het wijkrouter script worden aangeroepen.

Deze aangepaste gegevens worden dan op de switches toegepast. In de oude opzet zijn dit NKF 2700 switches en NKF 2600 switches. Op de onderstaande afbeelding is te zien hoe de infrastructuur is ingedeeld.



Figuur 5.3: gehanteerde netwerk topologie

Hier is te zien dat men bij Lijbrandt Telecom gebruik maakt van een bus topologie. Waarvan inmiddels ook bij Lijbrandt Telecom bekend is dat het niet de ideale oplossing was. Met het oog op de toekomst wil Lijbrandt Telecom ook hier verandering in brengen en overgaan op een ster topologie. Op basis van een VLAN wordt de snelheid van de verbinding door middel van software gereguleerd. En er wordt niets op de poort ingesteld. De enige wijzigingen die worden toegepast op de poorten van de switches zijn het toekennen van een VLAN. Wanneer dit gebeurd is zal de wijkrouter het verkeer routeren op basis van VLAN-id[5].

[5] Virtual LAN ID – Een IP header waarop onderscheiden wordt tot welke interface van de wijkrouter het verkeer behoort.

5.5 Het configureren van de switches

Om de switches zodanig te configureren zodat deze perfect aansluiten op de manier waarop de wijkrouter het verkeer routeert, heeft Lijbrandt Telecom een geautomatiseerde configuratie procedure. Dit is eveneens een PERL script dat er voor zorgt dat de corresponderende VLAN id's worden doorgevoerd op de switches en dat er een aantal onderhoudstaken worden uitgevoerd op de switches.

Op het moment dat ik begon met dit project maakte Lijbrandt Telecom gebruik van NKF 2700 switches. Deze switches kunnen worden geconfigureerd door middel van een eigen user interface die werkt via telnet.

Het was niet nodig om mij verder te verdiepen in deze configuratie procedure aangezien er gebruik zou worden gemaakt van 'nieuwe' switches. Dit is de Cisco Catalyst 2950.

Wel is het zo dat de configuratie op dit moment gebeurd door middel van een script die een telnet sessie opent met de desbetreffende switch en daarop een configuratie slag uitvoert. Dit gebeurd dan meestal op een aantel switches tegelijk. Uiteindelijk is het enige dat voor mij relevant was, was de instelling van VLAN's op de poorten van de switch.

De Cisco 2950 switch maakt gebruik van een geheel andere manier van management software. Waar de NKF 2700 switch nog gebruik maakte van een telnet user interface maakt de Cisco switch gebruik van Cisco IOS welke ook over telnet gaat echter is dit een command-line interface. Dit is een geavanceerd operating systeem van Cisco dat op elke router en switch van Cisco Systems te vinden is.

Om dit op een juiste manier te implementeren is kennis nodig over deze manier van configureren. Die is niet altijd bij iedereen voorhanden. Er zijn opleidingen van Cisco Systems gewijd aan deze manier van configureren(o.a. CCNA en CCNP). Zelf heb ik ooit Cisco Network Academy Program gevolgd waardoor ik redelijk in staat ben om met behulp van de documentatie die geleverd wordt bij de switch, een Catalyst 2950 te configureren.

Op het moment van schrijven staat Lijbrandt Telecom op het punt om over te gaan van de NKF 2700 switch naar de Cisco Catalyst 2950.

6. Definitie fase

6.1 Het verkrijgen van User stories

Bij het verkrijgen van de user stories heb ik de meeste functionaliteiten van het systeem kunnen achterhalen bij de bespreking met de opdrachtgever over wat er precies moest komen. De volgorde binnen Extreme Programming is om eerst de userstories vast te stellen in het klant jargon om daarna met behulp van use-cases en use case scenario's de user requirements vast te stellen. Dit heb ik dan ook gedaan.

De volgende user stories zijn naar voren gekomen:

- De wijkrouter moet verkeer kunnen doorvoeren naar abonnees
- De wijkrouter moet kunnen zorgen voor een maximale snelheid van 30 Megabit per seconde per abonnee
- De wijkrouter moet in staat zijn een abonnee aan en af te sluiten
- De wijkrouter moet betrouwbaar en niet storinggevoelig zijn.
- De wijkrouter moet het mogelijk maken om een Globally Routable IP adres toe te kennen aan abonnees.

Deze user stories zijn tot stand gekomen door met de opdrachtgever te discussiëren en zijn niet door mij opgesteld. Dit heeft als voordeel dat dit de rauwe informatie is waarmee ik dan use-cases kan opstellen en user requirements van kan afleiden. Ze lijken voor een deel hetzelfde en ze zullen ook overeenkomen.

Verder heet de linux router 'wijkrouter' binnen de organisatie van Lijbrandt Telecom.

6.2 Opstellen van de user requirements

De user requirements zijn een afgeleide van de user stories. Het grootste verschil met user stories is dat dit daadwerkelijke eisen zijn van het systeem. Deze eisen zijn tot stand gekomen door middel van de opdrachtgever en technische afwegingen die ik zelf gemaakt heb. Ook komen er voor een deel zaken aan bod die de opdrachtgever niet meteen zag als een eis voor de wijkrouter, maar toch zeer noodzakelijk bleken uit technische overwegingen.

De eisen:

- Gelijktijdigheid van doorvoer van het verkeer (erlang[6] verhoudingen)
  - Hoeveel verkeer kan de wijkrouter doorvoeren tegelijkertijd met een doel bandbreedte van 30Mbps.
- Bandbreedte beperking.
  - Een verbinding dient te worden ingesteld op een vaste bandbreedte.
- VLAN toewijzen
  - Om Laag 2 verkeer van elkaar af te schermen en aan elk VLAN een GRIP (Globally Routable Internet Adress) te koppelen. Een bijkomstige functie is dat ik per VLAN(subnet) een aantal attributen kan instellen.
- Veiligheid en integriteit van gegevens die van en naar de switches verstuurd wordt.
- Traffic Management
  - Het mogelijk maken van een download limiet op een bepaalde dienst.
- Port Management
  - Het uitschakelen van een bepaalde poort voor een gebruiker bij constatering van een virus.
- Te wijzigen attributen van een internet verbinding toepassen op de poorten van een switch in het leveringsgebied.
- Billing data om gegeneerde kosten te kunnen factureren per verbruikte data een factuur te genereren
- VoIP verbruik
  - Op basis van tarieven
  - Afluister mogelijkheid voor justitie
  - 2 maanden lang log mogelijkheid.
- Gegevens weergeven in een status scherm.
  - Hoeveel verkeer de wijkrouter doorvoert
  - Mogelijke problemen d.m.v. opgemerkte performance pieken.
- Gegevens via SNMP wegsturen naar een centrale database om zo de load per POP te kunnen zien.
- Unicast of Multicast verkeer Dit ten behoeve van Televisie oplossingen voor Lijbrandt Telecom.
- De Wijkrouter moet in te zetten zijn in een omgeving waar de bandbreedte leverancier niet Lijbrandt Telecom is.

[6] Erlang – een mechanisme om te berekenen hoeveel verbindingen op een distributiekanaal mogelijk zijn.

6.3 Releaseplan

ik heb voor het releaseplan gebruik gemaakt van het MoSCoW principe. Dit houdt in dat de prioriteiten op de volgende manier worden ingedeeld:

- **M**ust have
- **S**hould have
- **C**ould have
- **W**ould have

Om dit te kunnen doen heb ik met de opdrachtgever gesproken om er achter te komen waar hij de prioriteit aan geeft. Hierbij hebben we gekeken wat precies haalbaar is binnen gestelde tijd. Dit is ook de reden waarom er een releaseplan is gemaakt.

Normaal gesproken zouden de prioriteiten van het op te leveren werk gesteld worden in het plan van aanpak. Echter omdat ik nog niet precies wist waar ik mee te maken had heb ik gekozen om dit na het onderzoek naar de wijkrouter te doen. Hierdoor kon ik goed inschatten hoeveel werk bepaalde onderdelen zouden kosten.

Wat ook een argument is om dit na het plan van aanpak te evalueren is dat de opdrachtgever ook niet precies kon inschatten hoeveel werk een bepaald onderdeel van het project zou kunnen kosten. Mede hierdoor heb ik op deze manier de prioriteiten op een wel overwogen manier kunnen vaststellen.

- **Must have**

  Het verkeer moet gerouteerd worden door de wijkrouter en de volgende attributen dienen te worden gecontroleerd. En minimaal kunnen hanteren.

  - Gelijktijdigheid van doorvoer van het verkeer (erlang verhoudingen)
    - Hoeveel verkeer kan de wijkrouter doorvoeren tegelijk met een doel bandbreedte van 30Mbps.
  - Shaping
    - Een verbinding dient te worden ingesteld op een vaste bandbreedte.
  - VLAN toewijzen
    - Om Laag 2 verkeer van elkaar af te schermen en aan elk VLAN een grIP (Globally Routable Internet Adress) te koppelen. Een bijkomstige functie is dat ik per VLAN(subnet) een aantal attributen kan instellen.
  - Veiligheid en integriteit van gegevens die van en naar de switches verstuurd wordt.
  - De Wijkrouter moet in te zetten zijn in een omgeving waar de bandbreedte leverancier niet Lijbrandt Telecom is.

- **Should have**

  Beheersbaarheid van alle elementen van een internetverbinding door middel van de huidige Lijbrandt Telecom Internet Administratie.

  - Traffic Management

    - Het mogelijk maken van een download limiet op een bepaalde dienst.
  - Poort Management
    - Het uitschakelen van een bepaalde poort voor een gebruiker bij constatering van een virus.
  - Te wijzigen attributen van een internet verbinding toepassen op de poorten van een switch in het leveringsgebied.

- **Could have**

  Billing data om gegeneerde kosten te kunnen factureren per verbruikte data éénheid en een factuur te genereren
  - VoIP verbruik
    - Op basis van tarieven
    - Afluister mogelijkheid voor justitie
    - 2 maanden lang log mogelijkheid.

- **Would have**

  - Gegevens weergeven in een status scherm.
    - Hoeveel verkeer de wijkrouter doorvoert
    - Mogelijke problemen d.m.v. opgemerkte performance pieken.
  - Gegevens via SNMP wegsturen naar een centrale database om zo de load per POP te kunnen zien.
  - Unicast of Multicast verkeer Dit ten behoeve van Televisie oplossingen voor Lijbrandt Telecom.

6.3.1 Project Velocity

De voorgaande eisen zijn een specificatie van het uiteindelijke product. Om de werkzaamheden in te plannen die nodig zijn om tot een product te komen met de bovenstaande mogelijkheden, heb ik dit verwerkt in een Gantt chart. Deze Gantt chart is terug te vinden in het plan van aanpak in hoofdstuk 4 van dit document.

Ik heb voor dit project gebruik gemaakt van een timebased release plan waarin ik aangeef hoe lang ik over een bepaald deel van het project zal gaan doen. De reden hiervan is omdat ik van te voren wist hoe lang ik over dit hele project ging doen.

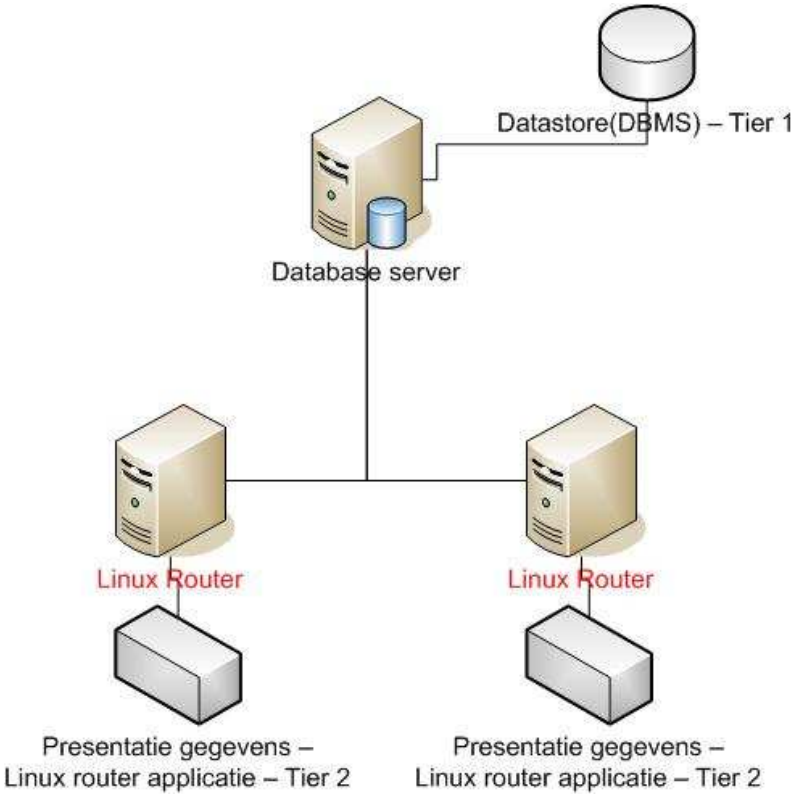
7. Ontwerp fase

7.1 Waarom ik het 3-Tier model gebruik voor het ontwerp

Voor het ontwerp van de wijkrouter heb ik gebruik gemaakt van het 3-Tier model. Dit model houdt in dat er een scheiding wordt gemaakt tussen: Data, Model en View. Bij het ontwerpen van Client/Server software is het 3-Tier model een veel gebruikte oplossing om zo efficiënt mogelijk met resources om te gaan.. En aangezien ik een Linux router ging maken die gegevens verwerkt vanuit een Administratief systeem, was het nodig om na te denken over een oplossing die het gemakkelijk maakt om een scheiding te maken tussen de manier waarop de gegevens worden aangeboden aan de Linux router en de manier waarop deze worden verwerkt. Dit betekent dat ik dan als grootste voordeel heb, dat wanneer iemand het relationele database ontwerp zou wijzigen, er niet een algehele update voor de Linux router applicatie nodig zou zijn.

Bij de ontwikkeling van de Linux router kwam er geen User interface aan te pas. Dus in dat geval wijkt het af van de normale gang van zaken bij het 3-Tier model. Maar het neemt niet weg de Linux router baat heeft bij de voordelen van het 3-Tier model. De Linux applicatie kan worden gezien als de ‘presentatie’ laag wanneer het om de gegevens gaat.

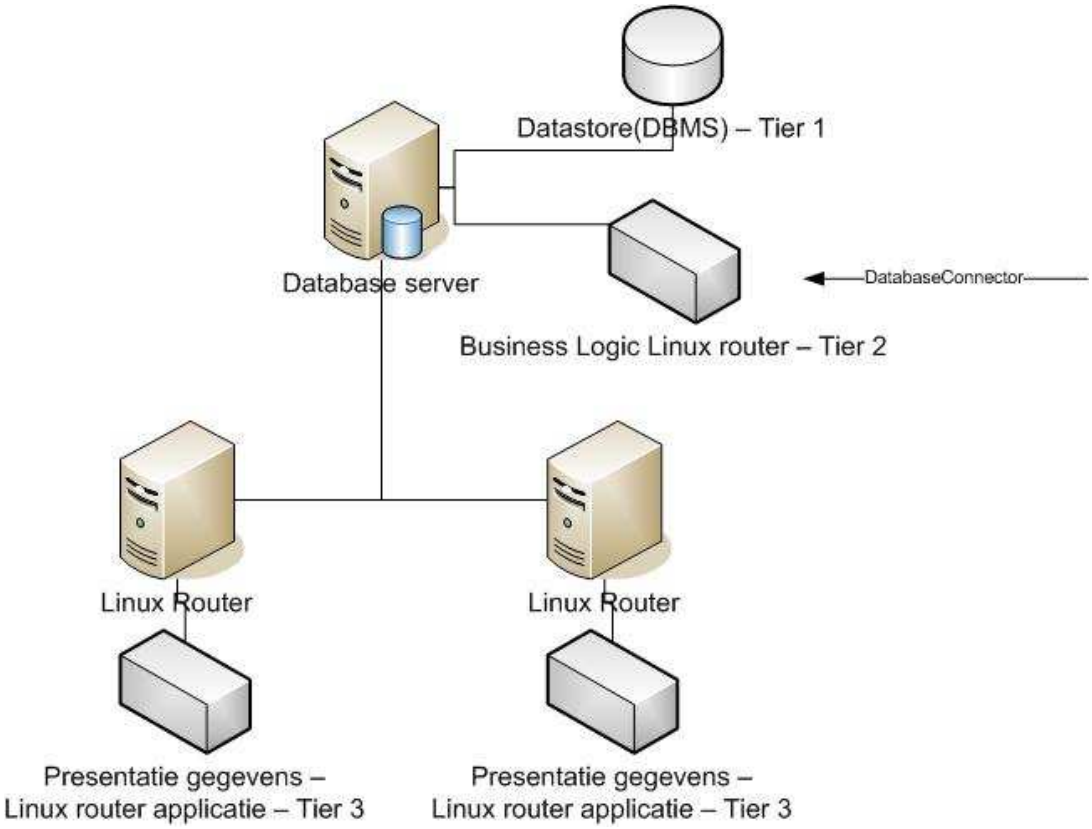
Om aan te geven hoe ik hier in de praktijk baat bij heb, heb ik een scenario uitgewerkt. Als ik de volgende situatie zou tegenkomen:



Figuur 7.1: een implementatie waarbij het 3-tier model *niet* gebruikt wordt

Aangezien de database voor meerdere doeleinden gebruikt wordt dan alleen voor de Linux router en er meer mensen mee werken, zou het zo kunnen zijn dat iemand iets wijzigt aan het ontwerp van het datamodel. Dit zou als gevolg hebben dat de software op elke individuele implementatie van de Linux router applicatie niet meer zou werken, en een update noodzakelijk is.

In het geval van 3-tier ziet het er als volgt uit:



Figuur 7.2: een implementatie waarbij het 3-tier model *wel* gebruikt wordt

In dit geval kan in de ‘Business Logic’ tier worden ingebouwd, dat wanneer er een wijziging in het datamodel gedaan wordt. Er maar op één punt een wijziging gedaan hoeft te worden. Dit is in de Business Logic tier. Dan is de conclusie dat een wijziging in het datamodel minder potentiële problemen met zich meebrengt.

Andere misschien meer generieke voordelen van het 3-Tier model zijn:

- De code is beter te testen en is beter onderhoudbaar en overdraagbaar. Met name het scheiden van de presentatie van de rest van de toepassing is hierbij belangrijk. De kosten vallen hierdoor ook lager uit.
- Het is geschikt voor latere uitbreiding. Het is bijvoorbeeld veel eenvoudiger een upgrade te doen van de database als alle daaraan gerelateerde code geconcentreerd is op één plek en niet overal in de applicatie voorkomt. Het toevoegen van een stuk business-logic kan opgepakt worden als een aparte module, een apart project.
- Er kan parallel ontwikkeld worden door verschillende expertteams.
- De code is goed herbruikbaar doordat elke module een op zichzelf staand stukje functionaliteit is en dus opnieuw gebruikt kan worden in verschillende applicaties.

bron: **http://www.thedutchrepublic.com/profile/technology.html**

Om aan te geven hoe deze structuur zich verhoudt tot het ontwerp van de Linux router heb ik een schema gemaakt om weer te geven waar welk onderdeel van de Linux wijkrouter toe behoort in dit model. Overigens is dit dezelfde structuur als in de voorgaande afbeelding.

Linux router	**3-Tier model**
Database(DBMS)	Datastore
DatabaseConnector(DatabaseConnector - script)	Business Logic
Linuxrouter applicatie(Applicatie – script)	User interface

7.2 Het vaststellen van de use-cases

Bij het vaststellen van de use-cases was het van belang om te laten zien op welke manieren een gebruiker, gebruik gaat maken van de wijkrouter software. In mijn geval was het anders dan normaal. De gebruiker is geen mens maar een gebruiker die vanaf een ander geautomatiseerd systeem inlogt en gegevens wijzigt en daarna deze toepast door middel van de software die ik ging maken.

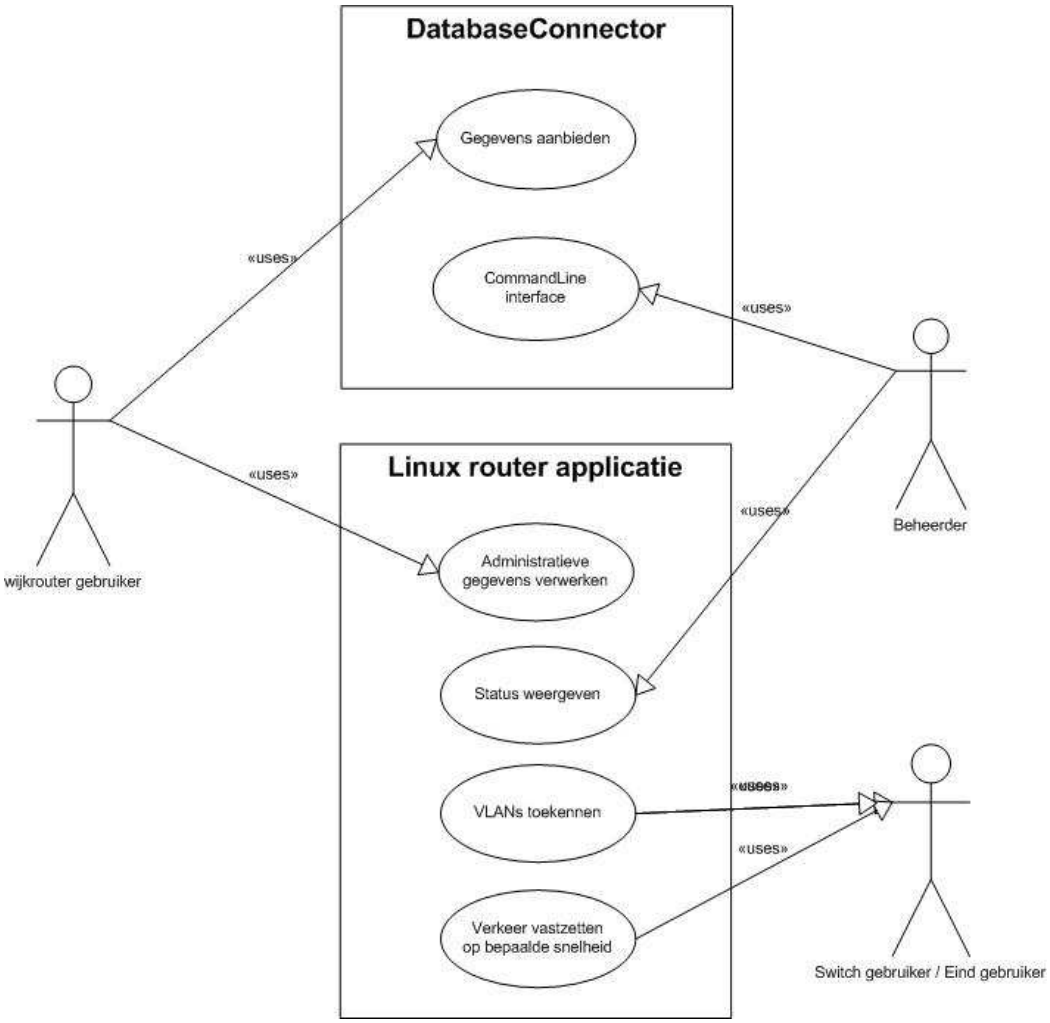
Hiernaast gaat het om een aantal scripts die gemaakt moeten worden. Dit betekent dat er een aantal use-case diagrammen uit kwamen. Als laatste wordt er onderscheid gemaakt tussen een technische gebruiker(linux gebruiker) en een gebruiker op concept niveau. Het is onhandig om binnen dit conceptuele ontwerp te praten over de gebruiker ‘root’ die bepaalde zaken uitvoerd. Mede omdat de verschillende scripts ook van elkaar gebruik maken heb ik relevante benamingen gebruikt voor de gebruikers, ook wel ‘actors’ genoemd binnen de UML terminologie.

Om deze use-cases duidelijk te maken heb ik dan ook gebruik gemaakt van een UML use-case diagram.

Uit dit diagram valt af te leiden dat het systeem bestaat uit twee aparte delen. Een systeem dat de gegevens ophaalt uit de database en een systeem dat de eigenlijke functies van de wijkrouter implementeert. Deze keuze is ooit al gemaakt om de beveiliging van de gegevens die over het internet verstuurd worden zo simpel mogelijk te laten verlopen. En uiteraard om het eerder genoemde 3-Tier ontwerp voordelen.

Het idee over de beveiliging heb ik overgenomen van de vorige ontwerpers van de eerdere software. Hierdoor is het nodig om in het ontwerp al aan te geven dat er een actor is die ervoor zorgt dat de communicatie goed verloopt. In plaats van een virtuele gebruiker van de database is er nu een virtuele gebruiker op linux niveau. Ik zal hier verder in deze beschrijving over het ontwerp op terug komen.

Voor de duidelijkheid moet ik verhelderen dat de use-cases in geen geval alle toekomstige functies van de wijkrouter bevatten. Ik heb deze use-cases opgesteld aan de hand van het releaseplan en de aangegeven prioriteiten daarin.



Figuur: 7.3: Use-case diagram van de wijkrouter software

7.3 Het beschrijven van de use-case scenario's

Bij het schrijven van de use-case scenario's was het de bedoeling om elke use-case uit te schrijven naar een scenario. De use-cases in het use-case diagram zijn daarom als volgt beschreven:

Use-case scenario "Administratieve gegevens verwerken"

Pre condities:

- Er moet een verbinding mogelijk zijn tussen de DatabaseConnector en de Wijkrouter software om de gegevens uit de database te halen.
- Er moeten gegevens aanwezig zijn in de database die verwerkt kunnen worden.

Scenario beschrijving:

Er wordt een verbinding geopend met de server waarop op aanvraag van het wijkrouter script de DatabaseConnector gegevens uit de database haalt. Hierna zal de DatabaseConnector gegevens terug sturen om deze te gebruiken om de wijkrouter te configureren.

Post condities:

- Gegevens moeten bruikbaar zijn om als basis te dienen voor de configuratie van de wijkrouter. Dit betekend dat de IP adres, upload en download snelheid en actief of non-actief configuratie items per verbinding moeten kunnen worden bewerkt.

Use-case scenario "Status weergeven"

Pre condities:

- Er moet een werkende wijkrouter zijn die log bestanden genereert waaruit statistieken gegenereerd kunnen worden.

Scenario beschrijving:

Het is voor de beheerder erg belangrijk om de mogelijkheid te hebben te zien wat er gebeurd op een wijkrouter. De doorvoer (current troughput) en het aantal verbindingen zijn twee van de meest basis gegevens die het status overzicht moet weergeven. Meerdere statistieken moeten mogelijk zijn.

Post condities:

- Gegevens moeten bruikbaar zijn om als basis te dienen voor beslissingen die mogelijk leiden tot het inzetten van extra capaciteit en verbetering van de diensten.

Use-case scenario “VLAN’s toekennen”

Pre condities:

- De gegevens die nodig zijn om de routes van het verkeer te configureren dienen aanwezig te zijn.
- Er moet worden gezorgd dat er voldoende bandbreedte beschikbaar is om aan de vraag van de eindgebruiker en de conceptuele instelling van de bandbreedte beperking te voldoen.

Scenario beschrijving:

De bedoeling van het routeren van het verkeer is om te zorgen dat mensen op basis van een apart VLAN elk een apart subnet krijgen. Aan dit subnet is één IP adres toegekend.

Post condities:

- De integriteit van de gegevens moet voldoende zijn. Hieronder moet onder worden verstaan: Het IP adres en het wel of niet actief zijn van de verbinding.

Use-case scenario “Toekennen van een snelheid beperking”

Pre condities:

- Er moet een VLAN ID bekend zijn van de verbinding waar een snelheidbeperking wordt ingesteld.
- De gegevens om in te stellen hoe snel een verbinding moet zijn dienen aanwezig te zijn.

Scenario beschrijving:

De snelheid van het verkeer dient te worden ingesteld op de snelheid die is toegekend vanuit de administratieve systemen binnen Lijbrandt Telecom. Hierbij is het de bedoeling dat de gegevens die opgehaald zijn worden toegepast op een verbinding waarbij het VLAN ID als identificerend object dient.

Post condities:

- De snelheid van de verbinding dient te zijn ingesteld op de bedoelde snelheid.

7.4 Het klassendiagram

Het proces van het tot stand komen van een klassen diagram was niet vanzelfsprekend. Het maken van een klassendiagram was in eerste instantie niet wat ik zou gaan doen. Ik vond dat met het oog op de manier waarop de software gebruikt gaat worden er niet meteen een reden is om de software object georiënteerd te maken. Ik wist ook nog niet of er mogelijkheden waren binnen PERL 5 om object georiënteerd te programmeren.

De grootste reden om toch naar een object georiënteerd systeem over te gaan is de herbruikbaarheid van de communicatie software tussen de Lijbrandt Telecom Internet Administratie en de wijkrouter. Deze zou herbruikbaar zijn voor toekomstige oplossingen van Lijbrandt Telecom. Ik heb inmiddels vernomen dat Lijbrandt Telecom in de toekomst ook Televisie aan en af wil sluiten via de wijkrouter. Om dit te kunnen doen is een aanpassing nodig aan de wijkrouter software, maar zijn de klassen om de communicatie te regelen tussen de wijkrouter en de administratieve systemen opnieuw te gebruiken.

Ik heb om er achter te komen of object georiënteerd programmeren binnen PERL een optie heb ik gesproken met Dhr Joost Vermeer. Hij legde me uit dat het mogelijk is om de object georiënteerde theorie toe te passen binnen PERL 5.

Hierna heb ik op het internet gezocht naar informatie over OOP[7] en PERL 5. Hier kwam naar voren dat ik met behulp van modules binnen PERL 5 de object georiënteerde denkwijze kon hanteren. Daarnaast ben ik ook op de hoogte gesteld van een aantal gedragsregels binnen PERL zodat ik het hiermee rekening kon houden tijdens het ontwerpen.

Een van deze gedragsregels binnen PERL is de manier waarop PERL 5 aan information hiding doet. Door een functie(of routine zoals dit binnen PERL5 heet) te beginnen met een '_' (i.e. _getWaardes()) is het makkelijk te herkennen als private functie. Vanzelfsprekend zijn de functies zonder '_' te herkennen als public functies binnen PERL 5.

Ook worden objecten binnen PERL 5 gezien als modules. Deze modules hebben de normale eigenschappen van een klasse die in elke andere programmeertaal ook beschikbaar zijn. Echter is het toch verwarrend aangezien een modulaire opbouw in elke andere programmeertaal niet duidt op een object georiënteerde omgeving. Echter binnen PERL 5 wordt hier dezelfde object georiënteerde theorie toegepast als in een taal zoals bijvoorbeeld JAVA.

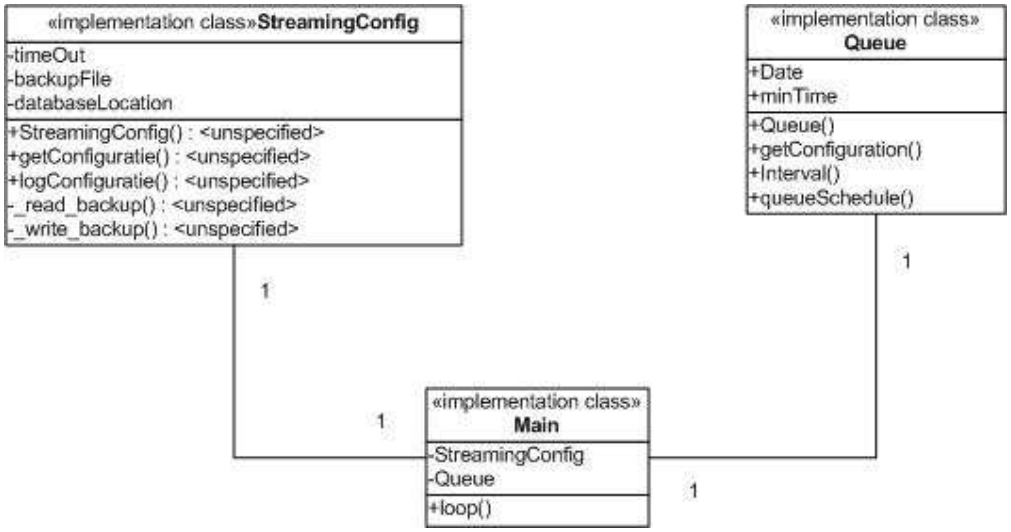
Nu ik overtuigd was van de mogelijkheden binnen PERL van Object oriëntatie heb ik mij gericht op het ontwerp. Ik moest er achter zien te komen welke delen bij de communicatie horen en welk delen van de functionaliteit specifiek zijn voor de wijkrouter. Ook moest ik mij afvragen hoe de software om zou gaan met meerdere aanvragen tegelijk.

Om dit te kunnen doen heb ik mij verdiept in de werking van de wijkrouter die er al was. Om die basis functionaliteit netjes te kunnen implementeren heb ik gekozen om de software opnieuw en voor een groot deel anders op te bouwen. Ik heb gekozen

[7] Object Oriented Programming – De object georiënteerde manier van programmeren.

om twee klassen te definieren die herbruikbaar zijn. Dit zijn de klasse ‘StreamingConfig’ en de klasse ‘Queue’. De functionaliteit die voor de wijkrouter specifiek is heb ik ‘Main’ genoemd.

In het onderstaande klassendiagram heb ik de klassen, attributen en operaties gedefinieerd.



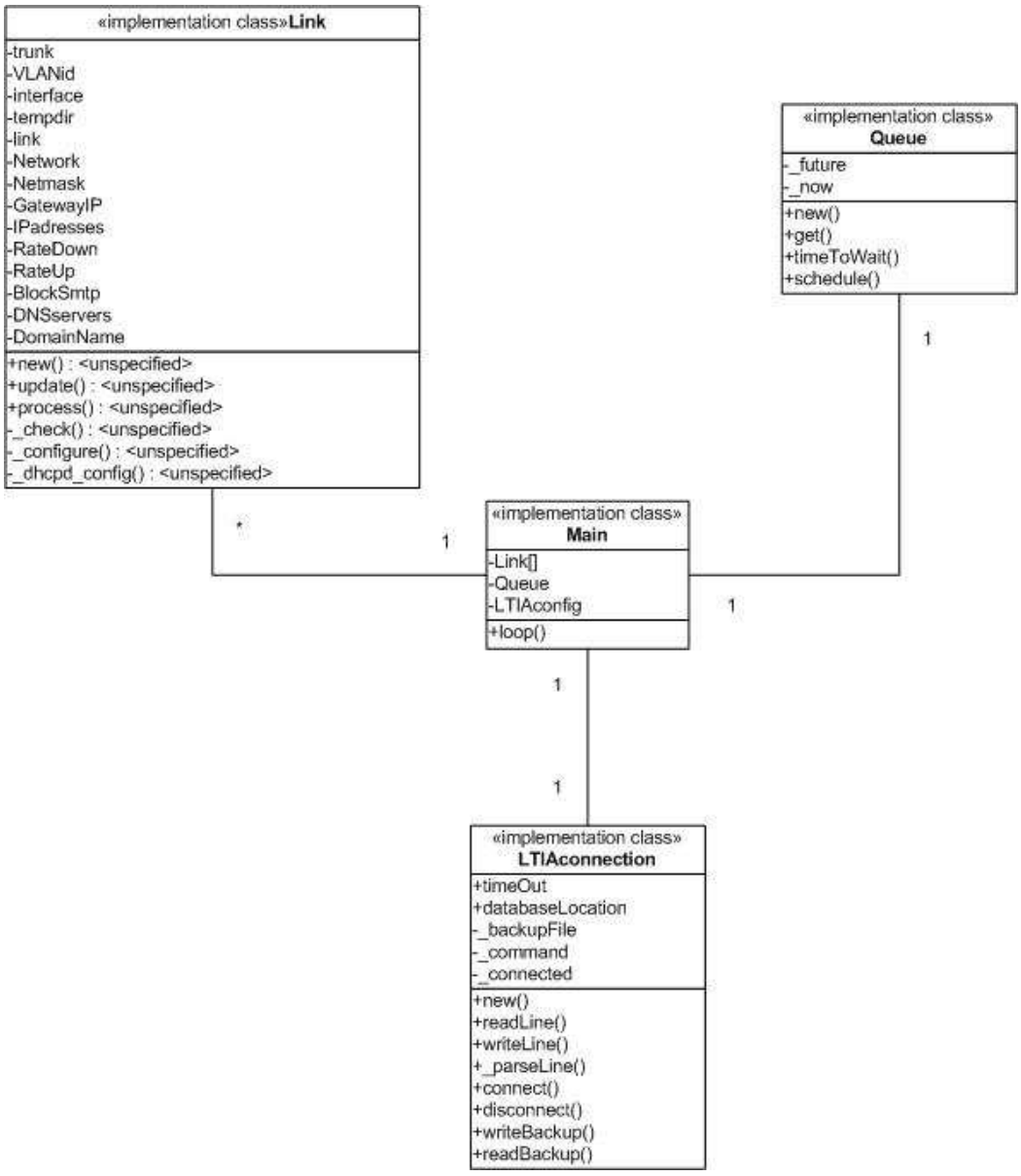
Figuur 7.4: Klassendiagram van de wijkrouter software

De StreamingConfig klasse heeft als functie het verzorgen van de aanvraag van de verbinding en het uitlezen van de teruggekomen informatie en het object dient dit aan te bieden aan het object Main. De Queue klasse heeft als functie om verbindingen op een goede manier te openen naar de DatabaseConnector en deze te plannen. Dit houdt in dat er om een bepaalde tijd een verbinding wordt geopend naar de DatabaseConnector om de configuratie van de wijkrouter te synchroniseren.

Hiernaast was er nog een script aan de database kant dat de queries naar de database stuurt om dit vervolgens weer terug te sturen aan de database. Ik heb er voor gekozen dit script niet object georiënteerd uit te voeren, aangezien het hier gaat om een verzameling van queries, die niets anders doen dan gegevens ophalen op aanvraag en deze weer terugsturen.

7.4.1 Klassendiagram Iteratie 2

Na het eerste deel van de software voor de Linux router geschreven te hebben ben ik tot de conclusie gekomen dat er een betere manier is om de presentatie tier(wijkrouter applicatie) te ontwerpen. Het klassendiagram voor de 2e versie van het ontwerp ziet er als volgt uit.



Figuur 7.5: Klassendiagram iteratie 2

Dit houdt in dat er 1 klasse is bijgekomen. Dit is de LTIAConnection klasse en er is een Klasse gewijzigd. Alleen de Queue klasse is voor het grootste deel hetzelfde gebleven. Het proces dat vooraf ging aan deze wijziging is een constante stroom aan ideeën die ter tafel gekomen zijn bij gesprekken met de opdrachtgever. Het ging dan om zaken die in eerste instantie nog niet duidelijk waren en misschien ook niet meteen als eis gesteld waren aan het systeem.

Dit bracht mij in een situatie waarbij ik functionaliteit ging verwerken in de software die in eerste instantie niet in de software verwerkt zou gaan worden wanneer ik uit zou gaan van het releaseplan.

Een voorbeeld hiervan is het gebruik van een back-up bestand voor de totale configuratie. Deze heeft als functie om de laatst opgehaalde configuratie op te slaan en deze aan te bieden aan de software wanneer de DatabaseConnector niet beschikbaar is. Deze eis is niet verwerkt in het releaseplan dus ik zou deze eis ook niet inwilligen in de software. Ik heb mij hierbij beroepen op een ethische keuze waarbij het toch verstandiger is om deze eis wel te verwerken, om zo de stabiliteit van de gehele software te verbeteren. Mijn reden hiervoor was dat de eis geen extra functionaliteit biedt maar wel de functionaliteit die er moet zijn, ondersteund en stabieler maakt.

Belangrijk voor mij was overigens ook dat er voor de DatabaseConnector geen apart klassendiagram is gemaakt. Ik vond het niet kloppen met de Object georiënteerde theorie om hiervoor een klassendiagram te maken. Dit omdat het echt een procedureel programma is. Wel zal ik hiervoor een PSD[8] maken om zo de programma structuur weer te geven. Omdat ik deze structuur van het programma nog niet precies van te voren kan vast leggen, en de natuur van Extreme Programming veel vrijheid biedt, heb ik deze PSD gemaakt tijdens het programmeren. En om deze reden is het PSD terug te vinden in hoofdstuk 8 van dit document.

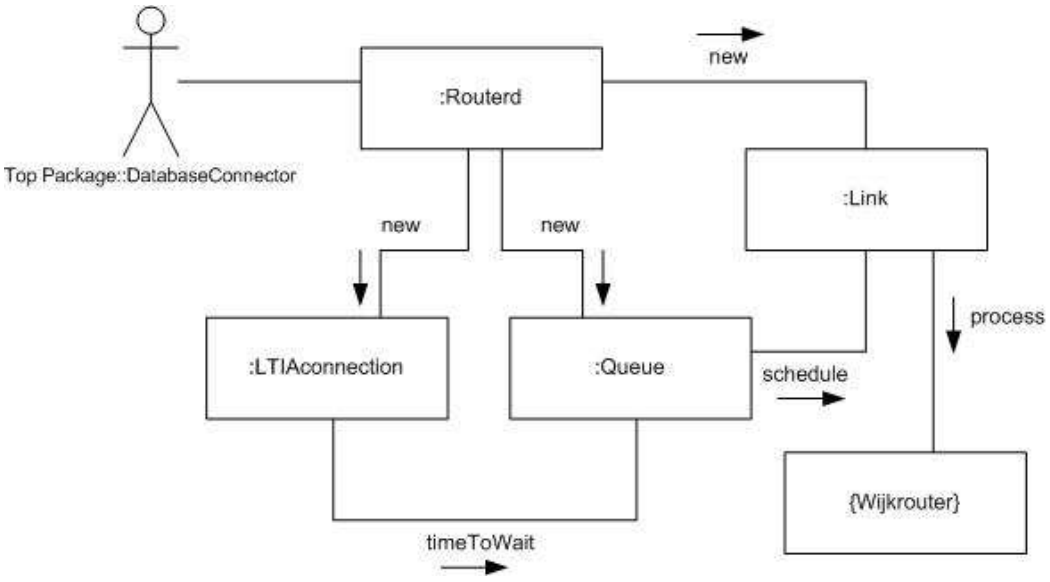
[8] Program Structure Diagram – techniek om de structuur van een programma of algoritme weer te geven.

7.5 Het collaboratiediagram

Ik moest een manier hebben om de interactie tussen de objecten te modelleren omdat er gebruik wordt gemaakt van een wacht rij in de Linux router applicatie. Niet alleen om dit op deze manier te ontwerpen maar ook om als naslag te dienen voor eventuele verdere ontwikkeling van deze software.

Binnen de techniek UML zijn er een aantal mogelijkheden om dit te doen. Het sequentiediagram voorziet onder andere in deze mogelijkheid. Echter is het sequentiediagram voornamelijk bedoeld om tijdafhankelijke objecten weer te geven. Dit speelde bij de Linux router applicatie geen rol. Ik wilde vooral de context beschrijven en de samenwerking tussen de objecten laten zien. Daarom heb ik gekozen voor het collaboratiediagram.

Het collaboratiediagram dient inzicht te geven in de objecten en de omgeving op het moment dat een object gecreëerd wordt. Het collaboratiediagram voor de Linux router applicatie ziet er als volgt uit:



Figuur 7.6: Collaboratie diagram Linux router applicatie

De bedoeling van dit diagram is om de samenwerking tussen de verschillende objecten te ontwerpen. In figuur 7.6 is te zien hoe de objecten: Link, LTIAConnection en Queue worden aangemaakt door het object Routerd. Verder heeft het object LTIAConnection een methode ‘timeToWait()’ die de configuratie regels één voor één per een bepaalde tijd uitleest en niet alles tegelijk. Wanneer dit gebeurd is zal er een Link object aangemaakt worden. Deze dient als container van informatie over een Link(abonnee). Het object Link heeft een methode ‘process()’ die er voor zorgt dat de ontvangen gegevens van de DatabaseConnector worden geïmplementeerd op de router. Ook dit gebeurd niet allemaal tegelijk maar wordt afgehandeld door het Queue object. Dit is de reden waarom het bericht ‘schedule’ wordt verstuurd naar het object Link.

8. Ontwikkel fase

8.1 Het opzetten van de testopstelling

Voor het opzetten van de test opstelling waren er een aantal benodigdheden. Dit waren:

- 1 Desktop PC
- 1 Server
- 1 Cisco Catalyst 2950 switch

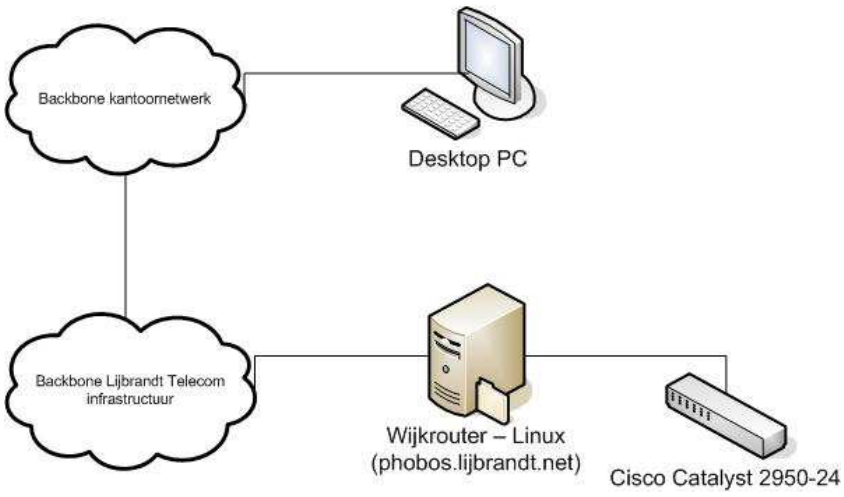
Verder zijn er om de software te kunnen ontwikkelen een aantal pre condities waaraan de server moet voldoen. Onder andere om te zorgen dat de server verbonden is met alle relevante netwerken die de wijkrouter moet kunnen bereiken.

Ook dient de server voorzien te zijn van het Linux operating system. Dit operating system is in principe kosteloos. De licentie voor dit operating system bestaat uit de GNU GPL ofwel de GNU General Public License. Dit zorgt ervoor dat degene die de software gebruikt dezelfde rechten moet verlenen aan de afnemer van de software als dat de gebruiker gekregen heeft.

Dit zorgt ervoor dat Lijbrandt Telecom gebruik kan maken van deze software zonder dat hieraan kosten verbonden zijn. Ook bezit dit operating system alle software die nodig is om als bouwstenen te dienen voor de wijkrouter. Als test operating software heb ik gekozen voor de Fedora Core 3 linux distributie. Dit omdat deze de eenvoudigste installatie procedure heeft, en ik daardoor snel tot resultaten kan komen.

Voor de Desktop PC is Microsoft Windows XP geïnstalleerd. Deze was al geïnstalleerd doordat er een nieuwe PC voor mij was besteld voordat ik aankwam op het kantoor van Lijbrandt Telecom.

Als laatste is er de Cisco Catalyst 2950 switch. Deze switch heeft 24 poorten en zal woren aangesloten op de wijkrouter. De test opstelling ziet er nu als volgt uit:



Figuur 8.1: Testomgeving

Binnen deze opstelling is het mogelijk om door middel van SSH (Secure Shell) een verbinding te openen op de Linux router om hier de software op te ontwikkelen. En de switch te benaderen om deze te configureren. De SSH verbinding die ik gebruikte heb ik opgezet door middel van de SSH client Putty.

Figuur 8.2: Foto testomgeving

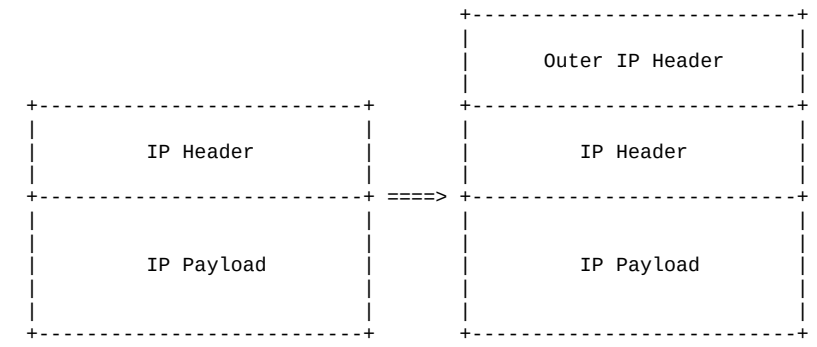
8.2 opzetten van een IPIP tunnel naar het Lijbrandt Telecom netwerk

In sommige gevallen is het voor de wijkrouter de bedoeling dat deze geplaatst wordt in een vreemd netwerk. Dit kan gebeuren doordat er een andere ISP wordt ingehuurd om er voor te zorgen dat er voldoende bandbreedte is op een locatie. Dit heeft meestal tot gevolg dat Lijbrandt Telecom geen gebruik kan maken van haar eigen IP reeksen. Wat inhoudt dat er een oplossing moest worden gezocht om het mogelijk te maken mensen op locatie een Lijbrandt Telecom IP adres te geven.

Om dit te kunnen doen is het mogelijk om een tunnel te maken vanaf het ‘vreemde’ netwerk naar het netwerk van Lijbrandt Telecom. Dit kan middels een VPN bijvoorbeeld. Om dit in Linux te kunnen regelen heb ik na dit onderzocht te hebben, uitgevonden dat PPTP(Point to point tunneling protocol) een mogelijkheid is, maar niet de makkelijkste optie is om een tunnel tot stand te brengen. Dit is een vrij ‘ingewikkeld’ protocol die meer doet dan ik wilde. Het enige dat ik wilde was een tunnel die het ene met het andere netwerk verbind. Een IPIP(IP Encapsulation within IP) verbinding was hierdoor ook een optie. Dit protocol werkt veel simpeler dan een PPTP verbinding. Het grootste verschil is dat een IPIP verbinding geen authenticatie vereist. En IPIP is een stateless protocol, oftewel er wordt niets bijgehouden. Er wordt simpelweg een deel aan het packet vastgeplakt met header informatie. Binnen de RFC is dit gedefinieerd als volgt:

```
3. IP in IP Encapsulation

   To encapsulate an IP datagram using IP in IP encapsulation, an outer
   IP header [10] is inserted before the datagram's existing IP header,
   as follows:
```

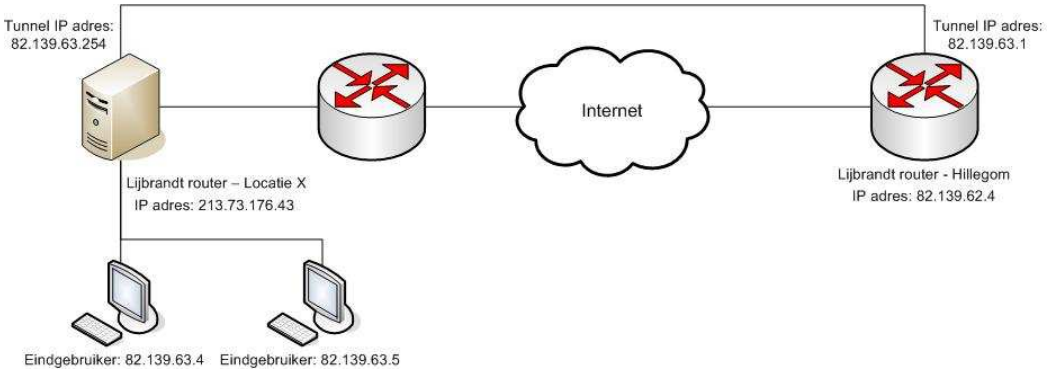


In mijn test opstelling was een tunnel niet nodig aangezien mijn wijkrouter direct op de Lijbrandt Telecom backbone was aangesloten. Echter vond ik het wel van belang om hier rekening mee te houden zodat deze mogelijkheid beschikbaar zou zijn. wanneer er een hard disk image zou worden gemaakt van de wijkrouter. Dus wanneer er een tunnel moet worden gemaakt dan gebeurd dit via een IPIP verbinding.

De mogelijkheden tot een IPIP verbinding zitten in de Linux kernel. Er hoeft dus geen aparte software te worden geïnstalleerd. Er is alleen een configuratie nodig om dit voor elkaar te krijgen. Allereerst is er de module van IPIP nodig, welke mee gecompileerd wordt met de Linux kernel. Deze heet CONFIG_NET_IPIP. In mijn geval was deze module al mee gecompileerd met de distributie die ik geïnstalleerd

had. Dan is het programma ‘iproute’ nodig. Ook dit programma is meestal al aanwezig bij een standaard linux distributie.

Om het principe aan te geven hoe een tunnel opereert geef heb ik dit weergegeven in figuur 8.3.



Figuur 8.3: schematische weergave van een IPIP tunnel implementatie

Om het concept uit figuur 8.3 toe te passen zal ik een voorbeeld geven. Het is in Fedora mogelijk om via een user interface een IPIP tunnel op te zetten. Wanneer dit niet het geval is kan er via bepaalde configuratie bestanden een IPIP tunnel worden opgezet. Aangezien ik zo een kaal mogelijke distributie wil hebben van Linux(zodat deze zo snel mogelijk te installeren is en als image op één cd past) is het nodig om dit via configuratiebestanden te configureren via de Linux console.

Om dit te doen dienen er op twee plaatsen interfaces geconfigureerd te worden. Deze interfaces hebben een IP adres dat binnen hetzelfde subnet valt. Hierdoor zijn ze in staat om met elkaar te communiceren. Dit ziet er als volgt uit:

Router 1

```
ip tunnel add <name> mode ipip remote 192.0.2.69 local 192.0.2.34
ip link set <name> up
ip addr add 192.168.1.1/24 dev <name>
```

Router 2

```
ip tunnel add <name> mode ipip remote 192.0.2.34 local 192.0.2.69
ip link set <name> up
ip addr add 192.168.1.254/24 dev <name>
```

Hierna is het nodig om in te stellen dat de interface bij het opstarten automatisch wordt aangemaakt. En dienen er nog een aantal configuratieslagen gemaakt te worden. Dit gebeurd door een configuratie bestand aan te passen. Deze staan bij zo goed als alle Linux distributies op een andere plek. In mijn testsituatie was de locatie van het desbetreffende bestand `/etc/network/interfaces`. Ook hier dienen de instellingen op beide plaatsen op deze manier gewijzigd te worden. Ik ga er hierbij vanuit dat de router waarmee de tunnel wordt gemaakt ook een Linux machine is.

Router 1

```
auto <name>
iface <name> inet static
  address 192.168.1.1
  netmask 255.255.255.0
  network 192.168.1.0
  broadcast 192.168.1.255
  pre-up /sbin/ip tunnel add <name> mode ipip remote 192.0.2.69 local
192.0.2.34
  post-down /sbin/ip tunnel del <name>
```

Om deze interface daadwerkelijk te starten op deze machine dient het commando `ifup` gegeven te worden gevolgd door de naam van de tunnel.

i.e.
```
ifup LaanVanAngers
```

Op de andere router dient hetzelfde te gebeuren.

Router 2

```
auto <name>
iface <name> inet static
  address 192.168.1.254
  netmask 255.255.255.0
  network 192.168.1.0
  broadcast 192.168.1.255
  pre-up /sbin/ip tunnel add <name> mode ipip remote 192.0.2.34 local
192.0.2.69
  post-down /sbin/ip tunnel del <name>
```

Ook nu dient de interface te worden gestart met het `ifup` commando.

i.e.
```
ifup LaanVanAngers
```

Hierna moet als alles goed is gegaan, het pingen van elkaars tunnel adres(192.168.1.1 en 192.168.1.254) mogelijk zijn.

8.3 Het ophalen van administratieve gegevens over een beveiligde verbinding

Om gegevens van de database server naar de linux router te transporteren heb ik gebruik gemaakt van een standaard SSH2 verbinding. Deze Secure Shell verbinding is beveiligd door middel van een asymmetrische sleutel. Om aan te geven wat ik precies wilde bereiken heb ik hier een afbeelding geplaatst met de situatie zoals hij was op dat moment.

Figuur 8.4: Schematische weergave van de beveiligde verbinding

Dit houdt in dat ik door middel van een SSH verbinding een programma kan starten op de database machine en hiervan de output op kan opvangen. In de 'oude' wijkrouter was het nodig om door middel van een PERL module in te loggen op de database server om hier gegevens vandaan te halen. Ik wilde dit anders doen. Ik wil de situatie dusdanig krijgen dat ik zonder een wachtwoord in te voeren kan inloggen op de database server.

Dit alles klinkt heel onveilig maar dat is het niet. Ik heb uitgezocht dat als ik een private sleutel opsla op de linux router en een public sleutel opsla op de database server ik in staat ben zonder wachtwoord in te loggen op de database server. Dit betekent dat ik in staat ben om in te loggen op de database server indien ik in het bezit ben van een private sleutel en de juiste gebruikersnaam. Daarnaast is het ook mogelijk om een hostname aan te geven. Dit betekend dat alleen iemand van het aangegeven host adres toegang heeft tot de database server. Dit heb ik in mijn testopstelling nog niet gedaan om zo de mogelijkheid te houden om het IP adres en hostname te veranderen. Uiteindelijk is het verstandig om dit als extra beveiliging maatregel in te stellen.

Om dit geheel op te zetten in linux heb ik gebruik gemaakt van het programma 'ssh-keygen'.

Om een indruk te geven hoe dit gaat volgt hier het commando en de prompt vragen die mij gesteld werden bij het instellen van zo een verbinding. De router in mijn testomgeving heette overigens 'phobos.lijbrandt.net'.

```
[root@phobos ~]# ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
/root/.ssh/id_test
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_test.
Your public key has been saved in /root/.ssh/id_test.pub.
The key fingerprint is:
ba:36:92:61:31:2b:94:32:dc:c8:7b:84:bc:c1:d8:52
root@phobos.lijbrandt.net
[root@phobos ~]#
```

Nu was het de bedoeling om de public key die ik zojuist gegenereerd heb, naar de database server te kopieren in het ‘known_keys’ bestand. Om dit te kunnen doen dien ik eerst als root in te loggen op de database server en de public key toe te voegen aan het known_keys bestand. Dit ging als volgt:

```
echo ssh-rsa "publiekesleutel" >> ~/.ssh/authorized_keys chmod
600 ~/.ssh/authorized_keys
```

Dit is eigenlijk een simpele manier om in één commando een bestand in /root/.ssh/ te openen in append mode dat ‘authorized_keys’ heet om er vervolgens een reeks karakters aan toe te voegen. Ik geef door middel van ‘ssh-rsa’ aan om wat voor type key het gaat en gevolgd door een spatie de public key reeks. Een key als deze ziet er als volgt uit:

```
---- BEGIN SSH2 PUBLIC KEY ----
Comment: "rsa-key-20050314"
AAAAB3NzaC1yc2EAAAABJQAAAIEAi4VZg+0hFY5c2YLocdZvkhfo9z233qxRA0
uPFRUPZl0cYfG4WbTb+z65D6E6qUQVU+9lGh4sTpxk9zcWFXI4HXzyQBpW4R8v
EE/KKAKhLI58faMJkv6uFLVx5ARSaSALD0pk8ckAjEHYofsIfnaHdunI8PP3gX
WI5xXRLqNcgPU=
---- END SSH2 PUBLIC KEY ----
```

Uiteindelijk heb ik de situatie zo gekregen dat ik in staat ben om zonder wachtwoord in te loggen. Waardoor ik in staat was om met één commando zonder tussenkomst van verdere keyboard interactie een commando op de database machine uit te voeren.

Dit had als groot voordeel dat ik vanuit PERL geen aparte en vaak door iemand anders gemaakte module nodig had om informatie uit de database te halen. Het grootste voordeel van een aparte PERL module is de gemakkelijke manier waarop er geprogrammeerd kan worden. Echter zou ik dan de module in PERL geïnstalleerd moeten hebben om de router applicatie te kunnen draaien. Op deze manier is dat dus niet meer nodig en op deze manier is de versiegevoeligheid veel minder. De functionaliteit van een module kan veranderen. Wat betreft de functionaliteit van de SSH daemon onder linux is het minder aannemelijk dat deze op korte termijn veranderd.

Als laatste heb ik getest of alles wat ik gedaan had ook werkte. Dit ging als volgt. Ik heb deze werkwijze toegepast op mijn test router om te zien of ik met één commando een ander commando op een andere machine kon uitvoeren.

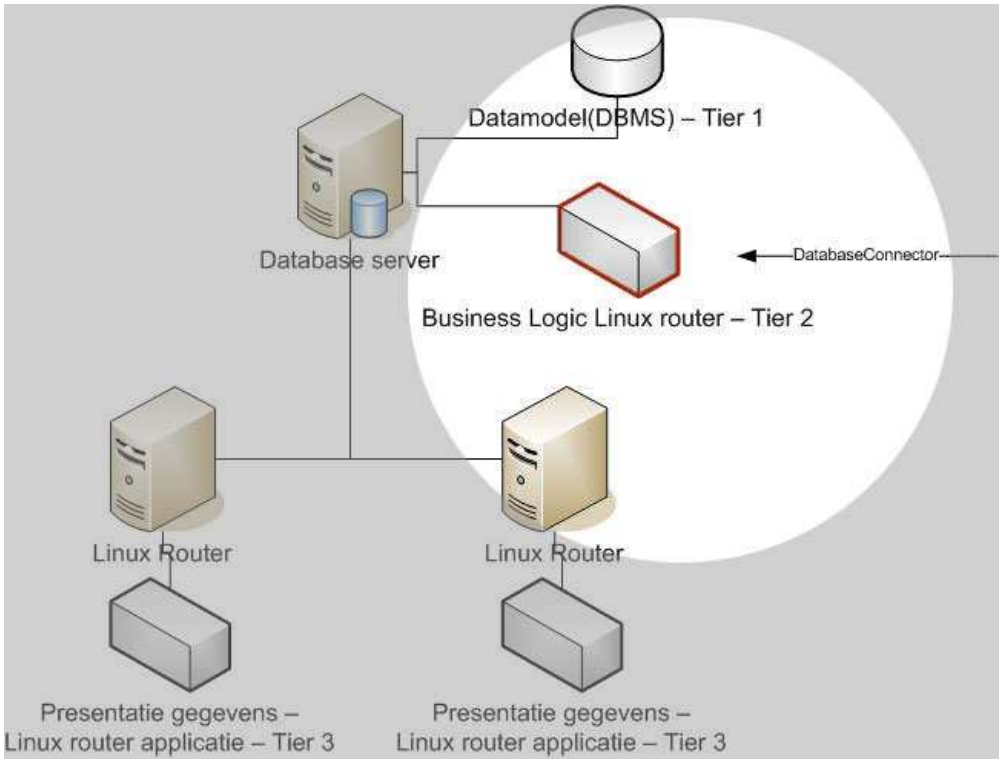
```
[root@phobos ~]# ssh zeus.lijbrandt.net ls
Using username "root".
Authenticating with public key "rsa-key-20050314" from agent
Last login: Fri Mar 25 17:56:22 2005 from ip82-139-66-
94.lijbrandt.net
anaconda-ks.cfg
hldsinstall
install.log
install.log.syslog
[root@phobos ~]#
```

Mijn bedoeling hiermee was om in plaats van een directory list op te vragen een ander PERL script aan te roepen dat als output gegevens uit de database terug geeft. Dit zit dan ook verwerkt in de applicatie.

Ik ben aan deze oplossing gekomen doordat ik in een SSH client genaamd ‘Putty’ heb gezien dat er een mogelijkheid bestond om met ‘Pagent’ automatisch in te loggen op een Linux machine. Hierna heb ik uitgezocht hoe dit mogelijk was, en of dit ook veilig was. Uiteindelijk heb ik getest of ik vanaf een Windows machine kon inloggen met in het bezit, een private key. Dit was mogelijk, en daarna heb ik met behulp van het internet gezocht naar de equivalent van deze oplossing onder Linux.

8.4 Iteratie: DatabaseConnector

Bij de implementatie van de DatabaseConnector is het van belang om aan te geven dat het hier om een procedureel script gaat. Dit script welke wordt gestart door de Linux router applicatie, zorgt ervoor dat informatie uit de database wordt gehaald en daarna in de console afgedrukt wordt. De linux router applicatie zal met die informatie een configuratie uitvoeren. Om aan te geven waar in de oplossing deze DatabaseConnector zich bevind heb ik dat aangegeven op de onderstaande afbeelding.



Figuur 8.5: Verduidelijking van het in deze paragraaf ontwikkelde deel van de wijkrouter software

Dit deel van de oplossing is het 'server' deel. Dit houdt in dat de server continu benaderd moet kunnen worden om informatie op te halen. Het script zal dus nooit eindigen. Tenzij uiteraard het proces wordt afgebroken. Om deze gegevens over te halen naar de Linux router applicatie is het nodig om dit te doen over een beveiligde verbinding. Dit heb ik uitgelegd eerder in dit verslag. Via die methode zal de DatabaseConnector worden uitgevoerd en informatie over de configuratie implementeren.

Ook zal bij het aanroepen van de DatabaseConnector een parameter moeten worden meegegeven. Deze heet het 'trunk_tag' wat staat voor een verzameling verbindingen in de Lijbrandt Telecom Internet Administratie. Dit wordt gerepresenteerd door een afkorting van het project. Een typische parameter voor de DatabaseConnector is 'lva'. Dit staat voor een project naam waar Lijbrandt Telecom internet biedt.

Omdat dit de eerste keer is dat ik met PERL ga programmeren heb ik tijd nodig gehad om de taal te leren. Omdat ik wel een idee had hoe de structuur van het programma er uit zou gaan zien, heb ik een PSD(Program Structure Diagram) gemaakt. Dit PSD laat de structuur zien van het DatabaseConnector script.

Figuur 8.6: PSD(program structure diagram) van de DatabaseConnector

Het idee achter dit PSD was om ervoor te zorgen dat ik een blauwdruk had waarover ik kon praten met de opdrachtgever en wat technische mensen. Ik kon door middel van dit diagram gericht vragen stellen naar de oplossing hiervan in PERL. Uiteraard zijn hier een aantal aannames gedaan waarvan ik niet zeker wist of deze handig dan wel mogelijk waren binnen PERL. Zoals bijvoorbeeld er van uit gaan dat ik met 1 query alle benodigde informatie uit de database kon halen. Dit bleek dan ook in eerste instantie erg lastig. Hierdoor was ik genoodzaakt om het PSD steeds bij te werken, totdat er een bevredigend niveau was behaald en het programmeren vergemakkelijkte.

8.4.1 Het programmeer proces

Bij het programmeren van de DatabaseConnector werd ik als eerste geconfronteerd met het feit dat ik nog niet wist hoe ik met PERL moest werken. Als eerste heb ik daarom een tutorial gevolgd en bekeken hoe ik bepaalde acties moest doen in PERL.

Ik had echter al wel ervaring met PHP. Deze taal is in vele opzichten te vergelijken met PERL. Het gaat hier om een in principe procedurele taal, die in vele vormen bepaalde zaken van andere talen heeft overgenomen. Voornamelijk van de taal C en C++ maar ook van bijvoorbeeld functionele talen.

Echter, toen ik werkelijk begon met programmeren was het vrij lastig om er achter te komen hoe bepaalde zaken werkte. Zo ben ik echt een hele tijd bezig geweest om te achterhalen hoe ik in PERL een functie, parameters mee kon geven. Normaal gesproken in een programmeertaal als Java gaat dit als volgt:

Java code:

```
public void runProgram( String tekst, int startByte)
{
  return tekst.substring(startByte);
}

//En dan met de aanroep:

runProgram("Het programma wordt gestart.",1);
```

Dit is overigens ook de manier waarop veel andere talen dit oplossen. Echter in PERL is er een aparte wijze waarop parameters worden gerefereerd. Dit gaat als volgt:

PERL code:

```
sub getValue
{
      my ($list_item_1, $list_item_2) = @_;
      $temp_value = $list_item_1 + $list_item_2;
      return $temp_value;
}

#De aanroep van deze functie is dan:

&getValue(1,2);
```

Dit is slechts nog maar één van de verschillen waar ik mee te maken kreeg. Om dit goed door te krijgen heb ik daarom een aantal test programma's geschreven om bepaalde problemen die ik voorzag bij de ontwikkeling van de DatabaseConnector te simuleren en op te lossen.

Ik kon dit uiteraard doen op basis van het PSD dat ik gemaakt had en daardoor was het zoeken naar de oplossing binnen PERL een stuk vereenvoudigd. Echter is er een gedeelte van de ontwikkeling waar weinig informatie over bekend was. Dit is het gebruiken en zelf creëren van modules.

In het geval van de DatabaseConnector was het alleen nodig om gebruik te maken van de modules die al bestonden in de CPAN[9] database met PERL modules. Deze CPAN database bestaat uit een collectie van modules die zijn toegevoegd door gebruikers van de taal PERL. Dit voorkomt bijvoorbeeld dat ik zelf relatief grote oplossingen zou moeten bedenken voor gestelde problemen.

Zo is er het Select object dat gebruikt wordt in de DatabaseConnector. Hiervoor dient een module geïnstalleerd te zijn. Ook is er de DBI module die nodig is voor de verbinding met de database. Nu had ik een aantal opties om dit probleem op te lossen. Ik kon via de shell van CPAN een module downloaden en installeren. Dit heb ik dan ook geprobeerd. Dit ging als volgt:

```
[root@phobos var]# perl -MCPAN -e shell
```

Door middel van dit commando is het de bedoeling dat er een verbinding wordt geopend met een CPAN mirror server. Hierop is de hele collectie van modules te vinden. De theorie zegt dan dat ik door middel van dit commando in staat ben om een module te downloaden, te compileren en uiteindelijk te gebruiken.

```
[root@phobos var]# perl -MCPAN -e shell
Terminal does not support AddHistory.

cpan  shell  --  CPAN  exploration  and  modules  installation
(v1.7601)
ReadLine support available (try 'install Bundle::CPAN')

cpan> install LWP
CPAN: Storable loaded ok
Going to read /root/.cpan/Metadata
  Database was generated on Mon, 28 Mar 2005 08:54:09 GMT
CPAN: LWP::UserAgent loaded ok
Fetching with LWP:
  ftp://ftp.perl.org/pub/CPAN/authors/01mailrc.txt.gz
Going to read /root/.cpan/sources/authors/01mailrc.txt.gz
Fetching with LWP:
…
…
…
```

Op deze shell kan ik dus aangeven dat ik de module DBI wil downloaden en installeren. Echter wanneer ik dit deed kreeg ik veel afhankelijkheden te zien die niet geïnstalleerd waren. Deze afhankelijkheden zijn ook de reden dat ik vond dat het een te groot onderzoek zou worden om dit pad nog verder te volgen. Er zit volgens de theorie weliswaar een automatische installatie methode op alleen toen ik dat probeerde ging dit fout.

[9] Comprehensive Perl Archive Network – Een archief met veel PERL modules.

Ik ben toen verder gaan zoeken naar een simpelere oplossing om toch PERL dusdanig geconfigureerd te krijgen om het gebruik van de module DBI mogelijk te maken. Het voordeel dat een module als DBI heeft is dat deze veel gebruikt wordt en dat hiervoor een aparte PERL distributie bestaat. Daar kwam ik achter toen ik naar een oplossing zocht voor mijn probleem op internet.

Elke Linux distributie heeft wel een manier om snel software te downloaden en te installeren. Ik zelf heb voor dit project de Fedora Core 3 linux distributie gebruikt en deze heeft een update client genaamd ‘yum’. Door middel van dit programma was ik in staat om de distributie van PERL op te waarderen naar een volgende versie en een versie te kiezen waar DBI al bij zat.

Ik deed dit door yum te draaien met het zoek commando. Dit ging als volgt:

```
[root@phobos var]# yum search dbi
Searching Packages:
Setting up Repos
base                      100% |=========================| 1.1 kB    00:00
updates-released          100% |=========================|  951 B    00:00
Reading repository metadata in from local files
base      : ########################################################## 2622/2622
updates-re: ########################################################## 862/862
```

hierdoor deed ik een zoek opdracht en kon ik er achter komen dat ik het volgende pakket nodig had:

```
perl-DBD-Pg.i386                          1.31-6                 installed
Matched from:
This package contains an implementation of DBI for PostgreSQL for
Perl.
```

Deze kon ik dan installeren door middel van het commando:

```
[root@phobos var]# yum install perl-DBD-Pg
```

Hierna zal het pakket worden geïnstalleerd en is dan ook klaar voor gebruik op deze Linux distributie.

8.4.2 Het opzetten van de verbinding met de database

Zoals ik in de vorige paragraaf heb beschreven zijn nu de benodigdheden compleet voor het opzetten van een verbinding met de database. Het eerste probleem waar ik mee geconfronteerd werd was dat ik niet wist wat voor soort database PostgreSQL was. Één ding wist ik wel zeker, dit was dat er gebruik werd gemaakt van een SQL interface om met de database te praten. Ook is mij verteld door ervaren mensen bij Lijbrandt Telecom dat ik de module DBI in PERL moet gebruiken.

Ik heb daarom dan ook gelijk een test script gemaakt om een database verbinding op te zetten.

code:

```
# Verbinding via DBI met PostgreSQL
my $db_uri = "dbi:Pg:dbname=inetadmin";
my $db = DBI->connect($db_uri, undef, undef, {RaiseError => 1}) || die
"Error connecting to the database: $DBI::errstr\n";
```

Deze code heb ik door een combinatie van tutorials van het internet en informatie van mensen ter plaatse bij Lijbrandt Telecom kunnen samenstellen. Overigens is er een hele hoop informatie te vinden via de manual pages die beschikbaar zijn via het Linux operating system.

Deze zijn te benaderen door middel van:

```
[root@phobos var]# man DBI
```

Dit geeft dan een volledige uitleg over de syntax en het gebruik van DBI.

```
DBI(3)                          User  Contributed  Perl  Documentation
DBI(3)

NAME
       DBI - Database independent interface for Perl

SYNOPSIS
         use DBI;

         @driver_names = DBI->available_drivers;
         @data_sources = DBI->data_sources($driver_name, \%attr);
---
```

Door middel van deze pagina's heb ik kunnen uitzoeken hoe ik het snelst de informatie uit de database kon krijgen en verwerken, zonder dat ik ontzettend veel stringbuffers of arrays moest toepassen.

Dit heeft als voordeel dat ik op het niveau van PERL snelheidswinst oplevert. Aangezien die functies allemaal op een lager abstractie niveau worden uitgevoerd. Door middel van al gecompileerde software.

8.4.3 Het ophalen van de configuratie uit de database

Het volgende dat erg lastig in elkaar zat was het samenstellen van de query waarmee ik de database mee zou gaan bevragen. Om dit te kunnen doen moest ik eerst precies weten wat ik uit de database wilde hebben. Hierbij kon ik gebruik maken van het al bestaande ERD en het gemaakte klassendiagram. Er zijn 2 queries in de DatabaseConnector. Waarvan er één dient als buffer om de gegeven parameters om te zetten in bruikbare gegevens voor de software. Aangezien de DatabaseConnector wordt aangeroepen met een zogenaamd 'Trunk_ID' dit is een project ID waarop de instantie van de DatabaseConnector moet gaan opereren.

Bijvoorbeeld:

code:

```
[root@phobos /]# ./databaseconnetor lva
```

Waarbij de parameter 'lva' staat voor het project 'Laan van Angers'. Dit is een plaats waar de linux router draait.

Het gaat om deze query:

```
SELECT trunk_id, inet_config
FROM trunk
WHERE tag = $trunk
```

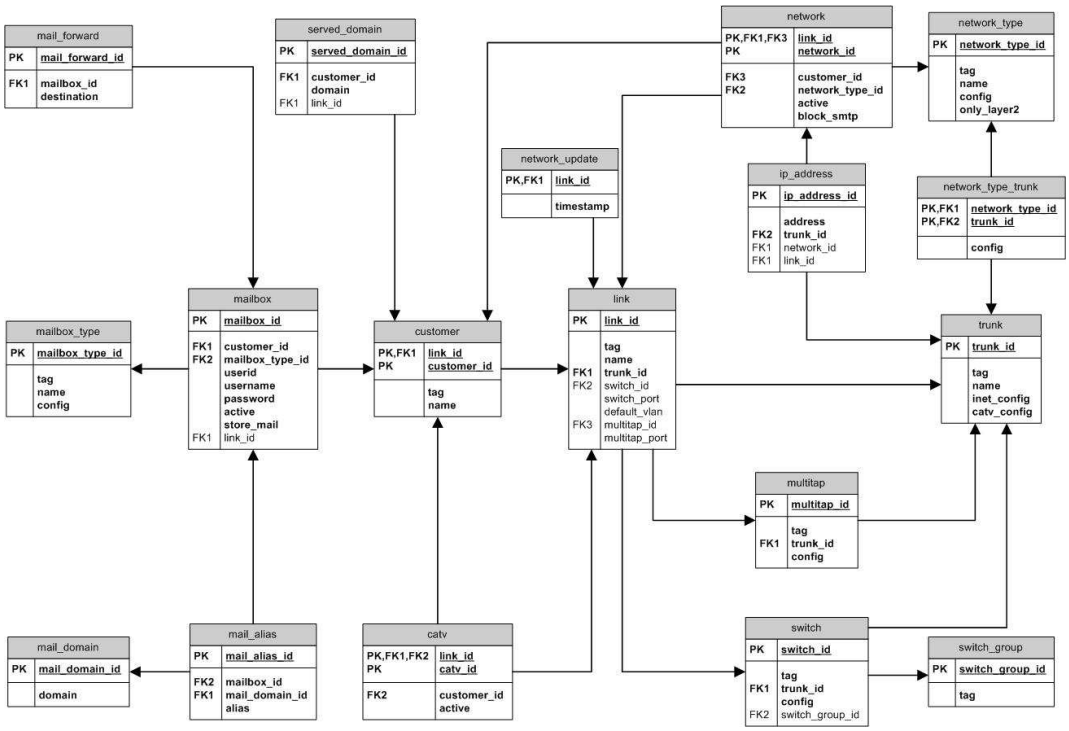
Door middel van deze query wordt gekeken of het inderdaad een bestaande configuratie is waarom wordt gevraagd. Als deze query niets teruggeeft dan zal het programma afbreken met de melding dat de gevraagde configuratie niet beschikbaar is. Dit is de reden waarom deze query apart gebeurd van een query waarbij de hele configuratie wordt opgehaald.

Het model waarop deze queries gebaseerd zijn was al aanwezig echter ben ik tijdens de implementatie er achter gekomen dat dit model niet compleet was. Dit kwam omdat het aanwezige model weliswaar geschikt was voor de Lijbrandt Telecom Internet Administratie maar niet 100% compleet was voor de nieuwe wijkrouter.

Ik heb hierna samen met Joost Vermeer gekeken welke velden ik nodig had en waar deze zich bevonden in de database. Op die manier kon ik door middel van trial en error een query samenstellen en zo de configuratie uit de database kon halen.

Het gevolg hiervan was overigens wel dat ik ook het model moest aanpassen dat tot nu toe alleen voor de Lijbrandt Telecom Internet Administratie werd gebruikt.

Uit dit model miste de tabel `network_update` met het attribuut `timestamp`. Dit model is vrij groot en is een ontwerp voor de Lijbrandt Telecom Internet Administratie. Hierdoor mocht ik niet zomaar een wijziging doorvoeren. Dit heb ik in samenspraak gedaan met de mensen die verantwoordelijk waren voor dit ontwerp. Na de wijziging te hebben doorgevoerd zag het ontwerp er als volgt uit.



Figuur 8.7: Gewijzigd ERD van de Lijbrandt Telecom Internet Administratie database.

Hieruit is de volgende query afgeleid.

```
SELECT
   link.link_id,
   link.default_vlanid AS vlanid,
   CASE
     WHEN network_type.only_layer2 THEN FALSE
     WHEN ip_address.address IS NULL THEN FALSE
     WHEN network.active THEN TRUE
     ELSE FALSE
   END AS active,
   network.block_smtp,
   network_type.config,
   ip_address.address AS ip_address,
   network_update.timestamp AS last_update
 FROM link
   LEFT JOIN network USING (link_id)
   LEFT JOIN network_type USING (network_type_id)
   LEFT JOIN ip_address USING (network_id)
   LEFT JOIN network_update USING (link_id)
 WHERE link.trunk_id = ?
   AND (? OR network_update.timestamp > ?)
 ORDER BY link.link_id, ip_address.address";
```

Door middel van deze query worden alle benodigde gegevens uit de database gehaald. Het lastige hierbij is om te achterhalen wanneer deze query gedraaid moet worden. Het was niet de bedoeling dat deze query gepolled[10] werd. Maar dat er real-time per wijziging een query op de database gedaan wordt. De oplossing hier voor was het gebruik van een timestamp in een veld dat bij elke wijziging in de Lijbrandt Telecom Internet Administratie wordt bijgewerkt en de wijkrouter aanroept om de wijzigingen door te voeren.

Deze timestamp wordt bijgewerkt na elke update door een trigger binnen het PostgreSQL ORDBMS. Omdat ik niet wist hoe deze triggers werkte heb ik gevraagd aan een ontwerper van de Lijbrandt Telecom Internet Administratie hoe ik dit moest doen. Het bleek dat ik via de PosgreSQL console een trigger kon maken met het commando 'CREATE TRIGGER'. Dit kon dan op basis van verschillende event handlers zoals: DELETE, UPDATE en INSERT op een tabel.

Ik mocht zelf geen wijzigingen in de L.T.I.A. database maken, ik heb wel het ontwerp van de database zelf kunnen aanpassen. Dit had tot gevolg dat ik door middel van technische werknemers van Lijbrandt Telecom heb kunnen achterhalen hoe deze triggers tot stand zijn gekomen en hoe ik deze kon gebruiken. Echter als onderdeel van de DatabaseConnector was het alleen van belang dat de triggers aanwezig waren en dat ik wist wat dit betekenden voor de later op te leveren DatabaseConnector.

De functie van een trigger is alleen het aanroepen van een functie binnen PostgreSQL en niet het uitvoeren van SQL statement. Dit betekend dat er twee aparte configuratie items zijn.

- triggers
- functions

Allereerst dienen de functies gedefinieerd te worden. Deze zijn geïmplementeerd door de mensen achter de Lijbrandt Telecom Internet Administratie. Ik heb hier zelf niets aan gedaan. Omdat ik er wel mee moest werken vond ik het wel nuttig om te weten hoe dit werkte.

De onderstaande functie is er één die er voor zorgt dat het timestamp veld in de tabel network_update bijgewerkt wordt met de huidige datum en tijd. Dit gebeurd per link van de debetreffende trunk.

```
CREATE FUNCTION network_update (integer) RETURNS void
    AS
'BEGIN
  IF NOT EXISTS(SELECT *
                FROM network_update
                WHERE link_id = $1) THEN
    INSERT INTO network_update (link_id, timestamp) VALUES ($1, now());
  ELSE
    UPDATE network_update SET timestamp = now() WHERE link_id = $1;
  END IF;
  RETURN NULL;
END;'
    LANGUAGE plpgsql STRICT;
```

[10] Polling – een methode om met een bepaalde frequentie een database te bevragen.

De volgende functie zorgt er voor dat het duidelijk is van welke link de timestamp bijgewerkt moet worden.

```
CREATE FUNCTION network_update () RETURNS "trigger"
    AS
'DECLARE
  id link.link_id%TYPE;
BEGIN
  IF TG_RELNAME = ''network'' THEN
    IF TG_OP = ''INSERT'' OR TG_OP = ''UPDATE'' THEN
      PERFORM network_update(NEW.link_id);
    END IF;
    IF TG_OP = ''DELETE'' OR TG_OP = ''UPDATE'' THEN
      PERFORM network_update(OLD.link_id);
    END IF;
  ELSIF TG_RELNAME = ''ip_address'' THEN
    IF TG_OP = ''INSERT'' OR TG_OP = ''UPDATE'' THEN
      id := (
        SELECT link_id
        FROM network
        WHERE network_id = NEW.network_id);
      PERFORM network_update(id);
    END IF;
    IF TG_OP = ''DELETE'' OR TG_OP = ''UPDATE'' THEN
      id := (
        SELECT link_id
        FROM network
        WHERE network_id = OLD.network_id);
      PERFORM network_update(id);
    END IF;
  END IF;
  NOTIFY network_update;
  RETURN NULL;
END;'
    LANGUAGE plpgsql SECURITY DEFINER;
```

Hieronder worden de verschillende triggers gemaakt op respectievelijk de tabel ‘network’ en de tabel ‘ip_adress’.

```
CREATE TRIGGER network_update
    AFTER INSERT OR DELETE OR UPDATE ON network
    FOR EACH ROW
    EXECUTE PROCEDURE public.network_update ();

CREATE TRIGGER network_update
    AFTER INSERT OR DELETE OR UPDATE ON ip_address
    FOR EACH ROW
    EXECUTE PROCEDURE public.network_update ();
```

8.4.4 Het afdrukken van de configuratie op de command line

De bedoeling was om de informatie die de query teruggaf te verwerken en op de command-line te printen. Dit om zo de Linux router applicatie te voorzien van de gegevens die het nodig heeft om de wijkrouter te configureren.

De bedoeling is om per link alle configuratie items af te drukken op de Linux console. De output van de DatabaseConnector ziet er als volgt uit.

```
CONFIG link98.burst_down: 131072
CONFIG link98.rate_down: 131072
CONFIG link98.tag: 0.1.1.1.15
CONFIG link98.vlanid: 115
CONFIG link98.burst_up: 32768
CONFIG link98.block_smtp: 0
CONFIG link98.active: 1
CONFIG link98.ip_addresses: 10.0.0.112
NOTIFY link98
```

In de broncode van de DatabaseConnector komt het printen van de query output terug als volgt.

code:

```
my $row = $query_links->fetchrow_hashref();

  while($row) {
    my $link_id = $row->{link_id};

    $last_update = $row->{last_update}
      if defined $row->{last_update}
        && (!defined $row->{last_update}
        || $row->{last_update} gt $last_update);

    print "CONFIG link$link_id.trunk_id: trunk", $row->{trunk_id},
"\n";
    print "CONFIG link$link_id.vlanid: ", $row->{vlanid}, "\n";
    print "CONFIG link$link_id.active: ", ( $row->{active} ? "1" :
"0" ), "\n";

    if($row->{active}) {
      print "CONFIG link$link_id.block_smtp: ", ( $row->{block_smtp}
? "1" : "0" ), "\n";
      foreach my $line (split /\n/, $row->{config}) {
        print "CONFIG link$link_id.$line\n";
      }

      # geef de ip addressen door
      print "CONFIG link$link_id.ip_addresses:";
  }
```

Het grootste probleem was hierbij dat er in sommige van de database velden data stond die uit meerdere regels bestond. En dus eindigde met een ‘`newline`’. Toen ik probeerde het veld af te drukken op de Linux console merkte ik dat de ‘`newline`’

characters niet automatisch op de juiste manier werden weergegeven. Dit was onder andere het geval bij `$row->{config}`. Om te zorgen dat er wel een `newline` werd weergegeven op de Linux console heb ik handmatig een `newline`(in PERL wordt dit gekenmerkt door `\n`) toegevoegd. De reden waarom dit niet automatisch goed ging was mij niet duidelijk. Ik kon wel de '`newline`' filteren binnen PERL maar deze niet automatisch op de juiste manier afdrukken. Mede hierom heb ik besloten om deze workaround te laten zitten.

Als laatste was het nodig om een NOTIFY commando te geven aan de Linux router applicatie. Dit NOTIFY commando dient als trigger voor de Linux router applicatie om deze link te verwerken. In de broncode komt deze als volgt terug.

Code:

```
if(@links) {
  print "NOTIFY ", join(' ', @links), "\n" || exit 0;
  STDOUT->flush();
}
```

Dit is een simpele constructie om alle te verwerken links af te drukken op de Linux console, zodat er wordt doorgegeven welke links verwerkt moeten worden.

8.5 Iteratie: Linux router applicatie

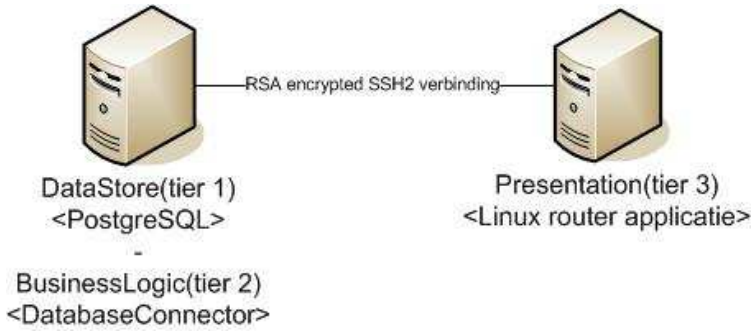
8.5.1 De Werking van de Linux router applicatie

Toen ik begon met het programmeren van de Linux router applicatie was het van groot belang dat de DatabaseConnector af was. Dit was belangrijk omdat ik het formaat nodig had, dat ik door middel van de DatabaseConnector aanbood, aan de Linux router applicatie. Ik kon dit ook doen door een aanname te doen over het formaat, maar ik vond dat ik op deze manier een beter overzicht had over de nog te verrichten werkzaamheden. Om een beeld te geven over wat de DatabaseConnector teruggeeft aan de Linux router applicatie geef ik een configuratie van Link98 weer in de onderstaande afbeelding.

```
CONFIG link98.burst_down: 131072
CONFIG link98.rate_down: 131072
CONFIG link98.tag: 0.1.1.1.15
CONFIG link98.vlanid: 115
CONFIG link98.burst_up: 32768
CONFIG link98.block_smtp: 0
CONFIG link98.active: 1
CONFIG link98.ip_addresses: 10.0.0.112
NOTIFY link98
```

In de bovenstaande afbeelding heb ik de gegevens weergegeven die de DatabaseConnector terug geeft met betrekking tot link98. Deze informatie is per link en wordt in één grote stream verstuurd aan de Linux router applicatie. De Linux router applicatie zal dan de Linux machine configureren met behulp van deze informatie. Om de bovenstaande attributen toe te passen is het nodig om een controle laag te hebben. Deze controle laag wordt gecreëerd door een VLANID toe te wijzen aan een interface op de linux machine. Daarna worden door verschillende software de doorgegeven attributen toegepast op de desbetreffende interface.

De mogelijkheden om van deze informatie bruikbare gegevens te maken zijn gelimiteerd. De meest voor de hand liggende manier is om elke teruggekregen regel van de DatabaseConnector in te lezen ,te interpreteren ,de gegevens te gebruiken en toe te passen op de interface waarop de abonnee op is aangesloten. Om duidelijkheid te scheppen over de huidige situatie heb ik deze geschetst in de volgende afbeelding.



Figuur 8.8: locatie van de in deze paragraaf beschreven wijkrouter software.

De software die in deze iteratie tot stand is gekomen(de Linux router applicatie) roept de DatabaseConnector aan over een beveiligde verbinding. Wanneer dit gebeurd zal de Linux router applicatie de gegevens die de DatabaseConnector terugstuurt opslaan in een bestand en daarna verwerken. Het is van belang om te onderscheiden dat de machine waarop de Linux router applicatie actief is 'Wijkrouter' genoemd wordt en dat dit hoofdstuk de ontwikkeling van de software applicatie beschrijft.

Om de werking van de software vast te leggen heb ik gebruik gemaakt van PSD's. Op deze manier kan ik in een diagram vormgeven hoe de structuur van de software is.

Figuur 8.9: PSD van de controle laag van de in deze paragraaf beschreven delen van de wijkrouter software(ook wel Main klasse)

Deze PSD laat globaal de werking van de Router applicatie zien. Het is niet een PSD die werkelijk de structuur van de uiteindelijk op te leveren software laat zien.

De uiteindelijke implementatie van de gegevens uit de DatabaseConnector op de wijkrouter zelf wordt verzorgd door het Link object. Dit Link object heeft een methode ‘process’ die er voor zorgt dat bepaalde utilities worden uitgevoerd op de linux machine en dat deze wordt uitgevoerd op de juiste interface en parameters.

In deze methode process wordt bekeken of het nodig is om de betreffende link te configureren indien dit het geval is dan zal de methode configure worden aangeroepen. In deze methode zit de meeste intelligentie van de Linux router applicatie. In deze methode worden een reeks van programma’s gedraaid om de wijkrouter te configureren.

Er wordt hier gebruik gemaakt van een aantal verschillende programma’s waaronder:

- **iptables** – om op een de smtp poort te kunnen blocken.
- **vconfig** – om de VLAN’s te configureren
- **ip** – om interfaces te configureren en routes in te stellen.
- **tc(traffic control)** – om per interface bandbreedte beperkingen op te leggen
- **dhcpd** – om per interface een dhcp daemon te draaien die de verbinding automatisch configureerd.

Deze programma’s worden benaderd door middel van een `system()` commando in PERL. In sommige gevallen is dit niet meteen overzichtelijk hoe dit precies gebeurd. In het geval van `dhcpd` is het nodig om per link een configuratie bestand te hebben die dan wordt samengesteld uit informatie die de Linux router applicatie heeft opgehaald uit de DatabaseConnector. Dit gebeurd door de methode `_dhcpd_config()`. Deze methode wordt vanuit de methode `configure()` aangeroepen.

code:

```
# Scrhijf een nieuwe DHCP daemon configuratie weg
open DHCPDCONFIG, ">", "$tempdir/$link.dhcpd.conf";
print DHCPDCONFIG $self->_dhcpd_config();
close DHCPDCONFIG;
# Zorg ervoor dat de leases file bestaat
system "touch $tempdir/$link.dhcpd.leases";
# Start de DHCP daemon
system "dhcpd -q "
  . "-cf $tempdir/$link.dhcpd.conf "
  . "-lf $tempdir/$link.dhcpd.leases "
  . "-pf $tempdir/$link.dhcpd.pid "
  . "$iface";
# Waarschuw als de DHCP daemon niet 0 terug geeft
warn "DHCPD returned $?" if $?;
```

De uitkomst van de functie `_dhcpd_config()` wordt dus weggeschreven in de filehandle `DHCPDCONFIG` die op haar beurt verwijst naar het bestand `$tempdir/$link.dhcpd.conf`. Hierna wordt de dhcp daemon gestart op de desbetreffende interface.

Om te laten zien wat er dan precies in het uiteindelijke bestand wordt weggeschreven geef ik hier weer hoe de inhoud van het dhcp configuratie bestand tot stand komt.

code:

```
# _dhcpd_config genereerd een dhcp configuratie op basis van
dit link object
sub _dhcpd_config {
  my ($self) = @_;

  # Geeft de DHCP configuratie terug als string
  return
    "default-lease-time 300;\n" .
    "max-lease-time 300;\n" .
    "option broadcast-address 255.255.255.255;\n" .
    "option routers " . $self->{GatewayIp} .";\n" .
    "option domain-name-servers "
      . join(',', split(/ /, $self->{DNSServers})) . ";\n" .
    "option domain-name \"" . $self->{DomainName} . "\";\n" .
    "ddns-update-style none;\n" .
    "\n" .
    "subnet  "  .  $self->{Network}  .  "  netmask  "  .  $self-
>{Netmask} . "{\n" .
    join('', map {"  range $_ $_;\n"} @{$self->{IpAddresses}})
.
    "}\n";
}
```

De bedoeling is dat het bestand met deze configuratieregels wordt gevuld. De andere configuratie elementen worden op dezelfde wijze geconfigureerd in de methode `configure()`.

8.5.2 Het Programmeer proces

Om dit deel van de wijkrouter te implementeren heb ik bijzonder veel zelfstudie moeten verrichten. Ondermeer omdat deze software een efficiënt gebruik van de PERL taal vereist. Om deze reden heb ik gebruik gemaakt van de ervaring van Dhr. Joost Vermeer. Hij wist mij in vele gevallen te helpen met programmeer constructies waar ik nog niet genoeg kennis voor bezat.

Om te kunnen praten over de software was het voor mij nodig om de structuur van het programma op papier te zetten. Ik gebruikte hiervoor PSD's. Deze PSD's dienden om met Joost Vermeer te discussiëren over de te nemen route voor de implementatie van de software. Aangezien Joost Vermeer ook zeer druk bezig was moest ik helder en snel een vraag kunnen stellen over mogelijkheden en keuzes in het programmeer proces.

Dit waren echter alleen programmeerproblemen op een hoger abstractie niveau. Ik kwam ook een aantal problemen tegen op een specifieker vlak. Voornamelijk was mijn probleem de 'regular expression'. Aangezien ik in staat moest zijn om teruggekomen informatie van de DatabaseConnector te interpreteren, was het van belang om hier een efficiënte manier voor te vinden. Dhr Joost Vermeer gaf mij de tip om mij te verdiepen in deze 'regular expressions'.

Deze regular expressions zijn een aparte taalt binnen PERL maar ook in andere programmeertalen zijn regular expressions mogelijk. Met deze taal is het mogelijk om de regels te parsen die de DatabaseConnector teruggeeft aan de Linux router applicatie. In het kort is een regular expression een methode om te zien of een stuk tekst aan een bepaald patroon voldoet.

Regular expressions zijn een hulpmiddel wat door de meeste moderne programeer omgevingen wordt aangeboden. Regular expressions stellen mij binnen PERL in staat om op een gemakkelijke manier stukjes tekst te verwerken.

Een voorbeeld:

```
if($line =~ m/^CONFIG ([a-z][a-z0-9_]*): (.*)$/) {
```

Op deze regel zien we het begin van een `if` statement die als conditie de vergelijking tussen `$line` en een regular expression heeft. De `=~` operator zorgt voor het uitvoeren van de regular expression op `$line`. Het resultaat van die `=~` operator wordt dan gebruikt als voorwaarde voor de if. Alle regular expressions voor het controleren of "matchen" zien er in PERL uit als `m/…/`. Binnen de twee slashes komt dan de werkelijke regular expression.

Naast het resultaat of de $line overeenkomt, heeft het uitvoeren van de regular expression nog een paar bijwerkingen. Hier is het belangrijk dat alles wat in de regular expression tussen haakjes staat, na het uitvoeren van de regular expression in de variabelen $1, $2, $3 enz komt te staan.

Deze regular expression heeft de volgende betekenis:

- Het dakje aan het begin van de regular expression moet overeenkomen met het begin van de te matchen string.
- Daarna moeten letterlijk de tekens C, O, N, F, I, G en spatie komen
- Het haakje openen daarna betekent dat de inhoud van dat paar haakjes in $1 moet komen te staan

- De uitdrukking [a-z] betekent dat op die plek een letter moet komen te staan tussen de a en de z (alle kleine letters mogen dus)
- [a-z0-9_]* betekent dat op die plek een kleine letter, een getal of de underscore nul of meer keer mogen voorkomen
- Het) daarna betekent dat $1 nu volledig is
- Vervolgens moeten er een dubbele punt en een spatie komen
- Daarna begint het tweede paar “capturing” haakjes die ervoor zorgen dat de inhoud in $2 komt te staan
- . betekent elk mogelijk character de asterisk die daar opvolgt geeft aan dat er nul of meer van elk mogelijk character mag voorkomen
- Het haakje sluiten geeft aan dat de inhoud van $2 compleet is
- Het $ moet overeen komen met het eind van de te matchen string

Door middel van de bovenstaande werkwijze heb ik de configuratie regels kunnen verwerken tot gegevens die ik kon toepassen op de wijkrouter.

Een volgende probleem dat ik bij deze iteratie tegenkwam was het object georiënteerd programmeren in PERL 5. Dit had ik nog nooit gedaan en vond het ook enorm lastig om mij hiermee bekend te maken. Allereerst was er bij mij grote verwarring omtrent de benaming van klassen en objecten in PERL. Deze heten in PERL modules, maar deze modules zijn klassen in de object oriëntatie theorie. Dit terwijl normaal gesproken modules het herbruikbare onderdeel representeren van een procedurele taal. En dat was in PERL nu net anders dan in een andere taal. Ik kon een klasse herkennen door de constructor in de PERL module. Voor de duidelijkheid geef een voorbeeld van standaard opmaak van een klasse in PERL.

code:

```
package LTIAConnection;

sub new {

}

1;
```

Het is niet noodzakelijk om de methodenaam ‘new’ te gebruiken maar ik vond dit wel zo handig om de object georiënteerde werkwijze zichtbaar te maken. Nu dient dit bestand te worden opgeslagen als `LTIAConnection.pm` en kan vanuit een perl bestand worden benaderd door het commando `use packagenaam`. Dit ziet er als volgt uit.

code:

```
use LTIAConnection;
use Queue;
use Link;
```

Dit is een direct equivalent voor het in Java gebruikte import statement. Wat wel een groot probleem was bij het uitvinden van deze techniek is het `‘return true’` statement aan het einde van een module. Dit ziet er in PERL uit als `‘1;’`. Ik heb daar vrij lang over gedaan om uit te vinden wat het nou precies inhield.

Het betekent dat wanneer een klasse gebruikt wordt, PERL altijd vraagt om een return waarde die '`true`' is. Dit om zo te zien of een module goed geladen is.
Dit is de manier waarop ik uiteindelijk de klassen: Queue, LTIAConnection en Link heb geimplementeerd in de Linux router applicatie.

In de bijlage van dit document is de volledige broncode te vinden, welke is voorzien van commentaar bij elke stap. Op deze manier is het mogelijk om te volgen wat de software precies doet. Ik heb ook voor sommige delen een PSD gemaakt om de structuur vooral bespreekbaar te maken. Deze PSD's zijn niet bedoeld om een accurate weergave te geven van de software en de algoritmes die gebruikt zijn.

8.6 Acceptatietests

De acceptatietests zijn uitgevoerd als blackbox tests voor de software. Dit houdt in dat de opgeleverde software is getest op functionaliteit. De opdrachtgever is dan ook verantwoordelijk voor de beoordeling van de tests. Om de functionaliteit van de software te kunnen testen ben ik samen met de opdrachtgever de user stories afgegaan. Deze waren:

- De wijkrouter moet verkeer kunnen doorvoeren naar abonnees
- De wijkrouter moet kunnen zorgen voor een maximale snelheid van 30 Megabit per seconde per abonnee
- De wijkrouter moet in staat zijn een abonnee aan en af te sluiten
- De wijkrouter moet betrouwbaar en niet storinggevoelig zijn.
- De wijkrouter moet het mogelijk maken om een Globally Routable IP adres toe te kennen aan abonnees.

De wijkrouter moet verkeer kunnen doorvoeren naar abonnees

Om deze test uit te kunnen voeren heb ik de test situatie uitgebreid door een PC aan te sluiten op de Cisco 2950-24 switch. Ik moest in staat zijn om op deze PC een internet pagina te openen via internet explorer. Nadat dit gelukt was heb ik ook de verbinding getest door een 100MB binary bestand te downloaden van een server bij Lijbrandt Telecom. Beide tests waren geslaagd en de opdrachtgever was tevreden met het resultaat.

De wijkrouter moet kunnen zorgen voor een maximale snelheid van 30 Megabit per seconde per abonnee

Om deze test uit te voeren heb ik het bestand 100mb.bin van http://helena.lijbrandt.net binnen gehaald via de inmiddels aangesloten PC welke op de testopstelling was aangesloten. De HTTP verbinding die ik heb geopend gaf een snelheid aan van ongeveer 3 MByte/sec. De opdrachtgever wilde een verbindingsnelheid van maximaal 30Mbps. Dit houdt in dat een maximale snelheid gehaald moet kunnen worden van 3.75MByte/sec. Voor de HTTP verbinding die ik getest had was ongeveer 3 MByte/sec een voldoende snelheid. Uiteindelijk dient deze test nog eens te worden uitgevoerd met voldoende gebruikers aangesloten op de wijkrouter. Maar voor nu was de opdrachtgever tevreden gesteld.

De wijkrouter moet in staat zijn een abonnee aan en af te sluiten en een Globally routable IP adres toe te kennen aan een abonnee

Om deze test te kunnen voltooien dient de Lijbrandt Telecom Internet Administratie aanwezig te zijn in de test opstelling. Dit heb ik niet kunnen doen omdat dit enorm veel arbeid vergt. Ik heb een kopie gemaakt van de database van de Lijbrandt Telecom Internet Administratie en deze gebruikt om wijzigingen in tabellen te simuleren. Doordat de DatabaseConnector reageert op wijzigingen in de tabellen kon ik het aan en afsluiten van een abonnee simuleren. Toen ik dit deed ging alles volgens de manier waarop dit zou moeten gaan. Hierdoor was de opdrachtgever tevreden met het behaalde resultaat.

De wijkrouter moet betrouwbaar en niet storinggevoelig zijn.

Om deze test uit te voeren heb ik de testopstelling gedurende de hele projectperiode in stand gehouden en is de wijkrouter nooit uit gegaan. Er dient nog een test te komen met meerdere actieve aansluitingen op de wijkrouter maar tot zo ver is aangetoond dat de wijkrouter stabiel genoeg is om in ieder geval 34 dagen zonder problemen te functioneren. De opdrachtgever was hiermee tevreden gesteld.

9. Evaluatie

9.1 Proces evaluatie

Het werken bij Lijbrandt Telecom vond ik zelf een hele leuke ervaring. Het project wat ik deed bij Lijbrandt Telecom was echter een erg lastig project. Het beschrijven van het project in de vorm van een afstudeerverslag was dan ook een grote uitdaging. Daarnaast was het programmeren van de wijkrouter software een enorme taak om af te krijgen.

Ik werd hierbij gelukkig goed geholpen door mensen van Lijbrandt Telecom. Naast de begeleider was er Joost Vermeer die enorm veel ervaring heeft met PERL. Ook kon ik op deze manier achter oplossingen komen waar ik in eerste instantie misschien niet aan gedacht zou hebben.

In het begin van het project was het vooral lastig om af te stemmen hoe het bedrijf dingen zag en hoe ik daarop de projectmethode kon toepassen. Over de aanpak van het project en de methode die gehanteerd werd heerste onduidelijkheid. De opdrachtgever wilde volgens een bepaalde werkwijze werken en die conflicteerde met de werkwijze van Extreme Programming die ik wilde gebruiken.

Een goed voorbeeld hiervan was de planning, die in Extreme Programming niet erg gedetailleerd hoeft te zijn omdat men er vanuit gaat dat te veel anticiperen vooraf niet handig is. Hier was de opdrachtgever het niet helemaal mee eens, dus hebben we hier goed over gesproken. Daarna zijn we tot overeenstemming gekomen voor het gehele traject, en wordt de methodiek van Extreme Programming toegepast. Ik begreep de bezorgdheid van de opdrachtgever uiteraard. Ik heb hem daarna uitgelegd dat we een middenweg zoeken tussen beide werkwijzen. Hierover heb ik min of meer onderhandeld tot dat we ons allebei konden vinden in de werkwijze.

Aangezien de producten die extreme programming oplevert niet bekend waren bij de opdrachtgever heb ik deze toegelicht in het plan van aanpak.

Als laatste heb ik overlegd met de opdrachtgever en gevraagd of alles wat hij vond dat er in dit project moest zitten ook daadwerkelijk aan bod kwam en of hij hier mee tevreden was. Hij was tevreden en heeft daarna zijn handtekening onder het plan van aanpak gezet.

Met het ontwerp heb ik vooral zelf veel kunnen beslissen. Er waren slechts een aantal zaken waarover ik van mening verschilde met de opdrachtgever. Ik vond het nodig om het 3-Tier model te gebruiken om de in dit verslag beschreven redenen. De opdrachtgever had als idee om hiervoor een beveiligde verbinding op te zetten met de database en alles door middel van één programma te laten verlopen. Hierover hebben we uiteindelijk overeenstemming bereikt en is het 3-Tier model als meest geschikt bevonden.

De implementatiefase was lastiger dan alle andere fasen, omdat er veel geprogrammeerd moest worden. Dit had tot gevolg dat ik veel moest leren van de programmeertaal PERL. Deze taal werd al heel lang gebruikt bij Lijbrandt Telecom en was ook de taal die ik moest gebruiken. Ik had wel geluk dat er een PERL expert aanwezig was binnen Lijbrandt Telecom die mij enorm veel kon uitleggen. Ik had al wel ervaring met verschillende andere programmeertalen echter was de PERL 5 programmeertaal toch weer anders op een aantal fronten.

In het begin was ik nog niet in staat om echt mooie constructies te programmeren in PERL maar met behulp van Joost Vermeer en mijn parate kennis wat programmeren betreft kon ik al vrij snel programmeren in PERL.

Bij aanvang van mijn afstudeer periode waren een groot aantal benodigdheden aanwezig. Wanneer ik hardware of software nodig had, kon ik dit meteen melden waarna het besteld werd. Verder was het nodig om door middel van veel afspraken met de opdrachtgever achter de eisen van het systeem te komen. Hiervoor werd altijd tijd vrij gemaakt en ik heb nooit problemen gehad om mensen te spreken te krijgen.

Uiteindelijk kijk ik terug op een hele leuke tijd bij Lijbrandt Telecom en ook een zeer leerzame tijd. Naast het project waar ik mee bezig ben geweest heb ik door mijn tijd bij Lijbrandt Telecom een gevoel gekregen over de omvang van de werkzaamheden die gemoeid zijn met het leveren van diensten aan abonnees.

9.2 Product evaluatie

Ik heb mij tijdens dit project vooral gericht op het ontwikkelen van software voor de wijkrouter. De bedoeling was om een compleet systeem op te leveren waar software op draait die gegevens uit de administratieve databases van Lijbrandt Telecom ,toepast op de wijkrouter. Dit is redelijk goed gelukt denk ik, echter ben ik nog niet helemaal tevreden. Er zijn een aantal vlakken waar ik meer aandacht aan zou willen besteden. Hiertoe heb ik niet de kans gehad door de gelimiteerde tijd die ik voor dit project had. Wat vooral meer aandacht verdient is de omgeving waarin de software draait. Het is voor de performance van de gehele wijkrouter enorm belangrijk om te testen met verschillende soorten software. Ik denk onder andere aan de gebruikte DHCP daemon en software om download en upload snelheden te beperken. Al deze onderdelen hebben te weinig aandacht gehad in deze project periode. Overigens denk ik wel dat ik een goed eind op weg ben geweest om uiteindelijk een systeem op te leveren dat in een productie omgeving goed kan functioneren. Ook aan de tests schort het denk ik nog aan het één en ander. Bijvoorbeeld een white-box test zou niet misstaan. Door met grenswaardes te testen is het veel makkelijker om de exacte specificaties weer te geven aan mensen die beslissingen moeten nemen over de implementatie van de wijkrouter. Op dit moment zijn er naast de acceptatietests en wat functionele tests(blackbox tests) nog weinig concrete testgegevens.

9.2.1. Iteratie: DatabaseConnector

De DatabaseConnector was vooral een procedureel programma dat dan ook als zodanig geschreven is in PERL. Dit heeft als gevolg dat hiervoor weinig ontwerp is terug te vinden in dit document. Omdat UML als modelleertechniek is gebruikt, heb ik om de DatabaseConnector bespreekbaar te maken gebruik gemaakt van een PSD. Het had voor mij simpelweg geen zin om hiervoor object georiënteerde oplossingen te verzinnen aangezien dit programma en zijn functie er zich totaal niet voor leent. De DatabaseConnector is overigens als eerste direct in gebruik genomen in het bedrijf. De software is op zichzelf compleet en kan ook worden toegepast in een omgeving waar de oude wijkrouter nog actief is. Met als grootste voordeel dat wijzigingen uit de Lijbrandt Telecom Internet Administratie nu real-time worden verwerkt in plaats van het polling systeem dat de oude wijkrouter hanteerde. Ook omdat deze software volledig opnieuw is opgebouwd is nu duidelijk hoe deze werkt en kunnen hier uitbreidingen voor worden geschreven. Bijvoorbeeld wanneer

Lijbrandt Telecom andere diensten gaat aanbieden. Dan is het nodig om meer configuratie items weer te geven vanuit de DatabaseConnector. Het is nu goed mogelijk om de software uit te breiden. Mede doordat de broncode voorzien is van commentaar over wat er precies gebeurd. Na de implementatie is deze iteratie opgeleverd als een small release.

9.2.2. iteratie: Linux router applicatie

De Linux router applicatie vergde het meeste tijd om te programmeren. Mede doordat er gebruik werd gemaakt van een object georiënteerde aanpak. Hierdoor zijn er nu ook UML modellen terug te vinden in het ontwerp. Naast dat het een enorm werk was, zijn ook niet alle functies van de wijkrouter die in eerste instantie bedacht waren terug te vinden in de Linux router applicatie. Wel zijn alle functies die in de ‘Must have’ clausule terug te vinden zijn van het releaseplan, geïmplementeerd. Uiteindelijk is ook deze iteratie opgeleverd als een small release.

Als laatste heb ik te weinig aandacht besteed aan het status scherm. Op dit moment bestaat dit uit een aantal meldingen op de Linux console die niet erg duidelijk zijn. Ik heb mij hier vanwege de tijdsdruk niet genoeg mee bezig kunnen houden.

9.2.3. Conclusie

Het uiteindelijke product van dit project is de wijkrouter software. Ik heb op de meeste onderdelen voldoende werk kunnen verrichten om zo een functioneel systeem op te leveren. Het is echter nog niet af, er moeten nog veel tests gedaan worden met de software en ook de software die gebruikt wordt om de instellingen op de wijkrouter toe te passen verdient meer aandacht. Ik denk ook dat ik met meer tijd voor dit project bepaalde dingen beter had kunnen uitwerken.

De ontwikkeling van een linux router applicatie

APPENDIX

Appendix

A. Bronnenlijst

Websites:

De officiële site van de projectmethode Extreme Programming:
http://www.extremeprogramming.org

Een informatie site voor de projectmethode DSDM. Hieruit kon ik de benodigdheden voor de MoSCoW methode halen en het plannen d.m.v. timeboxing:
http://www.dsdm.nl/nl/dsdm/timeboxing.asp

De documentatie van het PostgreSQL RDBMS:
http://www.postgresql.org/docs/8.0/static/index.html

Voor informatie over het 3-tier model heb ik een aantal websites geraadpleegd deze zijn:
http://www.corba.ch/e/3tier.html
http://www.thedutchrepublic.com/profile/technology.html

Voor het opzetten van de IPIP tunnel heb ik gebruik gemaakt van de volgende website:
http://www.wlug.org.nz/IPIP

Voor informatie over PERL modules heb ik de volgende website gebruikt:
http://www.webreference.com/programming/perl/modules/

Voor informatie over traffic shaping heb ik de volgende website gebruikt:
http://www.knowplace.org/shaper/examples.html

Naast deze websites heb ik gebruik gemaakt van de volgende boeken:

- *Perl Cookbook*
  - Tom Christiansen & Nathan Torkington, O'REILLY, ISBN 0-596-00313-7
- *Teach Yourself Perl*
  - Laura Lemay, SAMS, ISBN 0-672-31305-7
- *De UML toolkit*
  - Hans-Erik Eriksson & Magnus Penker, Academic Service, ISBN 90-395-1015-6
- *Programming Perl*
  - Larry Wall, Tom Christiansen & Jon Orwant, O'REILLY, ISBN 0-596-00027-8
- *A Guide to THE SQL STANDARD*
  - C.J. Date & Hugh Darwen, Addison Wesley, ISBN 0-201-96426-0
- *PostgreSQL*
  - Korry Douglas & Susan Douglas, Developer's Library, ISBN 0-7357-1257-3

B. Broncode: DatabaseConnector

```
#!/usr/bin/perl
use strict;

use DBI;
use IO::Select;
use IO::Handle;

# Verbinding via DBI met PostgreSQL
my $db = DBI->connect("dbi:Pg:dbname=ltia", undef, undef, {RaiseError
=> 1})
  || die "Error connecting to the database: $DBI::errstr";

# Ophalen trunk argument
my @trunk_tags = @ARGV;

# Geef error waarneer trunk argument niet is opgegeven
@trunk_tags || die "No trunk argument";

# Array voor het bijhouden van de trunk_ids
my @trunk_ids;

# Ophalen trunk_id en algemene configuratie
for my $trunk_tag (@trunk_tags) {
  my ($trunk_id, $trunk_config) = $db->selectrow_array("
    SELECT trunk_id, inet_config
    FROM trunk
    WHERE tag = ?", {}, $trunk_tag);

  # Geef error waarneer trunk_id niet gevonden is
  $trunk_id || die "Trunk $trunk_tag not found";

  # Voeg het trunk_id toe aan @trunk_ids
  push @trunk_ids, $trunk_id;

  # Print de algemene trunk configuratie regeltjes
  print "CONFIG trunk$trunk_id.tag: $trunk_tag\n";
  foreach my $line (split /\n/, $trunk_config) {
    print "CONFIG trunk$trunk_id.$line\n";
  }
}

# Deze query haalt de instellingen per aansluiting op uit de database
aan de hand van de trunk_ids en de last_update
my $query_links = $db->prepare("
  SELECT
    trunk.trunk_id,
    link.link_id,
    link.default_vlanid AS vlanid,
    CASE
      WHEN network_type.only_layer2 THEN FALSE
      WHEN ip_address.address IS NULL THEN FALSE
      WHEN network.active THEN TRUE
      ELSE FALSE
    END AS active,
    network.block_smtp,
```

```
    network_type.config,
    ip_address.address AS ip_address,
    network_update.timestamp AS last_update
  FROM trunk
    JOIN link USING (trunk_id)
    LEFT JOIN network USING (link_id)
    LEFT JOIN network_type USING (network_type_id)
    LEFT JOIN ip_address USING (network_id)
    LEFT JOIN network_update USING (link_id)
  WHERE (trunk.trunk_id = ?" . " OR trunk.trunk_id = ? " x
(@trunk_ids-1) . ")
    AND (? OR network_update.timestamp > ?)
  ORDER BY trunk.trunk_id, link.link_id, ip_address.address");

# Deze variable geeft aan de we updates aan het versturen zijn word
gezet aan het eind van de "eerste ronde"
my $updateing = 0;

# Deze variable bevat de timestamp vanaf waarneer voor het laatst is
geupdate
# Initieel is deze een datum in het verleden.
my $last_update = '1980-01-01 00:00:00';

# Maak een select object aan om te controleren of er data is binnen
gekomen op de database verbinding
my $select = IO::Select->new(\*STDIN, $db->func('getfd'));

# Deze variable wordt gezet waarneer er een notificatie is
binnengekomen van de database
my $notified;

# Registreer de database connectie voor asynchrone notificatie
# Een trigger in de database zal bij wijziging van een van de
tabellen van
# belang voor de wijkrouter een NOTIFY doen op "network_update"
$db->do("LISTEN network_update");

while(1) {
  # Deze variable houd bij welke aansluitingen erzijn of zijn
aangepast
  my @links;

  # Controleer of er notificaties staan te wachten tenzij dat al
bekend is (zie einde loop).
  $notified = defined $db->func('pg_notifies') unless defined
$notified;

  # Als we update aan het verzenden zijn controlleer dan of we een
  # notificatie van de database hebben binnen gekregen
  if((!$updateing) || $notified) {
    # Voordat de queries worden uitgevoerd halen we de notify queue
leeg
    while($db->func('pg_notifies')) {}
    # Clear ook de vlag die aangeeft dat we genotified zijn
    undef $notified;

    # Voor de query_links query uit om alle instellingen per
aansluiting uit de database te halen
    $query_links->execute(@trunk_ids, $updateing ? 'f' : 't',
$last_update);
```

```
    # Haal de eerste row op, de rows daarna worden later in de loop
opgehaald
    # tijdens het doorgeven van de ip addressen
    my $row = $query_links->fetchrow_hashref();

    while($row) {
      my $link_id = $row->{link_id};

      # Update de last_update variable als deze aansluiting later
gewijzigd is
      # dan de datum in last_update.
      # De twee datums worden met behulp van de gt operator
vergeleken. Dit kan
      # omdat de datums strings zijn in ISO 8601 formaat en dus
asciibetical
      # op de goede volgorde zijn.
      $last_update = $row->{last_update}
        if defined $row->{last_update}
          && (!defined $row->{last_update} || $row->{last_update} gt
$last_update);

      # Print per aansluiting de instellingen
      print "CONFIG link$link_id.trunk_id: trunk", $row->{trunk_id},
"\n";
      print "CONFIG link$link_id.vlanid: ", $row->{vlanid}, "\n";
      print "CONFIG link$link_id.active: ", ( $row->{active} ? "1" :
"0" ), "\n";
      if($row->{active}) {
        print "CONFIG link$link_id.block_smtp: ", ( $row-
>{block_smtp} ? "1" : "0" ), "\n";
        foreach my $line (split /\n/, $row->{config}) {
          print "CONFIG link$link_id.$line\n";
        }

        # geef de ip addressen door
        print "CONFIG link$link_id.ip_addresses:";
        do {
          print " ", $row->{ip_address};
          $row = $query_links->fetchrow_hashref();
        } while ($row && $link_id eq $row->{link_id});
        print "\n";
      } else {
        # Haal al vast de volgende row op, normaal gebeurt dat bij
het ophalen van de ip addressen
        do {
          $row = $query_links->fetchrow_hashref();
        } while ($row && $link_id eq $row->{link_id});
      }
      push @links, "link$link_id";
    }

    if(@links) {
      print "NOTIFY ", join(' ', @links), "\n" || exit 0;
      STDOUT->flush();
    }
  }
  $updateing = 1;
  # Controleer voor notificaties die zijn binnen gekomen tijdens het
verwerken van de queries
  if( $db->func('pg_notifies') ) {
    $notified = 1;
```

```
  } else {
    # Wacht via het eerder gemaakte IO::Select object voor het binnen
komen van notificaties van de database
    $select->can_read();
  }
}
```

C. Broncode: Linux Router Applicatie

Klasse: Routerd

```
#!/usr/bin/perl

use strict;
use LTIAConnection;
use Queue;
use Link;

# Inladen configuratie file uit /etc/ltia/routerd
{package Settings; do "/etc/ltia/routerd"}

# Variabelen over halen uit configuratie file of standaart waarden
gebruiken
# waar mogelijk
my $configcmd = $Settings::CONFIG or die "no CONFIG";
my %trunkcfg = %{$Settings::TRUNKS} or die "no TRUNKS";
my $update_interval = $Settings::UPDATE_INTERVAL || 3600;

# Configuratie ophalen vanaf database server
my $config = LTIAConnection->new($configcmd);

# Opzetten wachtrij voor het configureren van de aansluitingen
my $queue = Queue->new();

# Hash voor het opzoeken van de aansluiting objecten
my %links;

# Hash voor het opzoeken van trunk informatie
my %trunks;

# Mainloop; hierin worden de switches geconfigureerd
while(1) {

  # Wacht het binnenkomen van configuratie regels tot dat er weer een
  # aansluiting aan de beurt is om gecontroleert te worden
  my $line = $config->readline($queue->timetowait());

  # Als er een configuratie regel binnen gekomen is verwerk die dan
  if(defined $line) {
    if($line =~ m/^NOTIFY (.*)$/) {
      # Laat voor elke link waar melding van wordt gemaakt in de
NOTIFY regel
      # de configuratie (opnieuw) inladen en plaats de link in de
queue
      for my $link (split / /, $1) {
        # Controleer of deze link al bestaat in de links hash
        if(!exists $links{$link}) {
          my $trunk = $config->{"$link.trunk"};
          # Controleer of deze trunk al bestaat
          if(!exists $trunks{$trunk}) {
            # Zoek de tag op van de trunk
            my $trunk_tag = $config->{"$trunk.tag"};
            if(!defined $trunk_tag) {
```

```
              warn "No tag for $trunk";
              undef $trunks{$trunk};
              undef $links{$link};
              next;
            }
            # Zoek de interface op waaraan de trunk verbonden is in
de configuratie
            # file.
            my $interface = $trunkcfg{$trunk_tag}->{INTERFACE};
            if(!defined $interface) {
              warn "No INTERFACE for TRUNK $trunk_tag";
              undef $trunks{$trunk};
              undef $links{$link};
              next;
            }
            # Sla de trunk op in de trunks hash
            $trunks{$trunk} = {
              Interface => $interface,
            }
          }

          # Controleer of de trunk nu bestaat
          if(defined $trunks{$trunk}) {
            # Maak de link aan
            $links{$link} = Link->new($link, $trunks{$trunk},
$config);
          } else {
            # De trunk bestaat niet dus kunnen we ook deze link niet
aanmaken
            undef $links{$link};
          }
        }

        # Laat de link zijn configuratie inlezen
        $links{$link}->update($config);
        # Voeg de link toe aan de queue
        $queue->schedule($link);
      }
    } else {
      # Waarschuw voor een configuratie regel die niet begrepen wordt
      warn "Illegal configuration line: $line";
    }
  }

  # Als er een link beschikbaar is om te verwerken verwerk deze dan
  if(defined (my $link = $queue->get())) {
    $links{$link}->process();
  }
}
```

Klasse: Link

```
package Link;

use strict;

# Met new wordt een nieuw Link object temaakt
sub new {
  my ($package, $link, $trunkcfg, $config) = @_;

  # Haal de trunk op waar deze link bij hoort
  my $trunk = $config->{"$link.trunk"};
  # Haal het vlanid op waar deze link op geconfigureerd moet worden
  my $vlanid = $config->{"$link.vlanid"};
  # De interface naam is de trunk interface gevolgd door het vlanid
met
  # daartussen een punt
  my $interface = $trunkcfg->{Interface} . "." . $vlanid;

  # Maak de object referentie aan
  return bless {
    Trunk => $trunk,
    Link => $link,
    Vlanid => $vlanid,
    Interface => $interface,
    TempDir => '/var/tmp/routerd',
  };
}

# update wordt aangeroepen om de configuratie zoals opgehaald door
# LTIAConfigConnection over te zetten naar dit object
sub update {
  my ($self, $config) = @_;

  # Sla veel gebruikte attributen op in makkelijk toegankelijke
variabelen
  my $link = $self->{Link};
  my $trunk = $self->{Trunk};

  # Ontzet de ConfigComplete vlag om er voor te zorgen dat deze Link
niet
  # geprocess()t kan worden als er iets mis gaat
  $self->{ConfigComplete} = 0;

  # Kijk of deze link actief moet zijn
  my $active = $config->{"$link.active"};
  if(!defined $active) {
    warn "$link.active not defined";
    return undef;
  }

  # Als dit link actief moet zijn haal dan ook de rest van de
configuratie
  # over.
  if($active) {
    # Haal het network waarop deze Link moet worden geconfigureerd
over
    my $network = $config->{"$trunk.network"};
    if(!defined $network) {
```

```
      warn "$trunk.network not defined";
      return undef;
    }
    # Haal het netmask van de netwerk over
    my $netmask = $config->{"$trunk.netmask"};
    if(!defined $netmask) {
      warn "$trunk.netmask not defined";
      return undef;
    }
    # Kijk welk IP address wij moeten gebruiken naar deze aansluiting
    my $gatewayip = $config->{"$trunk.gateway"};
    if(!defined $gatewayip) {
      warn "$trunk.gateway not defined";
      return undef;
    }
    # Sla de IP addressen ten behouve van deze link op in $self-
>{IpAddresses}
    my $ip_addresses = $config->{"$link.ip_addresses"};
    if(!defined $ip_addresses) {
      warn "$link.ip_addresses not defined";
      return undef;
    }
    # De array voor het bijhouden van de IP addressen
    my @ips;
    # Vul de array met de door spaties gescheiden ip addressen in de
    # configuratie
    for my $ip (split / /, $ip_addresses) {
      push @ips, $ip;
    }

    # Haal de bandtbreedte instellingen op
    my $rate_down = int $config->{"$link.rate_down"};
    my $rate_up = int $config->{"$link.rate_up"};

    # Controleer of SMTP geblokeerd moet worden
    my $block_smtp = $config->{"$link.block_smtp"};

    # Haal de DNS servers op die met DHCP moeten worden doorgegeven
    my $dns_servers = $config->{"$trunk.dns_servers"};
    if(!defined $dns_servers) {
      warn "$trunk.dns_servers not defined";
      return undef;
    }
    # Haal de domein naam op die met dhcp moet worden doorgegeven
    my $domain_name = $config->{"$trunk.domain_name"};
    if(!defined $domain_name) {
      warn "$trunk.domain_name not defined";
      return undef;
    }

    # Sla de instellingen op als attributen van Link
    $self->{Network} = $network;
    $self->{Netmask} = $netmask;
    $self->{GatewayIp} = $gatewayip;
    $self->{IpAddresses} = \@ips;
    $self->{RateDown} = $rate_down;
    $self->{RateUp} = $rate_up;
    $self->{BlockSmtp} = $block_smtp;
    $self->{DNSServers} = $dns_servers;
    $self->{DomainName} = $domain_name;
  }
```

```
  $self->{Active} = $active;

  # Geeft aan dit dit Link object weer kan worden geprocess()t
  $self->{ConfigComplete} = 1;

  # Geef terug dat alles goed is gegaan
  return 1;
}

# process zorgt ervoor dat de eigenschappen van de aansluiting worden
overgezet
# naar de werkelijke instellingen van de interface.
sub process {
  my ($self) = @_;

  # Sla veel gebruikte attributen op in makkelijk toegankelijke
variabelen
  my $link = $self->{Link};
  my $iface = $self->{Interface};
  my $gatewayip = $self->{GatewayIp};

  # Als de configuratie niet in orde is stoppen we hier
  $self->{ConfigComplete} or return undef;

  # Als de link op dit moment al goed is ingesteld hoeven we niets te
doen
  my $msg;
  unless($msg = $self->_check()) {
    print "$link: $iface already configured correctly\n";
    return 1;
  }

  # Configureer de link
  print "$link: Configurering $iface ($msg)\n";
  $self->_configure();

  # Controleer of de link nu wel goed is ingesteld
  if(my $msg = $self->_check()) {
    warn "Configuration of $iface failed: $msg";
    return 0;
  }

  # Als alles goed is geven we 1 terug
  return 1;
}

# _check controleert of de instellingen van de link overeenkomen met
de
# gewenste instellingen volgens de attributen van het Link object
sub _check {
  my ($self) = @_;

  # Sla veel gebruikte attributen op in makkelijk toegangkelijke
variabelen
  my $link = $self->{Link};
  my $iface = $self->{Interface};
  my $gw = $self->{GatewayIp};
  my $tempdir = $self->{TempDir};

  # Controleer of de verbinding actief moet zijn
```

```
  if($self->{Active}) {
    # Kijk of de vlan interface bestaat
    if(!-e "/proc/net/vlan/$iface") {
      return "Link active, but vlan interface does not exists";
    }

    # Kijk of de interface "UP" is.
    if(`ip link show dev $iface` !~ m/UP/) {
      return "Interface down";
    }

    # Maak een hash met als keys de IP addressen voor deze verbinding
om
    # gemakkelijk te controleren dat een IP address gevonden in de
routing
    # tabel hoort bij deze interface
    my %ips;
    for my $ip (@{$self->{IpAddresses}}) {
      $ips{$ip} = 1;
    }
    # Haal de routing tabel op door het commando "ip route show" te
runnen.
    open ROUTE, "-|", "ip route show";
    while(<ROUTE>) {
      # Als de regel overeenkomt met "<IP-address> dev <Interface>"
moeten we
      # deze regel controleren
      next unless m/^([0-9.]+) dev $iface/;
      # Controleer of het IP adres uit deze regel inderdaad hoort bij
deze
      # interface
      if(!$ips{$1}) {
        return "Extra route configured for this interface: $1";
      } else {
        # Verwijder dit IP address uit de %ips hash zodat we later
kunnen zien
        # of er voor elk IP een routing entry is
        delete $ips{$1};
      }
    }

    # Als er een IP-address is overgebleven in de %ips hash betekend
dit dat
    # er geen routing entry voor dat IP-address is
    if(my ($ip) = keys %ips) {
      return "Route for $ip not configured for this interface";
    }

    # Haal de rate instellingen op uit de attributen
    my $rate_down = $self->{RateDown};
    my $rate_up = $self->{RateUp};
    # Controlleer of de bandbreedte beperkt moet worden
    if(defined $rate_down && defined $rate_up
        && $rate_down > 0 && $rate_up > 0) {
      # Haal de huidige qdisc instellingen op van deze interface
      my $qdisc = `tc qdisc show dev $iface`;
      # Controlleer of deze overeenkomen met een TBF qdisc
      my ($tbf_rate_down, $tbf_multiplier) = $qdisc =~
        m/^qdisc tbf [0-9a-f]+: rate ([0-9]+)([KM]?)bit /;
      # Als er geen TBF qdisc is geconfigureerd is het in elk geval
fout
```

```
      if(!defined $tbf_rate_down) {
        return "Qdisc configured incorrectly: $qdisc";
      }
      # qdisc kan zijn output geven in bit, Kbit of Mbit. Hier wordt
de
      # vermenigvuldigingsfactor toegepast als het Kbit of Mbit was.
      $tbf_rate_down *= 1000 if $tbf_multiplier eq "K";
      $tbf_rate_down *= 1000000 if $tbf_multiplier eq "M";
      # Controleer dat de ingestelde bandbreedte niet meer dan 1%
afwijkt
      # van de gewenste bandbreedte. (Dit om problemen met
      # afrondingsverschillen te voorkomen)
      if(abs($tbf_rate_down / $rate_down - 1) > 0.01) {
        return "Current rate_down is $tbf_rate_down, should be
$rate_down";
      }
    } else {
      # De bandbreedte moet niet beperkt worden er mogen dus geen
qdiscs
      # aan deze interface hangen
      if(`tc qdisc show dev $iface` ne "") {
        return "No rate limiting but qdisc set";
      }
    }

    # Kijk of SMTP verkeer geblokkeerd wordt door de block_smtp chain
van de
    # packet filter te analyseren
    my $block_smtp = 0;
    # Haal de chain op door iptables aan te roepen
    open IPTABLES, "-|",
      "iptables -t filter -L block_smtp -n -v -x --line-numbers";
    # Doorloop de output van iptables
    while(<IPTABLES>) {
      # Splits de velden van elke regel uit in de array @fields
      my @fields = split /\s+/;
      # Als deze regel niet over deze interface gaat ga dan naar de
volgende
      # regel
      next unless $fields[6] eq $iface;
      # Als deze regel wel over deze interface gaat dan staat SMTP
blocking aan
      $block_smtp = 1;
    }
    close IPTABLES;
    # Controlleer of het de bedoeling is dat SMTP blocking wel of
niet aan
    # staat
    if($self->{BlockSmtp}) {
      $block_smtp or return "SMTP blocking enabled, but no iptables
rule";
    } else {
      !$block_smtp or return "SMTP blocking disabled, but iptables
rule";
    }

    # Kijk of de DHCP daemon configuratie file bestaat
    if(!-e "$tempdir/$link.dhcpd.conf") {
      return "No dhcpd configuration file";
    }
    # Kijk of de leases file voor de DHCP daemon bestaat
```

```
    if(!-e "$tempdir/$link.dhcpd.leases") {
      return "No dhcpd leases file";
    }
    # Kijk of de PID file voor de DHCP daemon besaat
    if(!-e "$tempdir/$link.dhcpd.pid") {
      return "No dhcpd pid file";
    } else {
      # Haal het process ID uit de PID file
      open PID, "$tempdir/$link.dhcpd.pid";
      my $pid = int(<PID>);
      close PID;
      # Kijk of het een legaal PID is van een op dit moment
      # draaient process
      if($pid < 0 || !-e "/proc/$pid") {
        return "dhcpd not running";
      }
    }
    # Controleer of de huidige dhcpd.conf overeenkomt met de
    # gewenste DHCP configuratie
    open DHCPDCONFIG, "<", "$tempdir/$link.dhcpd.conf";
    if($self->_dhcpd_config() ne join('', <DHCPDCONFIG>)) {
      close DHCPDCONFIG;
      return "dhcpd config file not up to date";
    }
    close DHCPDCONFIG;
  } else {
    # Als de link niet actief moet zijn maar de vlan interface niet
bestaat
    # klopt het niet
    if(-e "/proc/net/vlan/$iface") {
      return "Link inactive, but vlan interface exists";
    }

    # Als er nog een pid file is van de DHCP daemon betekend dit dat
deze
    # nog niet is afgesloten
    if(-e "$tempdir/$link.dhcpd.pid") {
      return "Link inactive, but dhcpd still running";
    }
  }

  # Geef een lege string terug als teken dat alles in orde is
  return "";
}

# _configure wordt aangeroepen om de link in te stellen
sub _configure {
  my ($self) = @_;

  # Sla veel gebruikte attributen op in makkelijk toegankelijke
variabelen
  my $link = $self->{Link};
  my $iface = $self->{Interface};
  my $gw = $self->{GatewayIp};
  my $tempdir = $self->{TempDir};

  # Controlleer of een een PID file is voor de DHCP daemon
  if(-e "$tempdir/$link.dhcpd.pid") {
    # Lees het process id uit de PID file
    open PID, "<", "$tempdir/$link.dhcpd.pid";
    my $pid = int(<PID>);
```

```
    close PID;
    # Controleer of het een legaal pid is
    if($pid > 0) {
      # Kill de nog draaiende DHCP server
      kill 9, $pid;
    } else {
      warn "Illegal DHCPD pid";
    }
    # Verwijder de PID file
    unlink "$tempdir/$link.dhcpd.pid";
  }

  # Controleer of de VLAN interface nog bestaat
  if(-e "/proc/net/vlan/$iface") {
    # Zet de interface "DOWN" anders kunnen de VLAN interface niet
verwijderen
    system "ip link set dev $iface down";
    # Verwijder de VLAN interface
    system "vconfig rem $iface";
  }

  # Verwijder nog bestaande block_smtp regels met betrekking tot deze
interface
  open IPTABLES, "-|",
    "iptables -t filter -L block_smtp -n -v -x --line-numbers";
  # Doorloop de output van het iptables commando
  while(<IPTABLES>) {
    # Split de velden op in de array @fields
    my @fields = split /\s+/;
    # Controleer of deze regel bij deze interface hoort
    next unless $fields[6] eq $iface;
    # Verwijder deze regel dan
    system "iptables -t filter -D block_smtp " . $fields[0];
  }
  close IPTABLES;

  # Als de verbinding actief moet zijn kunnen we de verbinding nu
weer gaan
  # opzetten
  if($self->{Active}) {
    # $iface_s bevat de naam van deze interface met in plaats van een
punt een
    # spatie
    my $iface_s = $iface;
    $iface_s =~ s/\./ /;
    # Voeg de vlan interface toe
    system "vconfig add $iface_s";
    # Voeg ons (gateway) ip address toe aan de interface
    system "ip addr add dev $iface local $gw";

    # Haal de bandbreedte beperkins opties op
    my $rate_down = $self->{RateDown};
    my $rate_up = $self->{RateUp};
    # Controlleer of de bandbreedte beperkt dient te worden
    if(defined $rate_down && defined $rate_up
        && $rate_down > 0 && $rate_up > 0) {
      # Als burst wordt staandaart een halve seconde aan bandbreedte
gegeven
      my $burst_down = $rate_down / 4;
      my $burst_up = $rate_up / 4;
      # Voeg de TBF qdisc toe aan de interface
```

```
      system "tc qdisc add dev $iface root tbf latency 100ms "
        . "burst " . $burst_down . "b "
        . "rate " . int($rate_down / 1000) . "kbit";
    }

    # Voeg een block_smtp regel toe aan de packet filter waarneer dat
nodig is
    if($self->{BlockSmtp}) {
      system "iptables -t filter -A block_smtp -i $iface -j DROP";
    }

    # Activeer de interface
    system "ip link set dev $iface up";

    # Voeg routes toe voor elk toegewezen ip adres
    for my $ip (@{$self->{IpAddresses}}) {
      system "ip route add $ip dev $iface";
    }

    # Scrhijf een nieuwe DHCP daemon configuratie weg
    open DHCPDCONFIG, ">", "$tempdir/$link.dhcpd.conf";
    print DHCPDCONFIG $self->_dhcpd_config();
    close DHCPDCONFIG;
    # Zorg ervoor dat de leases file bestaat
    system "touch $tempdir/$link.dhcpd.leases";
    # Start de DHCP daemon
    system "dhcpd -q "
      . "-cf $tempdir/$link.dhcpd.conf "
      . "-lf $tempdir/$link.dhcpd.leases "
      . "-pf $tempdir/$link.dhcpd.pid "
      . "$iface";
    # Waarschuw als de DHCP daemon niet 0 terug geeft
    warn "DHCPD returned $?" if $?;
  }
}

# _dhcpd_config genereert een dhcp configuratie op basis van dit link
object
sub _dhcpd_config {
  my ($self) = @_;

  # Geeft de DHCP configuratie terug als string
  return
    "default-lease-time 300;\n" .
    "max-lease-time 300;\n" .
    "option broadcast-address 255.255.255.255;\n" .
    "option routers " . $self->{GatewayIp} .";\n" .
    "option domain-name-servers "
      . join(',', split(/ /, $self->{DNSServers})) . ";\n" .
    "option domain-name \"" . $self->{DomainName} . "\";\n" .
    "ddns-update-style none;\n" .
    "\n" .
    "subnet " . $self->{Network} . " netmask " . $self->{Netmask} .
"{\n" .
    join('', map {"  range $_ $_;\n"} @{$self->{IpAddresses}}) .
    "}\n";
}

1;
```

Klasse: LTIAConnection

```
package LTIAConnection;

use strict;

use IO::Handle;
use IO::Select;
use IPC::Open2;

# De methode new maakt een nieuw LTIAConfigConnector aan
sub new {
  my ($package, @cmd) = @_;

  my $self = {
    # Het commando wat uitgevoerd moet worden
    _command => [@cmd],

    # Deze boolean geeft aan of we verbonden zijn
    _connected => 0,

    # Dit is de filename van het bestand waarin de configuratie
gebackuped word.
    _backupfile => 'config-backup',
  };

  return bless $self, $package;
}

sub readline {
  my ($self, $timeout) = @_;

  # Door de huidige tijd bij timeout op te tellen geeft timeout nu
aan waarneer we klaar moeten zijn
  $timeout += time if defined $timeout && $timeout > 0;

  while(1) {
    # Kijk eerst in de input buffer of er nog regels staan te wachten
    if($self->{_inbuf} =~ m/\n/) {
      $self->{_inbuf} =~ s/^([^\n]*)\n//;
      my $line = $self->_parseline($1);
      if(defined $line) {
        return $line;
      } else {
        next;
      }
    }

    # Zolang de timeout niet verstreken is proberen we een regel te
lezen
    return undef if defined $timeout && time > $timeout;

    # Open de verbinding waarneer dat nog niet gebeurt is
    if(!$self->{_connected}) {
      # Controleer eerst of de laatste keer dat we verbinding hebben
gemaakt minimaal 5 s geleden is
      if(defined $self->{_last_connect}
          && $self->{_last_connect} + 5 > time) {
```

```
        my $timetogo = $timeout - time if defined $timeout;
        $timetogo = 1 if defined $timetogo && $timetogo < 1;
        if(!defined $timetogo || $timetogo > 5) {
          sleep 5;
        } elsif($timetogo > 1) {
          sleep $timetogo;
        } else {
          last;
        }
      }
      $self->{_last_connect} = time;
      $self->_connect();
    }

    if($self->{_connected}) {
      my $timetogo = $timeout - time if defined $timeout;
      $timetogo = 0 if defined $timetogo && $timetogo < 0;
      if($self->{_select}->can_read($timetogo)) {
        # Lees de binnen gekomen data
        my $r = sysread $self->{_in}, $self->{_inbuf}, 1024, length
$self->{_inbuf};
        if($r < 1) {
          $r == 0 || warn "Read from config stream failed: $!";
          $self->_disconnect();
          next;
        }
      }
    }
  };

  # Als er geen notificatie is binnengekomen lezen we de backup in
  if(!defined $self->{_notifications}) {
    warn "Reading backup";
    return $self->_readbackup();
  }

  return undef;
}

sub writeline {
  my ($self, @line);
  print {$self->{_out}} @line, "\n";
  $self->{_out}->flush();
}

sub _parseline {
  my ($self, $line) = @_;

  if(!defined $line || $line eq "") {
    return undef;
  } elsif($line =~ m/^CONFIG /) {
    if($line =~ m/^CONFIG ([a-z][a-z0-9_]*(?:\.[a-z][a-z0-9_]*)*):
(.*)$/) {
      $self->{$1} = $2;
    } else {
      warn "Illegal CONFIG line: $line";
    }
    return undef;
  } elsif ($line =~ m/^NOTIFY /) {
    if($line =~ m/^NOTIFY ([a-z][a-z0-9_]*(?: [a-z][a-z0-9_]*)*)$/s)
{
```

```
      for my $notification (split / /, $1) {
        $self->{_notifications}->{$notification} = 1;
      }
      $self->_writebackup();
      return $line;
    } else {
      warn "Illegal NOTIFY line: $line";
      return undef;
    }
  } else {
    return $line;
  }
}

sub _connect {
  my ($self) = @_;

  # Laat het commando opstarten en verbind de output met onze input
en de input van het command met onze output
  my $pid = open2(my $in, my $out, @{$self->{_command}});

  # Controleer of open2 gelukt is
  if(!$pid) {
    warn "open2 failed: $?";
    return undef;
  }

  $self->{_connected} = 1;
  $self->{_pid} = $pid;
  $self->{_in} = $in;
  $self->{_out} = $out;

  $self->{_inbuf} = '';
  $self->{_select} = IO::Select->new($in);
}

sub _disconnect {
  my ($self) = @_;

  # Sluit de input en output
  close $self->{In};
  close $self->{Out};

  # Wacht totdat het proces klaar is
  my $retval = waitpid($self->{Pid}, 0);

  warn join(' ', @{$self->{_command}}) . " returned: $?";

  $self->{_connected} = 0;
}

sub _writebackup {
  my ($self) = @_;
  my $backup = $self->{_backupfile};

  if(!open BACKUP, '>', "$backup~") {
    warn "Unable to write backup file: $!";
    return undef;
  }

  while(my ($key, $value) = each %$self) {
```

```
    next if $key =~ m/^_/;
    print BACKUP "CONFIG $key: $value\n";
  }

  print BACKUP "NOTIFY ", join(' ', keys %{$self->{_notifications}}),
"\n";

  if(!close BACKUP) {
    warn "Unable to write backup file $!";
    return undef;
  }

  rename "$backup~", $backup;
}

sub _readbackup {
  my ($self) = @_;
  my $backup = $self->{_backupfile};

  if(!open BACKUP, '<', $backup) {
    warn "Unable to open backup file: $!";
    return undef;
  }

  while(<BACKUP>) {
    chomp;
    my $line = $self->_parseline($_);
    if(defined $line && $line =~ m/^NOTIFY /) {
      close BACKUP;
      return $line;
    }
  }
  warn "Unable to read backup file";
  return undef;
}

1;
```

Klasse: Queue

```
package Queue;

use strict;

sub new {
  return bless {
    _future => {},
    _now => [],
  };
}

sub get {
  my ($self) = @_;
  my $timetowait = $self->timetowait();
  return undef if !defined $timetowait || $timetowait > 0;
  return shift @{$self->{'_now'}};
}

sub timetowait {
  my ($self) = @_;
  my $date = time;
  my $mindate;

  for my $item (sort keys %{$self->{'_future'}}) {
    my $itemdate = $self->{'_future'}->{$item};
    if($itemdate <= $date) {
      delete $self->{'_future'}->{$item};
      push @{$self->{'_now'}}, $item;
    } else {
      $mindate = $itemdate if !defined $mindate || $itemdate <
$mindate;
    }
  }

  return 0 if @{$self->{'_now'}};
  return undef if !defined $mindate;
  my $timetowait = $mindate - $date;
  return $timetowait;
}

sub schedule {
  my ($self, $item, $date) = @_;
  for (@{$self->{'_now'}}) {
    return if $item eq $_;
  }
  if(defined $date) {
    $date += time;
    return if
      defined $self->{'_future'}->{$item} &&
      $date < $self->{'_future'}->{$item};
    $self->{'_future'}->{$item} = $date;
  } else {
    if(defined $self->{'_future'}->{$item}) {
      delete $self->{'_future'}->{$item};
    }
    push @{$self->{'_now'}}, $item;
  }
```

```
}

1;
```