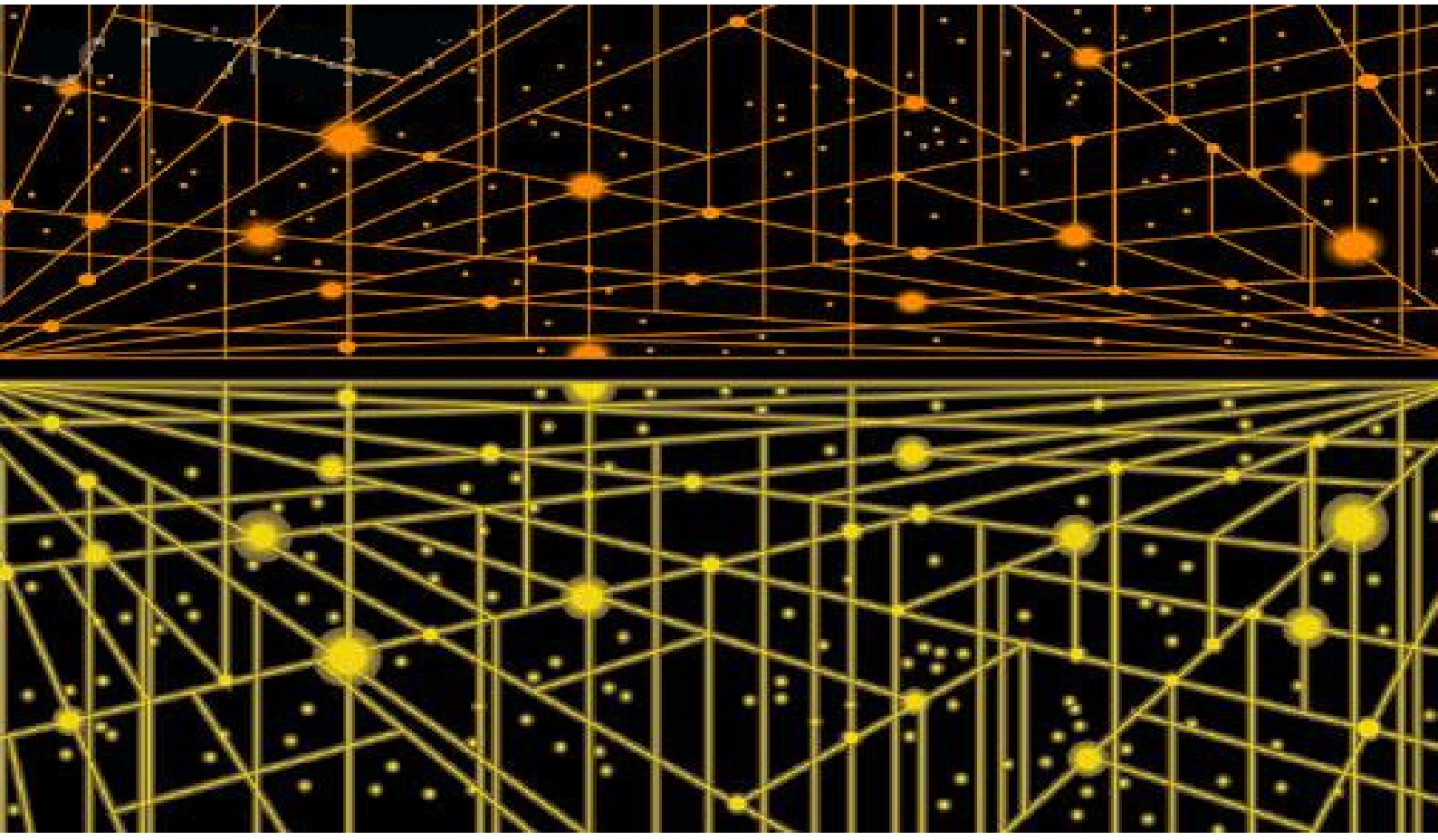
Ontwikkeling van een frame-work voor Distributed Computing.

Versie 2.1, 15 augustus 2005



Opleiding
UVA
1 jarige Master
Software engineering

Auteur
ing. Arjan Borst
a.borst@wireitup.nl
student nr. 0478199

Contact persoon
dr. Jeroen G. Snel
j.g.snel@amc.uva.nl
Radiologie

Stage bedrijf
AMC
Meibergdreef 9
NL-1105 AZ
Amsterdam

INHOUDSOPGAVE

1 Samenvatting

Voor het verwerken van medische beelden en het doorsturen daarvan naar bekijkstations of PACS (Picture Archive and Communication System) is het FAIPR (Framework for Automated Image Procssing and Routing) ontwikkeld.

Dankzij het succes van de FAIPR is er een groeiende vraag ontstaan naar het automatisch verwerken (analyseren) van medische beelden. Deze vraag uit zich op twee manieren:

1. Verhoging van de capaciteit voor het verwerken van de binnenkomende opdrachten.
2. Uitbreiding gewenst op de beschikbare analyses.

Met behulp van technieken en literatuur uit de software architectuur zal bepaald worden hoe het nieuwe framewerk zijn taak, het verwerken van medische beelden, gedistribueerd kan uitvoeren. In de architectuur bestaan diverse referentie architecturen. In dit onderzoek zal bekeken worden welke toegepast kan worden voor de realisatie van een gedistribueerde verwerking.

Voordat een architectuur bepaald wordt zal eerst worden ingegaan op het bepalen van de fasering, de huidige applicatie, requirements en legacy. De resultaten uit de vijf onderzoeksgebieden zijn aanbevelingen en conclusies. Op basis hiervan is een prototype ontwikkeld waardoor de resultaten uit de onderzoeken getoetst kunnen worden.

In het ontwikkelen van een prototype heeft met name het kwaliteitsattribuut ‘portability’ voor veel extra werk gezorgd. Het kwam er op neer dat veel van de source code toegespitst moest worden op de betreffende besturingssystemen. Een ander probleem waar dit project mee te maken kreeg was dat SOAP niet direct de referentie architectuur ondersteunde. Hiervoor moesten eerst verschillende aanpassingen worden gemaakt in de gegenereerde source code.

2 Inleiding

Dit project vindt plaats binnen de afdeling radiologie van het Academische Medische Centrum (AMC). Deze afdeling verricht onderzoeken met behulp van 'beeldvormende technieken'. Technieken die gebruik maken van röntgenstraling (CT-scan), geluidsgolven (echo) en magnetische resonantie (MRI-scan) worden hiervoor toegepast.

De afdeling telt meer dan 150 medewerkers: radiologen, onderzoekers, laboranten, fysici en technici, IT specialisten, managers en planners, secretarieel en administratief medewerkers, baliemedewerkers en archiefpersoneel.

Om bij een onderzoek de work-flow van de laboranten en radiologen gedeeltelijk te automatiseren is er door een onderzoeker het frame-work, genaamd FAIPR (Framework for Automated Image Processing and Routing) ontwikkeld. Hierdoor kan op autonome wijze analyses worden uitgevoerd op medische beelden. Voorheen gebeurde dit handmatig.

Het handmatige traject zag er als volgt uit: na het bedienen van de scanner moet de laborant de medische beelden versturen naar een bekijkstation, vervolgens versturen naar een bewerk station, daar de analyse starten, wachten tot deze klaar is (10 tot 20 minuten) om uiteindelijk de medische beelden te versturen naar de radioloog.

Na de introductie van FAIPR is dit traject voor de meest toegepaste analyses geautomatiseerd. Naast het feit dat laboranten niet zijn opgeleid voor dit soort werkzaamheden scheelt het automatiseren hiervan een hoop tijd. Hierdoor kunnen laboranten zich richten op hun basis taak, het besturen van de scanners en het zorg dragen voor de patiënten.

2.1 Ontwikkeling van een frame-work voor Distributed Computing

Dankzij het succes van de FAIPR is er een groeiende vraag ontstaan naar het automatisch verwerken (analyseren) van medische beelden. Deze vraag manifesteert zich op twee manieren: er is behoefte aan een verhoging van de capaciteit voor het verwerken van de binnenkomende opdrachten. Ook is er een uitbreiding gewenst op de beschikbare analyses.

Doordat het huidige systeem alleen lokale verwerking van de medische beelden ondersteunt kan het verwerken van een opdracht op drukkere dagen lang duren. Dit gebeurt namelijk serieel (één opdracht tegelijk) op basis van het first in first out principe. Hierdoor kan er een lange wachtrij ontstaan, het uitvoeren van een analyse kan soms wel een uur duren. Een dergelijke situatie kan voorkomen worden door het systeem uit te breiden met gedistribueerde verwerkingseenheden. Deze moeten simpel en snel uit te breiden zijn met nieuwe eenheden als dit nodig mocht zijn. Hierdoor kan de nieuwe applicatie verschillende opdrachten tegelijkertijd uitvoeren en wachtrijen tegengaan.

De verwerking van medische beelden gebeurt door middel van analyse programma's. De FAIPR ondersteunt er hiervan twee. Een voorbeeld is de MMBE (Match Mark Bone Elimination) analyse. Dit is een methode om automatisch de schedel te verwijderen uit scans van het hoofd met als doel de hersenen en de bloedvaten beter te visualiseren. Door de vraag naar ondersteuning van nieuwe analyses is het noodzakelijk dat het nieuwe systeem uitgebreid kan worden.

Naast het uitvoeren van een enkele analyse op pixel data is het bij specifieke verwerkingen nodig om een combinatie van analyses uit te voeren. De resultaten (vaak in de vorm van medische beelden) van een analyse kunnen namelijk dienen als invoer voor een andere analyse. Het moet dus mogelijk zijn om een work-flow te definiëren waarlangs de beelden geleid worden.

Ook moet het systeem voor een bredere doelgroep ingezet kunnen worden. Nieuwe services (die anders zijn dan analyses) moeten aan het systeem gekoppeld kunnen worden. Er kan hierbij gedacht worden aan verschillende routering services, zoals het verkrijgen van patiënt gegevens uit een database.

2.2 Introductie in de termen

In dit verslag wordt veelvuldig gebruikt gemaakt van medische vaktermen of termen die een bepaalde betekenis hebben binnen de afdeling waar ik werkzaam voor ben. De belangrijkste termen worden hieronder toegelicht om onduidelijkheden te voorkomen.

Beeldvormende technieken

Met behulp van beeldvormende technieken is het mogelijk om in het menselijk lichaam te kijken zonder hierbij het mes te hanteren. Er bestaan verschillende technieken voor het visualiseren van het lichaam. Binnen de afdeling radiologie van het AMC zijn de meest gebruikte technieken de computer tomografie (CT) en Magnetic Resonance Imaging (MRI).

Beeldvormende technieken worden gebruikt als hulpmiddel bij het stellen van de diagnose, als ondersteuning bij de behandeling, als middel om het effect van de behandeling na te gaan en om op langere termijn in de gaten te houden of de tumor niet terugkeert.

DICOM standaard

De standaard voor het vastleggen van het formaat van de medische beelden wordt DICOM genoemd. In deze standaard is er ruimte om naast de pixel data ook informatie over de patiënt, soort onderzoek en opname protocol vast te leggen.

Tevens is DICOM een communicatie standaard op basis waarvan de opname apparatuur en het archivering systeem de beelden uitwisselen. Dit gebeurt door middel van DICOM query's, stores en retrieves. Het verschil tussen deze methodes is dat met een query alleen patiënt gegevens wordt opgehaald. Een store archiveert daadwerkelijk de medische beelden en met een retrieve wordt ook de medische beelden verkregen.

Beeld analyses applicaties

Met behulp van analyse applicaties kan er sneller inzicht worden verkregen in de toestand van een patiënt. Hierdoor kunnen elementen als ruis weg worden gefilterd. De uitkomst zijn beelden waarop een specialist beter een diagnose kan maken.

PACS

PACS staat voor Picture Archiving and Communications System, wat wil zeggen dat het de opslag en distributie van beelden regelt. Alle digitale beelden die op de afdeling radiologie worden gemaakt, worden met behulp van het DICOM protocol hierin opgeslagen.

2.3 Probleemstelling

Dankzij zijn eigen succes wordt het FAIPR vervangen door een nieuw framework. De oorspronkelijke functionaliteit blijft behouden, dit betreft voornamelijk de automatische ondersteuning van de work-flow van de laboranten. De grote verandering is dat het nieuwe framework flexibeler en uitbreidbaar moet zijn, in zowel capaciteit als een bredere ondersteuning in analyses.

Het huidige systeem moet vervangen worden door een nieuwe. Voor het vervangen van dit systeem moet bekeken worden welke strategieën hiervoor beschikbaar zijn.

De huidige applicatie ondersteunt alleen locale verwerking van de beelden op de server en kan maar één analyse tegelijk in behandeling nemen. Door de groeiende vraag naar de automatische verwerking van medische beelden, wordt het belangrijk om meerdere analyses tegelijkertijd uit te kunnen voeren, waardoor FAIPR uitgebreid moet worden met gedistribueerde computer faciliteiten.

Daarnaast ondersteunt FAIPR twee analyse programma's, door de komst van nieuwe analyse methodes is het noodzaak om de nieuwe FAIPR het toevoegen van nieuwe analyses programma's op een relatief simpele manier te gaan ondersteunen.

2.4 Afbakening en aannames

Door de beperkte tijd van drie maanden zullen er diverse afbakeningen en aannames gemaakt moeten worden voor dit project. Door het stellen van deze afbakeningen kan gericht onderzoek plaatsvinden naar de onderzoeksgebieden die belangrijk worden gevonden. Hierdoor krijgen deze onderzoeken meer diepgang.

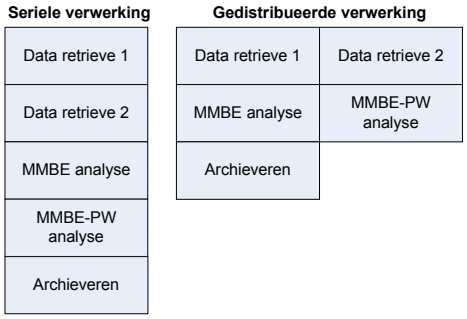
Onderzoeken binnen deze scriptie zijn niet gericht op volledigheid. Als er meerdere werkbare oplossingen worden genoemd in de literatuur wordt na een korte selectie een geschikte oplossing gekozen. Hierdoor wordt voorkomen dat alle beschikbare literatuur over het desbetreffend onderwerp moet worden doorgenomen voor het selecteren van de 'ultieme' oplossing. Er kan zodoende na twee tot vier oplossingen worden gekozen om uit deze selectie een goed werkbare te kiezen.

Legacy
Applicaties die legacy worden genoemd zijn moeilijk te vervangen doordat kennis in de vorm van zowel impliciete kennis als expliciete kennis ontbreekt. In de paragraaf legacy van het hoofdstuk uitvoering wordt hier dieper op ingegaan.

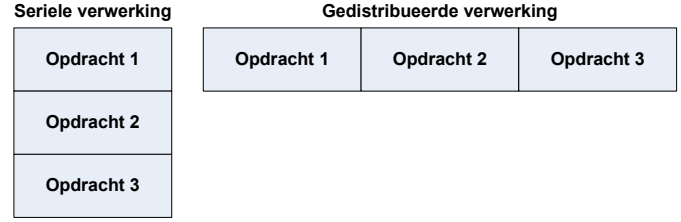
In dit project wordt aangenomen dat het huidige FAIPR een legacy applicatie is. De aanpak die voor dit project gehanteerd is, berust op een strategie die voortkomt uit een beschreven legacy aanpak. Er wordt stilgestaan bij welke risico' s het vervangen van een legacy systeem met zich mee brengt en wat de standaard legacy eigenschappen zijn. Zo kunnen maatregelen worden genomen om een succesvol project op te leveren en wordt er getracht te voorkomen dat het nieuwe systeem binnen afzienbare tijd een legacy systeem wordt.

Architectuur
Binnen dit project vindt het gedistribueerd verwerken van opdrachten op twee manieren plaats. Een opdracht bestaat uit meerdere deelopdrachten die vaak parallel uitgevoerd kunnen worden (zie figuur 1, deelopdracht niveau). De huidige manier van verwerken staat weergegeven aan de linkerkant van Figuur 1. Aan de rechterkant van Figuur 1 staat beschreven hoe in de nieuwe situatie de verwerking plaats zal gaan vinden. Goed is te zien dat de stappen 1-2 (Data retrieve) en 3-4 (MMBE - en MMBE-PW analyse) onafhankelijk van elkaar plaats kunnen vinden.

Links in figuur 2 staat weergegeven hoe de huidige FAIPR zijn binnenkomende opdrachten verwerkt. Dit gebeurt nu opdracht voor opdracht. Hierdoor kan het op drukke dagen lang duren voordat een opdracht verwerkt is. De nieuwe situatie ondersteunt ook het verwerken van meerdere opdrachten tegelijk (rechts in figuur 2).

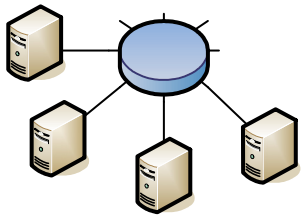


Figuur 1: deelopdracht niveau

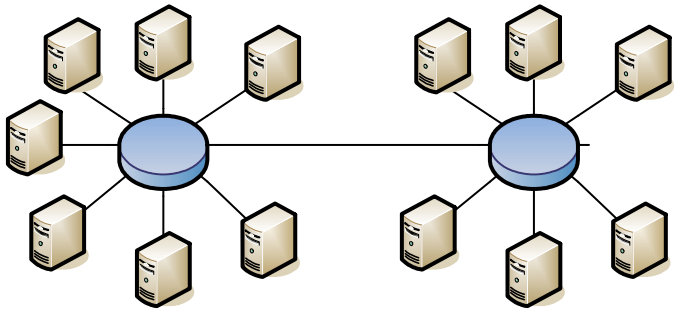


Figuur 2: opdracht niveau

Door de groeiende vraag naar het verwerken van medische beelden moet het nieuwe systeem schaalbaar opgezet worden. In figuur 3 staat een voorbeeld over hoe de nieuwe FAIPR in een begin situatie er uit zou kunnen zien, en heeft dan genoeg aan bijvoorbeeld vier servers. In de toekomst kan het systeem uitgerust gaan worden met meer servers, zoals te zien is in figuur 4.



Figuur 3, nieuwe FAIPR



Figuur 4, nieuwe FAIPR

Ontwikkeling prototype
Door middel van het geven van prioriteiten aan de requirements, zal bepaald worden welke eisen en wensen in het prototype worden geïmplementeerd. Met behulp van dit prototype wordt het functioneren van het systeem aangetoond.

2.5 Onderzoeksvraag

Dit onderzoek richt zich op twee hoofdonderzoeksgebieden. De eerste is legacy, hierin wordt onderzocht hoe de huidige FAIPR vervangen gaat worden door de nieuwe applicatie. Het tweede onderzoeksgebied is architectuur. Hier wordt de opzet en werking van het systeem bepaald.

De twee hoofdonderzoeksgebieden zijn afgeleid uit de onderzoeksvraag die in deze scriptie centraal staat, namelijk: “Hoe wordt het huidige framework vervangen door een nieuwe, met een gedistribueerde software architectuur? “

Voor het vervangen van de huidige FAIPR wordt er en legacy vervangingsstrategie toegepast. Binnen de software engineering is het vervangen van legacy software een veel voorkomende gebeurtenis. Hierdoor bestaan er verschillende vervangingsstrategieën.

Voordat er een strategie wordt bepaald wordt eerst ingegaan op de risico’s die een vervanging met zich mee kan brengen. Ook wordt onderzocht waarom een legacy systeem wordt vervangen.

Het ontwikkelen van de nieuwe FAIPR applicatie gebeurd door middel van architectuur. Tijdens de ontwikkeling van een applicatie kan architectuur meer overzicht en structuur geven. Dit resulteert in een applicatie die beter onderhoudbaar is.

De onderzoeken die worden uitgevoerd zijn praktisch van aard. Hiermee wordt bewerkstelligd dat de vervangingsstrategie en architectuur ook echt kunnen worden toegepast.

Bovengenoemde stellingen resulteren in de volgende onderzoeksvragen:

Legacy

- Wat zijn de risico's bij het vervangen van legacy?
- Redenen voor het vervangen van een legacy systeem?
- Welke legacy vervaningsstrategie?

Architectuur

- Waarom architectuur?
- Welke architectuur stijl?
- Welke referentie architectuur?

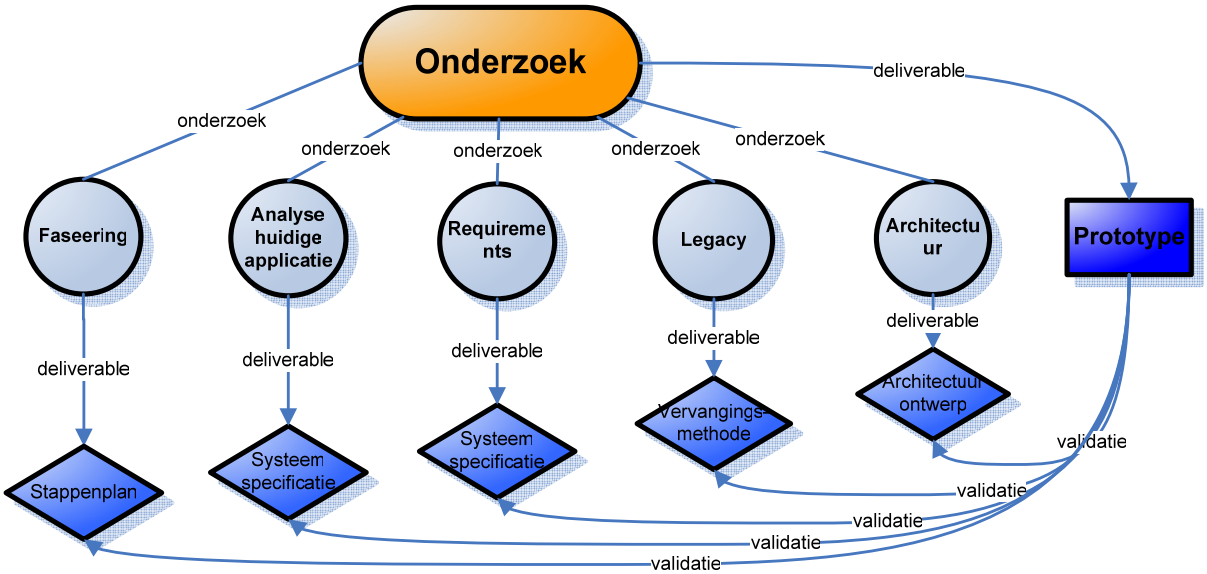
2.6 Doelstellingen

Het doel van dit project is het vervangen van het huidige systeem door een nieuw systeem, die de deelopdrachten en opdrachten zelf op een gedistribueerde wijze kan uitvoeren. Dit zoals uitgebreid beschreven staat in het hoofdstuk 'Afbakening en aannames'.

- Bepalen van de fasering.
- Opstellen van een systeemspecificatie.
- Afleiden van de juiste legacy methode.
- Met het gegeven legacy methode en systeemspecificatie de architectuur.
- Het uitwerken van een prototype.

3 Plan van aanpak

In dit hoofdstuk worden de stappen beschreven die ter ondersteuning dienen voor het uitvoeren van dit project. In dit project wordt met name ingegaan op de hoofdonderzoeksgebieden: legacy en software architectuur.



Bovenstaand figuur geeft schematisch weer hoe het verloop van dit project is gegaan. De verschillende onderzoeksfasen hebben elk hun eigen deliverables waar vanuit verder wordt gegaan wat uiteindelijk als resultaat een prototype is. Uiteindelijk kan door middel van het prototype een reflectie worden gemaakt op de deliverables van elke fase. Hierdoor kan op een betrouwbare manier worden onderzocht of uit de onderzoeken een juiste conclusie is gekomen.

Fasering
De uitvoering van dit project is opgebouwd met behulp van een standaard software ontwikkelingstraject. Door deze uit te breiden met de project specifieke eigenschappen is het resultaat een aansluitend, gefaseerd stappenplan.

Analyse huidige applicatie
Door de huidige applicatie functioneel in kaart te brengen kan er een inventarisatie gemaakt worden van de modules voor code hergebruik.

Requirementsenginieering
Hier wordt vastgesteld wat de requirements en kwaliteitsattributen zijn van het nieuw te ontwikkelen frame-work.

Legacy
Het hoofdonderzoeksgebied ‘Legacy’ bestaat uit drie deelonderzoeken:
Risico’s bij het vervangen van legacy Het vervangen van een legacy applicatie brengt risico’s met zich mee. Door deze te inventariseren kunnen deze door middel van maatregelen terug gedrongen worden.

- Redenen voor het vervangen van een legacy systeem?: Het vervangen van een legacy systeem gebeurd vaak wegens standaard redenen. Door deze te onderkennen kan voorkomen worden dat het systeem wederom vervalt in een legacy systeem.
- Welke legacy vervangingsstrategie?: Het vervangen van een legacy systeem kan volgens standaard strategieën. Het voordeel hiervan is dat deze zich vaak al heeft bewezen. Voor de vervanging van de FAIPR zal een vervangingsstrategie worden vastgesteld.

Software architectuur
Na een korte introductie over software architectuur, worden de drie onderzoeksvragen behandeld:

- Waarom architectuur?: In dit onderzoek wordt bekeken of de keuze voor het toepassen van architectuur een juiste is geweest. De aannamen dat door het toepassen van architectuur het software traject beter verloopt, wordt hierin onderzocht.
- Welke architectuur stijl?: Binnen de software architectuur bestaan verschillende software architectuur stijlen. In het hoofdstuk uitvoering wordt een keuze gemaakt uit een selectie van verschillende stijlen.
- Welke referentie architectuur?: Op basis van de stijl zal een passende referentie architectuur worden geselecteerd. Dit gebeurd op basis van meerdere referentie architecturen. Uit dit onderzoek komt een aanbeveling voor het opzetten een daadwerkelijk ontwerp.

Binnen het onderzoeksgebied architectuur zijn twee additionele onderzoeksvragen opgezet. Deze zijn niet genoemd in hoofdstuk 2 paragraaf ‘onderzoeksvraag’. Toch is tijdens de daadwerkelijke uitvoering gebleken dat beide vragen voor dit project belangrijk zijn.

- Welke architectuur views?: Voor het ontwerpen van de daadwerkelijke architectuur zijn verschillende views voor verschillende stakeholders nodig. In dit onderzoek wordt een selectie gemaakt uit de beschikbare views en worden deze daadwerkelijk ontworpen.
- Welke techniek voor interproces communicatie?: Door dat dit een praktisch project betreft diende er een communicatie mechanisme ontwikkeld te worden ten behoeve de communicatie tussen de verschillende modules. In dit onderzoek wordt een selectie gemaakt en hieruit wordt de meest toepasbare techniek geselecteerd.

Resultaten

Binnen dit onderzoek zijn verschillende resultaten behaald. Op het gebied van onderzoek zijn meerdere bronnen uit de literatuur geraadpleegd, waardoor uit de opgestelde onderzoeksvragen een conclusie is geformuleerd. Op basis hiervan zijn verschillende deliverables geproduceerd waar een prototype mee is ontwikkeld.

Ten behoeve hiervan zijn verschillende bibliotheken en modules ontwikkeld. Een van de meest complexe bibliotheek die niet behandeld wordt in dit verslag is het executeren van een commando. Dit zat hem voornamelijk in het kwaliteitsattribuut ‘portability’. Daarnaast is o.a. een bibliotheek ontwikkeld die het gehele SOAP verkeer afhandelt op de achtergrond van een applicatie.

De bibliotheken samen zorgen voor twee modules namelijk de worker en scheduler. De worker draagt zorg voor het executeren van een applicatie die aangestuurd wordt door de scheduler.

Samenwerking

Deze opdracht is hoofdzakelijk zelfstandig uitgevoerd waarbij veel vrijheid is gegeven. In overleg is besloten om de opdracht te veranderen waardoor het toepassen van een software architectuur een belangrijk onderdeel werd. De overige samenwerking met mijn begeleider bestond uit gesprekken over de voortgang en discussies over bepaalde ontwerp beslissingen. Dit waren vaak diepgaande gesprekken waarin alle voor- en nadelen werden behandeld.

4 Uitvoering

In dit hoofdstuk wordt de uitvoering beschreven van dit project waarin de verschillende onderzoeksvragen centraal staan. De resultaten van de onderzoeksvragen worden door middel van code getoetst. Hiermee worden de uitkomsten uit de theorie getoetst in de praktijk.

Door de beperkte tijd in combinatie met de omvang van de opdracht moet efficiënt worden omgegaan met de tijd. Wanneer bepaalde uitkomsten en of inzichten van een onderzoek voldoende zijn om verder mee met het project te gaan is dit voldoende.

4.1 Software ontwikkelingproces

[SOMMERVILLE 00] Het standaard software ontwikkelingsproces is een set van proces activiteiten. Deze activiteiten leveren uiteindelijk het product op. De vier fundamentele proces activiteiten zijn:

- specificatie
- ontwikkeling
- validatie
- evolutie

Voor het vervangen van de huidige FAIPR is ook gekozen voor een gefaseerde aanpak. De standaard die Sommerville voorschrijft wordt hiervoor uitgebreid aangezien het hier een vervanging betreft.

Doordat het huidige systeem als legacy is beschouwt wordt binnen deze context in de literatuur gezocht. Volgens [SOMMERVILLE 00] is de eerste stap voor het vervangen het reverse engineering fase (zie [CHIKOFSKY 90] voor een uitgebreide beschrijving van de twee termen).

De uitvoering van deze fase geeft inzicht of code/component hergebruik mogelijk is. Ook eventuele impliciete requirements en/of bedrijfsprocessen worden weer expliciet. Een bijkomend voordeel is dat dit een uitstekende manier is om de ontwikkelaars kennis te laten maken met het systeem. Hierdoor is een groot deel van de requirements alvast bekend.

De tweede fase is de requirements inventarisatie fase. In grote lijnen is reeds bekend hoe het nieuwe systeem dient te functioneren (zie paragrafen ‘Ontwikkeling van een frame-work’ en ‘Afbakening en aannames’). In deze fase worden ter aanvulling hiervan de kwaliteitsattributen en scenario’s vastgesteld en gedocumenteerd, conform [BASS 98]. De huidige FAIPR dient als uitgangspunt voor de requirements.

Nadat de kwaliteitsattributen, scenario’s en requirements zijn bepaald samen met de opdrachtgever. Wordt onderzocht welke legacy vervangingsstrategie aansluit op dit project. Deze fase is een logische gevolg op de voorgaande twee fases wat bevestigd word door [WU 97] [BLOMSMA 03] [ROELVINK 03].

In de fase software architectuur wordt bekeken hoe de gestelde eisen en wensen van de opdrachtgever gevormd kunnen worden in een architectuur. Dit is tevens de laatste fase die binnen de scope van dit project valt. De uitvoering van de stappen software ontwikkeling, software evolutie en software validatie vallen buiten de scope.

Conclusie
De uitvoering van dit project bestaat uit de volgende fasering:

- Analyse van de huidige applicatie
- Opstellen requirements
- Bepalen legacy vervangingstraject
- Ontwikkeling van een software architectuur

De fases software ontwikkeling, software validatie en software evolutie zoals deze binnen het standaard software ontwikkelingsproces gedefinieerd worden vallen buiten de scope van dit project.

4.2 Analyse van de huidige applicatie

Dit project betreft het vervangen van een oude applicatie door één met dezelfde basis functionaliteit. Het verschil zit in de uitvoering die op een gedistribueerde wijze plaatsvindt (zie hoofdstuk 2); database management; logging en een betere onderhoudbaarheid. Conform de voorgaande paragraaf is de eerste stap een analyse van de huidige applicatie. Het doel is een goede basis voor de selectie van een legacy vervangingsstrategie. Hiervoor worden de niet-functionele eisen, context, stakeholders en structuur/architectuur van het systeem bepaald.

4.2.1 Bepaling van de analyse methode

De analyse methode dient de twee gestelde doelen te kunnen ondersteunen. Daartoe moet bekend zijn op welke criteria de vervangingstrategie gekozen wordt. Er bestaan verschillende papers met onderzoeken naar analyse methodes voor legacy systemen [MULLER 97], in [Brand 97] worden verwijzingen gemaakt naar onderzoeken hierover.

Een groot nadeel van het toepassen van deze methodes is dat zij moeilijk in gebruik zijn [MULLER 97] [RUGABER 04]. In [RUGABER 04] wordt zelfs gesteld dat de tijd die nodig is voor het uitvoeren hiervan vrijwel onmogelijk is in te schatten.

Op basis van eerdere ervaringen met dergelijke projecten kan de analyse ook gebeuren op basis van handmatige inspectie. Afhankelijk van de grootte van de legacy applicatie kost dit een bepaalde tijd. De huidige applicatie is ongeveer 3000 regels. Met behulp van de bestaande documentatie en de oorspronkelijke programmeur is gekozen om de code handmatig te analyseren.

4.2.2 Context

Gebruikers van het FAIPR systeem plaatsen een opdracht tot een analyse, door de medische beelden via FTP of DICOM protocol te plaatsen in een directory. Wanneer er binnen een gestelde tijd die specifiek is voor het scanner protocol geen nieuwe medische beelden in de directory er bij komen wordt de serie als compleet ontvangen beschouwt. Wanneer er in de tussentijd andere opdrachten binnen komen worden deze geplaatst in een queue.

4.2.3 Stakeholders

Veel mensen, afdelingen en/of teams zijn betrokken tijdens het ontwikkelen van een software systeem. Dit soort ‘groepen’ mensen worden ook wel stakeholders genoemd, elk van deze stakeholder vertegenwoordigd zijn eigen belang tijdens de ontwikkeling van een software systeem. In dit project worden de volgende stakeholders onderscheiden:

- PACS beheer
- Systeembeheer afdeling radiologie
- Onderzoekers, radiologen en laboranten
- Applicatie beheerder

De PACS beheerders dragen zorg voor alle patiënt gegevens, deze worden gecentraliseerd opgeslagen in een database. Systeembeheer onderhoud het netwerk en alle computers. Tevens draagt het systeembeheer zorg voor de toegang tot het netwerk. De eindgebruikers zijn de laboranten en radiologen, deze stakeholders maken de analyse op basis van de medische beelden welke verkregen zijn uit de CT en MR scanners. De laatste stakeholder is de applicatiebeheerder, naast het goed functioneren van de applicatie, is hij vooral geïnteresseerd in management informatie: gebruik van, CPU, analyses, end-users, traceren van fouten etc.

4.2.4 Structuur architectuur van de huidige FAIPR

Er kan niet gesproken worden van een modulaire opbouw in de source code, in de gehele applicatie komen namelijk afhankelijkheden voor met andere services. Wel is het zo dat de code van een bepaalde services redelijk bij elkaar staan in een of meerdere files. Door de verschillende services te onderkennen kunnen deze later makkelijker hergebruikt worden.

Data import
Een scanner plaatst DICOM files in de betreffende directory. Waarna de FAIPR applicatie deze files inleest en valideert of de DICOM files voldoen aan de DICOM standaard.

Data export routing
De output van de analyses die de huidige applicatie ondersteunt genereert wederom DICOM files. Deze data wordt uitgestuurd via het DICOM of FTP protocol

Scheduler
Alle opdrachten worden in een standaard FIFO (first in first out) queue bijgehouden. Dit gebeurd in een text file. Wanneer een opdracht klaar is wordt de opdracht verwijderd uit de queue. Een nieuwe binnengekomen opdracht wordt als laatste toegevoegd aan deze queue.

XML running service Base
Door middel van vastgestelde criteria in een XML files worden DICOM files gekoppeld aan een analyse. Het achterliggende idee is dat er bepaalde ‘steekwoorden’ (lees als criteria) in de DICOM header staan. Wanneer een DICOM file overeenkomt met de criteria die zijn vastgelegd in het XML bestand weet het systeem aan welke analyse de DICOM file gekoppeld moet worden. Deze procedure wordt voor elke binnengekomen DICOM bestand herhaalt.

Log
Wanneer er een fout optreed binnen het huidige FAIPR systeem wordt dit gemeld met behulp van een e-mail bericht en op de web log. Via de webbeheer interface worden de ontvangers ingesteld.

4.3 Requirementsengineering

Het vastleggen van de eisen en wensen van de verschillende stakeholder kan op verschillende manieren gebeuren. In dit project wordt er gewerkt onder architectuur. In [BASS] staat beschreven hoe de werking van de applicatie dient te worden vastgelegd.

Er wordt onderscheidt gemaakt in functionele requirements en kwaliteitsattributen. Daarnaast worden er diverse scenario's beschreven die de requirements ondersteunen. Dit tezamen wordt vastgelegd in een soort POR (Programme of Requirements). Doordat de huidige applicatie dient als uitgangspunt voor de requirements van de nieuwe, worden de functionele eisen binnen dit project niet expliciet in een document vastgelegd.

Voor dit project is er daarom geen POR opgesteld maar is overeengekomen met de opdrachtgever om een systeemspecificatie document op te stellen. Dit begint met een korte introductie met uitleg over het project. Hierna worden de belangrijkste kwaliteitsattributen beschreven. Sommige worden ondersteund door middel van scenario's, anderen zijn in tekst uitgewerkt. De onderstaande kwaliteitsattributen zijn in overleg met de stakeholders van het AMC gekozen als belangrijk:

4.3.1 Availability

Om ervoor te zorgen dat dit systeem snel door de gebruikers gebruikt kan worden, dient er een snelle response tijd te zijn in de uitvoering van de verschillende scenario's. Hieronder staat wat de maximale response tijden dienen te zijn:

Scenario	**Response tijd (seconden)**
AA-1, Autonome analysis	Standaard, instelbaar
AA-2, Autonome analysis	Standaard, instelbaar
QR-1, Query request	0,2

Scenario AA-1 en AA-2 zijn niet gebonden aan bepaalde response tijden. Dit omdat een analyse zelf al minstens 10 minuten duurt. Wel moet per analyse de maximale response instelbaar zijn. Hierdoor kan voorkomen worden dat een worker permanent wordt bezet door een analyse die wellicht al is vastgelopen.

4.3.2 Dependability

Het systeem wordt ingezet t.b.v. het analyseren van medische beelden van patiënten, dit ter ondersteuning voor het maken van beslissingen over de toestand van een patiënt. Wanneer dit systeem klinisch wordt toegepast moet een arts kunnen vertrouwen op dit systeem.

Het kwaliteitsattribuut 'dependability' is relatief moeilijk te waarborgen. Het systeem testen is een mogelijkheid, maar volledig testen is te kostbaar en daardoor vrijwel onmogelijk. Om toch de betrouwbaarheid van het systeem te kunnen waarborgen moet tijdens de ontwikkeling aan het volgende worden gehouden:

1. Werken volgens een code standaard
2. Code reviewing
3. Alleen gebruik maken van third party bibliotheken die aan onderstaande eisen voldoen:
    - De bibliotheek moet tenminste 2 jaar in ontwikkeling zijn.
    - Website waar product te downloaden is.
    - Goede documentatie in het engels.
    - Werkende voorbeelden.
    - Tenminste 1 grote (beursgenoteerd of 2000+ medewerkers) organisatie moet van de bibliotheek gebruik maken.

4.3.3 Portability

De nieuwe FAIPR moet werken onder de volgende besturingssystemen:

- Windows XP
- Linux, Debian 3.1

4.3.4 Scalability

Kijkend naar de toekomst waarin het systeem een groot aantal analyse applicaties ondersteunt, zal dit systeem ook gebruikt gaan worden door een bredere gebruiksgroep. Hierdoor moet de applicatie de mogelijkheid ondersteunen om flexibel uitgebreid te worden.

4.4 Vervangen van FAIPR

FAIPR is ontwikkeld door een onderzoeker van de afdeling radiologie met het doel een prototype te bouwen dat getest kan worden in de kliniek. Door deze manier van ontwikkelen is er geen documentatie beschikbaar, behalve in de vorm van presentaties die vrij algemeen zijn.

Dankzij het succes van de FAIPR is er een groeiende vraag ontstaan naar het automatisch verwerken (analyseren) van medische beelden. Deze vraag uit zich op twee manieren: Er is behoefte naar een verhoging van de capaciteit voor het verwerken van de binnenkomende opdrachten. Ook is er een uitbreiding gewenst op de beschikbare analyses. Deze uitbreidingen worden niet ondersteund door de huidige structuur van het framework. Hierdoor is 'restructuring' (zie bijlage 1) nodig.

Deze paragraaf begint met een inleiding met o.a. de definitie van een legacy systeem. De hierna volgende subparagrafen wordt ingegaan op drie onderzoeksvragen die zijn opgesteld t.b.v. de vervanging van het huidige systeem.

De eerste onderzoeksvraag gaat in op wat de risico's zijn bij het vervangen van legacy applicaties hierdoor kan rekening gehouden worden met de corresponderende problemen. Ook de redenen voor het vervangen van een legacy systeem wordt onderzocht. Hierdoor kan onderzocht worden of vervanging van de nieuwe applicatie in de toekomst voorkomen kan worden. De laatste onderzoeksvraag is de bepaling van een standaard legacy vervangingsstrategie.

4.4.1 Inleiding

In veel organisaties zijn software systemen aanwezig die als te waardevol beschouwd worden om uit gebruik te nemen, maar niet meer flexibel en rigide zijn om te worden onderhouden en uit te breiden. Dit worden ook wel legacy systemen in de literatuur genoemd.

In [BROD 95] wordt de volgende definitie van een legacy systeem gegeven: "Any information system that significantly resists modification and evolution to meet new and constantly changing business requirements.". Vanuit dit gezichtspunt kan gesteld worden dat FAIPR een legacy systeem is. Andere definities van legacy systemen zijn te vinden in o.a. [O'CALLAGHAN 97] [BENN95] [FOLD96].

4.4.2 Risico's bij het vervangen van legacy systemen?

Door te bekijken welke risico's er zijn bij het vervangen van legacy systemen kan inzicht worden verkregen in de problemen die zich kunnen voordoen tijdens mijn project. Volgens [SOMMERVILLE 00] brengt het vervangen van een legacy systeem niet te onderschatten risico met zich mee. Belangrijker om te weten is welke risico's dit volgens Sommervile precies zijn. In hetzelfde artikel maakt hij een opsomming welke naar zijn mening de belangrijkste zijn, namelijk:

1. De ontwerp documentatie is achterhaald. Dit heeft tot gevolg dat de ontwerpdocumentatie niet wordt vertrouwd, waardoor men voor de zekerheid de source code gaat bestuderen en de ontwerp documentatie wordt genegeerd.
2. Huidige bedrijfsprocessen zitten verweven in de legacy systeem. Wanneer de applicatie vervangen wordt moeten de bedrijfsproces opnieuw werken met de applicatie. Dit kan onvoorspelbare kosten en consequenties met zich mee brengen.
3. Er zit waardevolle domein kennis in de legacy systeem die niet expliciet beschikbaar is. Hierdoor kan tijdens het herontwerpen verkeerde beslissingen worden gemaakt over de functionaliteiten en mogelijkheden die applicatie moet hebben.
4. Het ontwerpen van de nieuwe applicatie zelf is al risico vol. Software ontwikkeling [CHAOS 94] gaat in 70% van verkeerd doordat de software niet of niet helemaal aan de specificaties voldoet. Daarnaast is het gevaar ook dat het budget overschreden wordt of het project niet op tijd af is.

Door de risico's in een vroeg stadium van dit project te onderkennen kunnen er voorzorgsmaatregelen worden genomen. Hieronder staan de maatregelen beschreven die worden genomen tegen de legacy vervangrisico's.

In dit project is er weinig documentatie beschikbaar. De oorspronkelijke programmeur van het FAIPR systeem is wel aanwezig voor eventuele assistentie. Hierdoor kan gevraagd worden om de expliciete kennis die in het systeem zit over te brengen. Door regelmatig meetings te houden kan voorkomen worden dat waardevolle kennis verloren gaat. Hiermee worden punten 1, 2 en 3 zo goed mogelijk ingedekt. Daarnaast is de source code beschikbaar en doordat de applicatie niet dusdanig groot is alles nog te overzien.

Punt vier daarentegen, hoe triviaal dan ook, blijft belangrijk. Door middel van verschillende hulpmiddelen zoals een systeemspecificatie, architectuur, voortgangsbesprekingen wordt het risico terug gedrongen.

4.4.3 Redenen voor het vervangen van een legacy systeem?

Met behulp van het inzicht in de redenen voor het vervangen van een legacy systemen, kan ook worden onderzocht of deze niet waren te voorkomen. Hierdoor kan er een applicatie worden geschreven die langer aan de eisen en wensen voldoet van de stakeholders.

Er moet gerealiseerd worden dat dit gedeeltelijk toekomst voorspellen blijft. Daarom word er eerst onderzocht wat de meest voorkomende redenen zijn voor het vervangen van een applicatie. Hierna wordt geanalyseerd of één van deze redenen ook terug komt in de gestelde probleemstelling. Met behulp van deze kennis kan extra aandacht worden gegeven om dit probleem vroegtijdig in te dekken met tegen maatregelen.

[SOMMERVILLE 00] Ondanks de risico's, genoemd in de voorgaande subparagraaf, bij het vervangen van een legacy systeem gebeurd dit toch op grote schaal. Dit komt doordat het aanhouden van een legacy applicaties naar mate ze ouder worden steeds kostbaarder wordt. Zelfs legacy applicaties die pas een paar jaar oud zijn, kunnen erg kostbaar worden om te onderhouden door de volgende redenen:

- De applicatie is ontwikkeld door verschillende ontwikkel teams. Hierdoor is er geen consistente programmeer stijl over het gehele systeem is toegepast.
- Delen van de applicatie zijn geprogrammeerd in een niet meer gebruikte programmeer taal. De kennis om deze software te onderhouden is daarom schaars.
- Documentatie is vaak niet volledig of niet meer actueel. In sommige gevallen is de enige documentatie de source code. En soms gebeurd het dat de source code niet meer beschikbaar is en de executable de enige versie van het systeem vormt.
- Dankzij jaren van onderhoud is de applicatie zijn oorspronkelijke structuur verloren waardoor het zeer tijdrovend is om de structuur van de applicatie te doorgronden.
- De applicatie is alleen maar ontwikkeld voor performance zonder dat er rekening gehouden is met het begrijpbaar houden van de code. Hierdoor is het moeilijk voor programmeurs om de code te begrijpen.
- De gegevens die verwerkt worden door het systeem worden onderhouden in verschillende bestanden met een ongelijke structuur. Hierdoor kan er onnodig gedupliceerd worden of inconsistent raken.

Als de probleemstelling wordt vergeleken op overeenkomsten met de bovenstaande redenen voor het vervangen van een applicatie. Kan er gesteld worden dat er geen overeenkomst is. Dit kan inhouden dat in dit project voor het vervangen van onze software niet tot standaard zijn. Kijkend naar de reden voor de start van dit project kan gesteld worden dat dit door het ontbreken van gedistribueerde verwerkingsmogelijkheden de gehele applicatie anders dient te worden opgebouwd.

4.4.4 Welke legacy vervangingsstrategie?

Dankzij ervaringen uit het verleden bestaan er verschillende standaard procedures voor het vervangen van legacy applicaties. Wanneer één van deze standaarden te gebruiken is als hulpmiddel kan de vervanging op efficiënte wijze worden verricht.

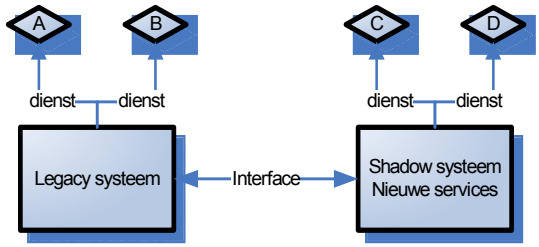
In [BLOMSMA 03] [ROELVINK 03] worden drie mogelijke strategieën beschreven. Aan te nemen valt dat er meer strategieën bestaan. Dankzij het relatief kort tijdsbestek van dit project wordt niet onderzocht of andere strategieën interessant kunnen zijn.

Tabel 1, [Blomsma 03] Overzicht van strategieën

	Strategie	Impact	Investering	Efficiëntie verbetering
A	parallelle implementatie/shadowing	Klein	Klein	Geen – kosten nemen toe
B	Software renovatie	Groot	Klein	Groot – korte en lange termijn
C	Software migratie	Groot	Groot	Groot – lange termijn

Bovenstaand tabel toont de drie strategieën met daarbij wat de impact, investering en efficiëntie verbetering voor het bedrijf is. [Brand 95] Voordat er een strategie kan worden gekozen, en een traject kan worden gestart, voor het vervangen van een legacy applicatie moet de applicatie eerst geïnventariseerd en gespecificeerd worden. Dit proces wordt ook wel reverse engineering genoemd (meer informatie over deze termen zie “Bijlage 1, Reverse engineering en design recovery”).

A) Parallelle implementatie / shadowing
[Blomsma 03] Bij parallel implementeren kiest een organisatie ervoor om nieuwe functionaliteit te realiseren met nieuwe technologie en bestaande legacy systemen te laten staan. Qua ontwikkeling is dit eigenlijk het standaard software ontwikkelingsproces. Het bijzondere zit hem in de interface (zie figuur parallelle implementatie). Hierdoor kan de nieuwe functionaliteit gebruik maken van bijvoorbeeld de gegevens van het oude legacy systeem (zie bijv [DIETRICH 89]). De voordelen van deze techniek zijn:



Figuur 1, parallelle implementatie

- De scope van het project wordt beperkt tot de nieuw te realiseren software.
- Slechts een (klein) deel van de IT afdeling hoeft omgeschoold te worden.

Niet elk systeem leent zich voor een dergelijke interface en moet de huidige structuur zich maar net lenen voor de nieuwe services.

B) Software renovatie
Bij software renovatie wordt er gekozen voor het hergebruiken van de functionaliteit die de legacy systemen vertegenwoordigen. De huidige structuur van de software wordt grafisch kaart gebracht. De gewenste delen van de source code, zoals business rules, workflow, functionaliteit, data-access etc. worden geëxtraheerd. Er wordt een nieuwe structuur of architectuur voor het systeem ontwikkeld waarin de geëxtraheerde programma code ingezet wordt.

Een voordeel van deze techniek is dat door het hergebruiken van de source code een hoop tijd en geld bespaard kan worden. Een maar is dat de nieuwe code compatible moet zijn met verouderde technieken moet gaan samenwerken. Hierdoor kan niet altijd gekozen worden voor de nieuwste technieken of methodes.

C) Software Migratie
“Migratie bestaat uit het proces en de gehanteerde technieken en hulpmiddelen om bestaande software systemen te vervangen door (nieuwe) systemen die voldoen aan nieuwe wensen en/of makkelijker onderhoudbaar zijn.” [MAAT 98]

Bovenstaande definitie geeft goed aan hoe breed het begrip migratie is. Ook het herontwikkelen van een nieuw systeem zonder gedeeltes van het oude systeem toe te passen kan migratie zijn. Er kan dus gesteld worden dat de weg tussen bron (legacy systeem) en doel (nieuw systeem) het migratie is en dat deze weg op verschillend ingevuld kan worden. Dit wordt ook wel een migratiestrategie genoemd.

Conclusie
Het eerst genoemde voordeel van ‘shadowing’ werkt nadelig voor dit project. Het betreft namelijk niet het toevoegen van functionaliteit. De keuze tussen software renovatie en migratie ligt een stuk moeilijker. Bij renovatie moet rekening gehouden worden met de toegepaste technologieën. Het risico is dat de oude source code het nieuwe systeem gaat beperken in mogelijkheden door het niet kunnen toepassen van recente technieken. Daarnaast is het huidige systeem opgezet als prototype met weinig technische documentatie. Dit kan een indicatie zijn dat het huidige systeem snel en onzorgvuldig is opgezet. Op basis hiervan wordt gekozen voor software migratie.

4.5 Software architectuur

De uitvoering van deze opdracht wordt gedaan met behulp van software architectuur. Dit wordt ook wel ontwikkelen onder software architectuur genoemd. In dit hoofdstuk wordt kort ingegaan op wat software architectuur is en enkele relevante architectuur stijlen. Ook worden voor- en nadelen onderzocht van het werken onder architectuur. waardoor inzicht wordt verkregen waarom voor het ontwikkelen met een architectuur is gekozen.

Het ontwerpen van een architectuur is niet nieuw en in de loop der jaren zijn er referentie architecturen ontworpen. Er zal worden bekeken welke geschikt is voor gebruikt in deze situatie. Als laatste wordt bekeken welke architectuur views er ontworpen moeten worden.

Figuur 2, Gaudi (de la Sagrada Familia)

4.5.1 Inleiding

Software architectuur is een abstracte beschrijving op hoog niveau van de structuur van een software applicatie. Dit met als doel om over deze structuur te kunnen redeneren en uitspraken doen zonder te spreken in concrete programmatuur. Een architectuur is een beschrijving van een structuur die nog om verschillende manieren te interpreteren valt [FLORIJN 97].

Het toepassen van architectuur, als ondersteuning voor het ontwikkelen van software, is nog een jong vakgebied. Pas eind jaren ‘80 kwamen er ideeën om een architectuur ook te gebruiken voor het ontwikkelen van software applicaties. Op dit moment bestaan er diverse definities van software architectuur.

Een bekende architect in de bouwkunde is Christopher Alexander. Hij heeft een collectie van 253 patterns verzameld en die zijn uitgesplitst in drie categorieën, namelijk steden, gebouwen en constructie. Ook in de software architectuur is dit overgenomen en zo zijn er architecturen patterns en design patterns ontstaan.

In de literatuur bestaat er een uitgebreid assortiment aan verschillende stijlen waardoor het onmogelijk is om deze allemaal samen te vatten. De volgende drie stijlen worden in dit onderzoek voor het bepalen van een architectuur behandeld:

- Client server
- Distributed
- Service georiënteerd

4.5.2 Introductie in de termen

Architectuur
Het uitgangspunt in de software architectuur is het op hoog abstractie niveau beschrijven van de werking van systemen. Hierdoor kan er over worden geredeneerd en kunnen er uitspraken worden gedaan zonder te spreken over concrete technische oplossingen [SHAW 89]. Een architectuur geeft een structuur aan die op verschillende manieren valt te implementeren.

Architectuur stijl
Een architectuur stijl is een beschrijving voor het invullen van bepaalde requirements. Hierdoor kunnen er meerdere technische implementaties bestaan. Twee grootheden op dit gebied, Garlan en Shaw hebben in verschillende onderzoeken [GARLAN 94] [SHAW 94] [GARLAN 96] [SHAW 96] naar software architectuur verschillende stijlen ontwikkeld.

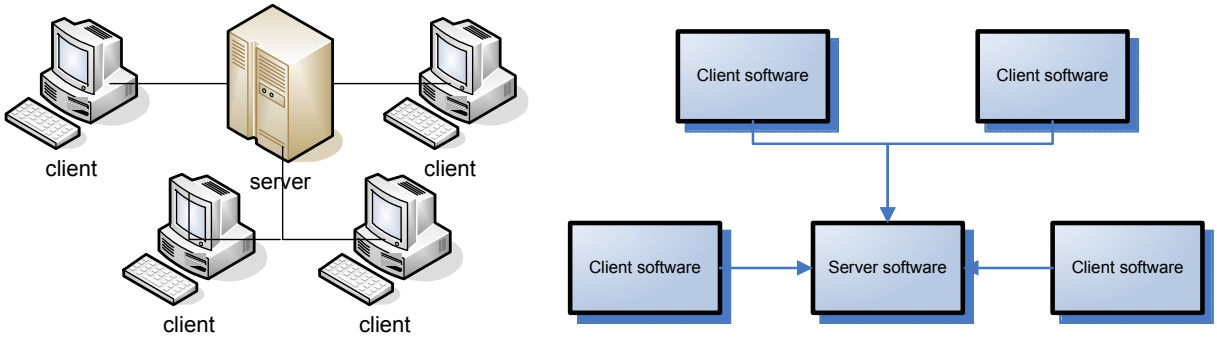
Referentie architectuur
Dit zijn abstracte architecturen die als basis dienen voor het ontwerp van concrete architecturen, dat wil zeggen architecturen van te realiseren informatiesystemen.

Architectuur views
Is een beschrijving van een architectuur van uit een bepaald perspectief die vaak bedoelt is voor een specifieke stakeholder.

4.5.2.1 Client/server architectuur

In 1980 werd de term client/server voor het eerste gebruikt als referentie naar PC's in een netwerk omgeving waarbij deze PC's gebruik maakten van een server. In de software engineering is het client/server model geaccepteerd in eind 1980 [SADOSKI 97].

Net zoals er in een computer netwerk, waarbij meerdere computers (cliënts) gebruik maken van 1 server, gaat dit principe ook op voor het cliënt/server model in de software architectuur (zie figuur Cliënt/Server model).



Figuur 2, Cliënt/Server model

4.5.2.2 Distributed architectuur

Bij distributed computing worden relatief kleinere rekentaken verdeeld over verschillende nodes. Hierdoor kan een opdracht in een kortere tijd worden uitgevoerd. Eén van de vele definities van distributed is dan ook "spread out or scattered about or divided up".

Binnen een distributed architectuur gebeurd het verdelen van de deeltaken door een broker of ORB (Object Request Broker) [PETKOV 03]. Vervolgens gebeurd de uitvoering door workers [STANKOVIC 02] [JAFFAR 03]. De cliënts die gebruik maken van de worker resources, hebben, doordat de broker voor hen transparant is, hier vervolgens niks van door.

4.5.3 Waarom architectuur?

Het toepassen van een software architectuur is redelijk nieuw, een gevaar hiervan kan zijn er nog relatief weinig praktische ervaring mee is. Om te voorkomen dat er straks een complete architectuur opgezet wordt die niet goed gebruikt kan worden, staat dit deelonderzoek stil bij de voor- en nadelen.

[GARLAN 00] Er bestaan diverse architectuur stijlen, elk geschikt voor een bepaald toepassingsgebied. Naast deze stijlen kan een architect ook kiezen om een architectuur te ontwerpen, die een combinatie kan zijn van meerdere stijlen. Hiervoor kan gekozen worden als er naar de mening van de architect geen goede referentie stijl bestaat.

Doordat er voor het selecteren van de juiste stijl geen 'handleiding' [SHAW 94] bestaat kan er eenvoudig een verkeerde keuze worden gemaakt. Bekend is dat een verkeerde ontwerp beslissing aan het begin de software ontwikkelingsproces grote gevolgen heeft voor het vervolg van het proces.

Wanneer er geen gebruik wordt gemaakt van architectuur, kan de ontwerper volgens het standaard software ontwikkelingsproces, na het maken van een functioneel ontwerp wordt het OO model ontwikkelt. In [LAINE 01] wordt gesteld dat het ontwikkelen van een OO model erg moeilijk is doordat niet geheel bedacht kan worden wat een individuele onderdeel precies doet. Zij bevelen zelfs het maken van een development view aan. Hierna kan er vanuit deze architectuur een OO model gemaakt worden. Met als voordeel dat de ontwerper een beter idee heeft wat ook geldt bij grote software systemen van wat de applicatie precies doet.

Naast de development view wordt er in de praktijk ook gebruik gemaakt van andere views waarin verschillende informatie voor verschillende stakeholders wordt gepresenteerd. In [BASS 03] wordt het [KRUCHTEN 95] model aanbevolen en gesteld dat deze geschikt is voor de meeste projecten. Elke view toont de applicatie vanuit een bepaald perspectief. Hierdoor is het voor verschillende stakeholders makkelijk te begrijpen wat de consequenties voor het gebruik en invoering van de applicatie. Binnen dit project zal later bepaald worden welke views nodig zijn.

In [BASS 03] wordt gesteld dat wanneer er meteen ontwerpbeslissingen gemaakt worden, dit een positieve bijdrage levert aan het software ontwikkelproces. Hierdoor kunnen tegenstrijdigheden in een vroeg stadium worden ontdekt. Naast dit voordeel biedt software architectuur volgens [BASS 03] ook het voordeel van communicatie tussen stakeholders, dit wordt ook bevestigd door [GARLAN 93], [MONROE 97] en [PERRY 92]. Doordat een architectuur abstract is kan er afhankelijk van eventuele veranderingen in kwaliteitsattributen gekozen worden voor een ander platvorm.

[GARLAN 00] stelt dat naast deze drie voordelen er ook het voordeel is van hergebruik van componenten en evolutie. Dankzij een architectuur wordt het hergebruiken van een module die al eerder is ontwikkeld eenvoudiger. Dit komt doordat de koppelingen en functionaliteit hetzelfde zijn en doordat er een beter overzicht is dankzij het abstractie niveau kan eerder gezien worden welke componenten er al gemaakt zijn en welke nog niet.

Samengevat kan gesteld worden dat door het toepassen van een architectuur, er gebruik gemaakt kan worden van verschillende referentie architecturen. Daarnaast biedt het maken van een architectuur verschillende andere belangrijke voordelen. Voor het ontwerpen van deze applicatie zal ik dus kiezen voor het ontwerpen van een architectuur. Alvorens de daadwerkelijke ontworpen kan worden zal er eerst onderzocht moeten worden welke referentie architectuur er gebruik kan worden en welke views er gemaakt gaan worden. Deze twee vragen zullen eerst behandeld worden alvorens de daadwerkelijk ontworpen zal worden.

4.5.4 Welke architectuur stijl?

In de loop der jaren zijn voor verschillende ontwerp problemen standaard ontwerp oplossingen ontwikkelt, deze worden ook wel referentie architecturen genoemd. Voorbeelden zijn de stijlen genoemd in [SHAW 96]. Door te kijken naar architectuur stijlen toegespitst op dit project wordt onderzocht welke stijl de beste is om als referentie te gebruiken.

In de probleemstelling kwam naar voren dat door de toenemende vraag voor het verwerken van medische beelden, meerderde analyses parallel uitgevoerd dienen te worden. Belangrijk om nogmaals te vermelden is dat één analyse niet verspreid op verschillende nodes uitgevoerd hoeft te worden.

Uit onderzoek van [SHAW 96] blijkt dat 'distributed' client/server architecturen voor deze probleemstelling een oplossing kan bieden. Andere mogelijke referentie architecturen zijn te vinden in onderzoek naar grid computing, zoals in het onderzoek van [FOSTER 01]. Over het algemeen zijn dit complexere oplossingen voor het parallel uitvoeren van rekentaken. Hierdoor is het beter om te zoeken naar een standaard architectuur dan te gaan onderzoeken hoe grid computing werkt. Uiteraard blijft de optie open om te kijken naar hoe ze daar omgaan met distributed computing.

Doordat grid computing een breed vakgebied beslaat, is het risikant om dit te onderzoeken. De korte onderzoekstijd staat dit niet toe. Daarom is ervoor gekozen om grid-computing buiten de scope te plaatsen.

4.5.5 Welke referentie architectuur?

In het vorige hoofdstuk is geconcludeerd dat een gedistribueerde architectuur stijl de toenemende vraag naar beeld verwerking kan ondersteunen. Nu moet er een keuze worden gemaakt welke referentie architectuur toe wordt gepast. Door in de literatuur naar een of meerdere bewezen oplossingen te zoeken kan op basis hiervan een gewenste referentie stijl worden gekozen.

[MORISAWA 01a], An Architectural Style of Product Lines for Distributed Processing Systems

In de onderzoeken van [MORISAWA 01a] worden negen referentie architecturen beschreven die op basis van data opslag en communicatie onderling verschillen. De locatie is geclassificeerd als gecentraliseerd en gedistribueerd waarbij de laatste uitgesplitst is in synchronous- en asynchronous processing. Een keuze uit de processing type en data type resulteert met dit model in een referentie architectuur.

Doordat uit de opgestelde referentie architectuur niet af te leiden is wat computing nodes (De werkers die de analyses processen) en front-end. Er kan gesteld worden dat de module cliënt zoals dit wordt gepresenteerd in [MORISAWA 01a] de computing nodes voorstellen maar dan is naar mijn inziens de voorgestelde referentie architecturen te vaag. Hierdoor kan er geen goede keuze worden gemaakt uit een van de architecturen.

Processing type between C/S	Data type (*1) / (*2)	Centralized	Distributed	
			Synchronous Processing	Asynchronous Processing
Synchronous Processing	Transaction Type	Centralized Transaction Style	Distributed Transaction Style	Asynchronous Transaction Style
	Query Type	Centralized Query Style	Distributed Query Style	Asynchronous Query Style
Asynchronous Processing	Notification Type	Centralized Notification Style	Distributed Notification Style	Asynchronous Notification Style

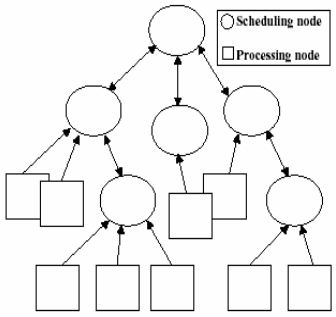
(*1) Processing type between Servers (*2) Message Type

[PETKOV 03] , MultiTiered Distributed Computing Platform

Een gedistribueerde architectuur stijl die op basis van een boomstructuur de verwerking van de deel opdrachten verzorgt. Er wordt onderscheid gemaakt tussen twee modules:

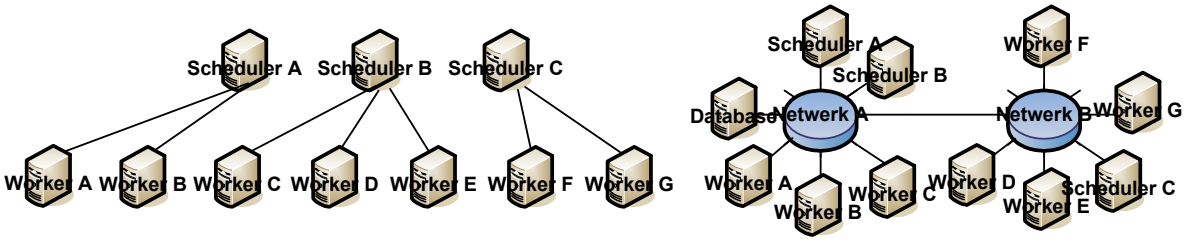
- Scheduling node
- Worker node

Er is één root scheduling node waaronder meerdere scheduling en worker nodes bestaan. Ook hieronder bestaan weer n aantal scheduling nodes. Het verwerken van de opdrachten gebeurd door de worker nodes. Zie het figuur Multi tiered distributed computing platform voor een grafische presentatie.



Figuur 3, Multi tiered distributed computing

De voorgestelde architectuur uit [PETKOV 03] is goed toepasbaar voor dit project. Om de ontwikkeling te versnellen wordt de multie tiered techniek voorlopig weg gelaten. De consequenties hiervan is dat de aantal workers wordt gelimiteerd zo stelt [PETKOV 03] in zijn verslag. Dit geeft echter geen problemen aangezien het aantal wat hiermee wel wordt ondersteund gewoon toereikend is voor dit project. Het resultaat wordt dan het onderstaand ontwerp waarin duidelijk te zien is dat de gelaagdheid weg is.



4.5.6 Welke architectuur views?

Er bestaan verschillende soorten views, belangrijk is dat de views gekozen worden waarmee de verschillende stakeholders uiteindelijk verder mee kunnen werken. Uit het uitgebreide programma van views zullen er een paar gekozen worden om uitgewerkt worden.

Een invloedrijke paper van [KRUCHTEN 95] waarin vier views worden gekozen als uitgangspunt voor het ontwerpen van een architectuur. Dit model van wordt ook wel ‘four plus one’ genoemd. De naam hiervoor komt doordat de scenarios (plus one) de vier (four) architectuur ontwerpen mapped op elkaar. De volgende vier views worden in dit model onderscheiden:

- Logical
- Process
- Development
- Physical

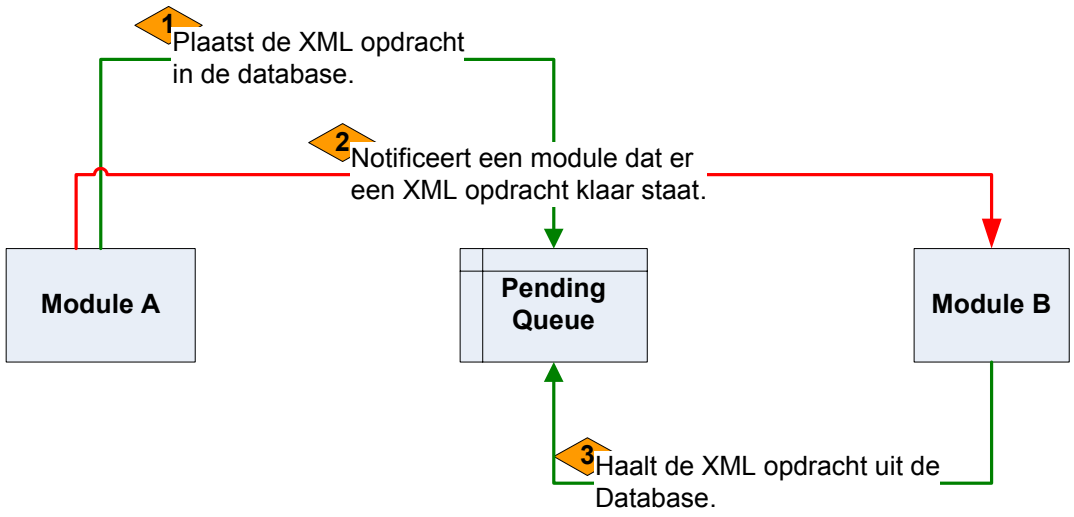
Op ongeveer hetzelfde tijdstip dat Kruchten zijn werk presenteerde kwam ook in [SONI 95] met vier views om een architectuur mee te presenteren. Deze views komen overheen met de views uit [KRUCHTEN 95] wat wordt bevestigd door [BASS 03]. Het grote verschil is dat in het model van Kruchten gebruik wordt gemaakt van scenario’s om de views te verifiëren.

Voor het ontwerpen van de architectuur van het nieuwe FAIPR applicatie zal gebruikt worden gemaakt van het ‘four plus one’ model zoals deze in [KRUCHTEN 95] wordt gepresenteerd. De keuze voor het model van [KRUCHTEN 95] in plaats van het model van [SONI 95] komt doordat

4.5.7 Welke techniek voor interproces communicatie?

De nieuwe FAIPR applicatie bestaat uit meerdere modules, elke module bezit een bepaalde functionaliteit/service. Wanneer een module gebruik wil maken van een service die een andere module levert is daar een communicatie methode voor nodig. In figuur, communicatie ontwerp is weergegeven hoe in het nieuwe ontwerp de communicatie verloopt tussen modules.

Een opdracht wordt geplaatst door een XML bericht in de database te plaatsen (zie oranje ruit 1). Hierna wordt de desbetreffende module genotificeerd dat er een opdracht klaar staat (zie oranje ruit 2). De module haalt het bericht op (zie oranje ruit 3). In dit onderzoek wordt bepaald welke implementatie van een techniek gebruikt gaat worden voor de realisering van stap twee.



Figuur, Communicatie ontwerp

Communicatie tussen verschillende modules kan op basis van verschillende technieken, daarnaast zijn er vaak verschillende implementaties beschikbaar van één techniek. Dit onderzoek is niet bedoeld om volledig te zijn maar is een quick-scan van de huidige ontwikkelingen. Dit kan dus betekenen dat een mogelijke interessante technologie niet meegenomen wordt in dit een onderzoek. Wanneer geen van de technieken voldoet aan de gestelde requirements vindt er een heronderzoek plaats.

De uiteindelijke selectie zal gebeuren op basis van de mate waarin de techniek voldoet aan de kwaliteitsattributen. Dit omdat alle technieken tenminste moeten voldoen aan de gestelde requirements alvorens zij in verdere onderzoek mee doen. De selectie zal gebeuren op basis van de mate waaraan de techniek voldoet aan de gestelde kwaliteitsattributen dependability, portability en availability. Scalability is in dit geval niet van toepassing aangezien dit van toepassing is op de architectuur en niet op het communicatie protocol.

Op basis van de techniek die aan de gestelde eisen voldoet word een prototype gebouwd. Hierdoor kan bekeken worden of een daadwerkelijke implementatie van de techniek overeenkomt met de verwachtingen. Hiermee wordt voorkomen dat veel tijd verloren gaat wanneer de techniek niet mocht werken om welke reden dan ook.

4.5.7.1 Remote Procedure Calls (RPC)

Een implementatie van de techniek RPC is SOAP (Simpel Object Access Protocol). Er bestaan verschillende SOAP implementaties waaronder gSOAP. Deze techniek is geschikt voor het creëren van applicaties volgens de SOA (Service Oriented Applications). Net als bijvoorbeeld CORBA (Common Object Request Broker Architecture) is SOAP bedoeld voor het creëren van gedistribueerde applicaties. In tegenstelling tot methodes als CORBA is SOAP platform/programmeertaal onafhankelijk. De belangrijkste voordelen van SOAP zijn [BARENDSE 05]:

Availability: [gSOAP 1] Uit praktijk onderzoek is gebleken dat het SOAP protocol voldoet aan de gestelde eisen van availability.

Dependability: Op twee punten blijkt dat SOAP voldoet aan de gestelde kwaliteitsattribuut dependability. Het eerste punt waarop dit te zien is, is dat de gSOAP bibliotheek met regelmaat wordt geüpdatet [gSOAP 1]. Hierdoor kan gesteld worden dat problemen worden verholpen en extra toevoegingen worden gemaakt. Het tweede punt is dat de documentatie duidelijk en uitvoerig is, die bij de GSOAP bibliotheek geleverd wordt.

Portability: Voor vrijwel elk besturingssysteem is er een SOAP implementatie ontwikkeld, hetzelfde geldt voor de programmeertalen. De gSOAP implementatie daar in tegen is ontwikkeld voor C++ compilers zoals die van Visual Studio 4 of hoger (Windows) en g++ (Linux). Doordat SOAP een open standaard is en gSOAP hieraan voldoet is gSOAP bijvoorbeeld compatible met de microsoft implementatie van SOAP. Daarnaast maakt SOAP gebruik van gestandaardiseerde protocollen zoals:

HTTP (Hyper Text Transfer Protocol), SOAP biedt de mogelijkheid om via HTTP te communiceren met andere service interfaces. Aangezien eigenlijk alle servers wel over TCP beschikken is de implementatie van HTTP eenvoudig. Een groot voordeel van HTTP is dat firewalls dit meestal niet blokkeren. Ook is het mogelijk om SSL (Secure Socket Layer) toe te voegen aan HTTP. Hierdoor is de beveiliging op een transparante manier te regelen voor het systeem.

XML (eXtensible Markup Language**),** Tussen de service interfaces gaat de communicatie over HTTP. De berichten die uitgewisseld worden zijn in het XML formaat. In tegenstelling tot bijvoorbeeld CORBA is dit een formaat wat voor een mens begrijpelijk is. Dit maak het opsporen van fouten een stuk eenvoudiger.

4.5.7.2 Message Passing (ook wel Message Queuing genoemd)

Message passing is een methode waarbij applicaties onderling berichten naar elkaar kunnen sturen zowel asynchroon als synchroon. Deze methode die erg goed te gebruiken is voor het koppelen van loosely-coupled of autonome applicaties. Een van de bekendere implementaties is QMP.

Availability: Op (website: http://www.lqcd.org). Wordt gesteld dat QMP geschikt is voor real-time communicatie. Hierdoor voldoet hij aan de gestelde eisen.

Dependability: Deze techniek is in verhouding met RPCs erg onvolwassen [THAM], dit valt o.a. te merken aan de aantal beschikbare API's. QMP is bij het zoeken op google naar bekende message passing API's de meest gevonden toch is de API relatief moeilijk te downloaden (website: http://www.lqcd.org). Dit wil natuurlijk niks zeggen over de kwaliteit van QMP maar zegt genoeg over de betrouwbaarheid van dit project.

Portability: De huidige API's zijn alleen geschikt voor een bepaalde programmeertaal en of besturingssysteem.

4.5.7.3 Database

Een andere methode voor onderlinge communicatie is het gebruik van een relationele database. Met behulp van deze methode kan er gekozen worden voor twee technieken namelijk: Triggers en Polling. Beide technieken hebben hun eigen voor en nadelen.

4.5.7.3.1 Triggers

Een database trigger is een soort stored procedure die wordt gestart wanneer er iets gebeurd in de database [ORACLE 1]. De database kan vanuit de stored procedure een applicatie starten [ORACLE 2].

De volgende voor en nadelen zijn gevonden van het gebruik van deze techniek:

Availability: [ORACLE 1] De techniek triggering is nagenoeg real-time, nadat er een bepaalde actie is plaats gevonden wordt de trigger direct geactiveerd.

Dependability: MySQL is een van de meest gebruikte database in de wereld waaronder verschillende multinationals (http://www.mysql.com/customers/).

Portability: Niet elke relationele database ondersteunt het gebruik van triggers zo is het bijvoorbeeld in de meest gebruikte database op internet namelijk MySQL nog steeds een beta functionaliteit. Daar in tegen ondersteunen fabrikanten als Oracle en MSSQL dit wel.

4.5.7.3.2 Polling

Een andere manier van berichten uitwisselen tussen applicaties is het gebruik van een polling mechanisme. Elke applicatie kijkt op een gestelde frequentie of er wat staat in de database wat uitgevoerd moet worden.

Availability: Deze mechanisme is niet geschikt voor real time systemen. Dit komt doordat er op een vooraf gestelde frequentie wordt bekeken of actie nodig is waardoor dit nooit echt real time kan gebeuren. Hierdoor zit er altijd een bepaalde wacht tussen waardoor bijvoorbeeld aan de front end een wachttijd ontstaat.

Dependability: MySQL is een van de meest gebruikte database in de wereld waaronder verschillende multinationals (http://www.mysql.com/customers/).

Portability: De meeste database zijn te benaderen door meerdere besturingssystemen en aan te spreken via een gestandaardiseerde taal namelijk SQL.

Makkelijke implementatie
Het implementeren van deze techniek is zeer eenvoudig. Dit komt doordat er vaak al een database wordt gebruikt voor het opslaan van gegevens. Hierdoor is het toevoegen van de message tabellen zeer eenvoudig, hetzelfde geldt ook voor de polling queries.

4.5.7.4 Conclusie

Om een real-time systeem te ontwikkelen waarbij rekening wordt gehouden met meerdere platvormen zal goed bekeken moeten worden aan welke eisen de techniek moet voldoen. Elke techniek heeft zijn voor en nadelen waarbij Message Passing eigenlijk geen uitspringende voordelen heeft ten opzichte van de andere twee technieken. Het gebruik van een database in combinatie met polling heeft als belangrijkste voordeel dat het relatief eenvoudig is toe te passen ten opzicht van de andere twee technieken maar is niet real-time. Database met triggers overkomt dit probleem maar met als nadeel dat de functionaliteit is gelimiteerd door de instructie set van de database.

Voor het realiseren van real-time communicatie, tussen modules onderling, die platvorm onafhankelijk is kies ik voor het toepassen van SOAP. De RPC eigenschappen van SOAP zijn hier uitermate geschikt voor. De standaardisatie van SOAP is doorslaggevend geweest bij het maken van de keuze.

5 Resultaten

In dit hoofdstuk wordt kort samengevat wat de resultaten van dit project zijn. Door een tweedeling te maken in de belanghebbenden kan snel ingezien worden voor wie, wat is ontwikkeld. De eerste belanghebbende is mijn opdrachtgever, het AMC. De tweede is de UvA.

5.1 AMC

Het AMC is met name geïnteresseerd in de opgeleverde documentatie/ontwerpen, deel en eind producten. Deze zijn voortgevloeid uit de verschillende onderzoeken in de paragraaf uitvoering. Hieronder een overzicht per categorie wat er opgeleverd is:

Documentatie,
Ten behoeve van het uitwerken en ontwikkelen van het FAIPR systeem zijn er diverse ontwerpen en documenten gemaakt.

- Systeem specificatie document
- Architectuur ontwerpen
    - Physical view
    - Logical view
    - Model view
    - Proces view
    - Scenario's
- Code standaard

Bibliotheken
De bibliotheken zijn allen volgens de gestelde requirements in het 'Systeem specificatie' document ontwikkeld. Met name het kwaliteitsattribuut 'portability' zorgde voor veel werk tijdens de ontwikkeling. Dit doordat Windows en Linux in sommige gevallen verschillen in de beschikbare functies.

- Database
    - MySQL connection
    - data container
- Inter process communicatie
    - SOAP manager
    - single threaded
    - multi threaded
- Command executing
- Logging
    - Stream
    - Database

Modules
Van de onderstaande drie modules, zoals deze ook in het 'programme of requirements' en 'architectuur views' staan gespecificeerd, is een prototype ontwikkeld.

- Worker
- Recipe selector
- Scheduler

5.2 Onderzoek

De wetenschappelijke kant van dit verslag is met name te vinden in de paragraaf ‘uitvoering’. Hierin komen de verschillende onderzoeken naar voren die tijdens mijn stage plaats gevonden hebben.

Hieronder, in een kort overzicht, nogmaals de gedane onderzoeken:

- **Aanpak**
  - Welke structuur voor ‘Plan van Aanpak’?
- **Requirementsengineering**
  - Requirementsengineering
- **Legacy**
  - Risico’s bij het vervangen van legacy?
  - Redenen voor het vervangen van een legacy systeem?
  - Welke legacy vervangingsstrategie?
- **Architectuur**
  - Waarom architectuur?
  - Welke architectuur stijl?
  - Welke referentie architectuur?
  - Welke techniek voor interproces communicatie?

5.3 Persoonlijk

Persoonlijk vind ik “Welke referentie architectuur?” het meest leerzame onderzoek. Ondanks dat ik al eerder met software architectuur in aanraking ben geweest, bleek uit dit onderzoek hoe breed dit vak gebied eigenlijk wel niet is. Hierin maakte ik kennis met onder andere service georiënteerd- en distributed architectuur.

Een zeer uitdagend en uiterst moeilijk onderzoek is het onderzoek naar SOAP geweest. Vooral op programmeer technisch gebied bleek het opvangen van de standaard output en standaard input erg uitdagend.

Niet alles waarmee kennis is gemaakt tijdens dit project staat beschreven in deze scriptie. Hieronder een overzicht met alles waar kennis over opgedaan is:

- UML
- XML
- C++
- Werken met Visual studio 2003
- Programmeren onder:
  - Windows
  - Linux

6 Evaluatie

6.1 Wat is er bereikt?

Voor het AMC is er en prototype ontwikkeld waarmee de architectuur bewezen is. Dit is een aanzet voor de nieuwe applicatie waarmee de ontwikkelaars binnen de afdeling radiologie verder mee kunnen. Door het ontwikkelen van deze prototype heb ik veel kennis opgedaan hoe platform onafhankelijk geprogrammeerd kan worden.

6.2 Wat ging er mis?

In het begin van mijn stage wist ik niet wat precies de balans moest zijn tussen theorie en praktijk. Aan de ene kant moest er binnen drie maanden een scriptie ingeleverd worden en aan de andere kant een functioneel prototype. Een bijkomend probleem was dat dit project te groot voor drie maanden is, waardoor ik vaak dagen van gemiddeld 10 à 11 uur maakte en ook in het weekend aan het werk was. Dit alles bij elkaar zorgde ervoor dat ik niet tevreden was over mijn scriptie en prototype.

Door de relatieve onervarenheid van mij en mijn begeleider met software architectuur was het opstellen moeilijker dan gedacht. Dit resulteerde in veel veranderingen tijdens de ontwikkelingsfase waardoor vele uren aan werk onbruikbaar werd.

Het kwaliteitsattribuut ‘portability’ heeft veel ontwikkelingstijd gekost. Het werk zat in het schrijven van operating systeem specifieke source code. Ook moest veel tijd worden gestoken in onderzoek naar de realisatie hiervan.

Additionele programmeer werkzaamheden werden uitgevoerd door een tweede programmeur die de realisatie van verschillende componenten zou verzorgen. Ondanks goede afspraken over de werkverdeling kwamen sommige essentiële componenten voor mijn modules erg laat waardoor deze pas in de laatste week geïntegreerd werden in mijn modules. Dit is grotendeels te wijten aan het gebrek aan samenwerking wat gedeeltelijk door tijdsdruk komt.

6.3 Tevredenheid opdrachtgever

Mijn opdrachtgever heeft te kennen gegeven erg geïnteresseerd te zijn in de deliverables van de onderzoeken, maar met name in het prototype zelf. Hierdoor werkte ik vooral resultaat gericht en gebeurde de oplevering van deelproducten eens per week. Mijn begeleider reviewde deze waarna kritiek doorbesproken werd. Deze werkwijze werd als goed ervaren door mijn begeleider en was tevreden over mijn geleverde werk.

6.4 Reflectie op onderzoeksaanpak

Door het gebrek aan kennis op het gebied van software architectuur heeft de ontwikkeling van een referentie architectuur veel vertraging opgelopen. Dit komt door de grootte van het onderzoeksgebied. Dit had wellicht voorkomen kunnen worden door scherpere onderzoeksvragen en afbakeningen te formuleren.

7 Bibliografie

[BASS 03] L. Bass, P. Clements, R. Kazman; *Software Architecture in practice, Second Edition*; Addison Wesley; 2003;

Een toonaangevende boek op het gebied van de laatste ontwikkelingen, concepten en best practices in de software architectuur. In dit boek wordt beschreven hoe en waarom software architectuur nodig is en hoe dit toe gepast kan worden in de praktijk. De theorie dat behandeld is wordt aan de hand van diverse praktijk voorbeelden in het boek toegelicht.

[BARENDSE 05] J. Barendse; Universele configuratie methode; 2005;

Onderzoek naar 'service-oriented architecture framework' op basis van gSOAP om universeel beheer van applicaties en systemen mogelijk te maken.

[BENN 95] K.H.Bennett; "Legacy Systems: Coping With Success", IEEE Software, January 1995, Vol 12, No. 1, pp 19-23; 1995;

[BLOMSMA 03] M. Blomsma, Van A naar .NET, 2003

Drie verschillende legacy vervangingsstrategieën worden uit een gezet en word er bekeken hoe deze gebruikt kunnen worden in combinatie met .net technologie van Microsoft.

[BROD 95] M.L.Brodie; M.Stonebraker, "Migrating Legacy Systems", 1995, Morgan Kaufmann Publishers.

[BUSCHMANN 96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M. Stal; *Pattern Oriented Software Architecture: A System of Patterns;* John Wiley & Sons; 1996;

[BRAND 97] M.G.J. van den Brand, P. Klint, C. Verhoef - Reverse Engineering and System Renovation – An annotated Bibliography –

Een overzicht van pointers naar verschillende bronnen over reverse engineering en System renovatie staan hierin. In de bibliografie staat bij elke bron een korte samenvatting.

[COOPER 03] J. W. Cooper C#; *Design Patterns*; Addison-Wesley; 2003;

Een boek waarin de 23 design patterns uit [GAMMA 94] worden beschreven aan de hand van C# code voorbeelden. Doordat naast de abstracte beschrijvingen ook "echte" source code voorbeelden gegeven worden kan er een betere voorstelling gemaakt worden wat een design pattern inhoudt

[CHAOS 94] http://www.standishgroup.com/sample_research/chaos_1994_1.php

[CHIKOFSKY 90] E. J. Chikofsky en J.H. Cross. Reverse Engineering and Design Recovery: a Taxonomy. IEEE Software.

Beschrijft een aanpak op het gebied van reverse engineering. Forward en reverse engineering, her-documenteren, her-ontwerpen, herstructureren en reengineering worden door hem beschreven.

[DIETRICH 89] Walter C. Dietrich JR., Lee R. Nackman en Franklin Gracer; Saving a Legacy with Objects; OOPSLA '89 Proceedings, pagina 77-83.

[FAYAD 97] M. Fayad, D. C. Schmidt; *Object-Oriented Application Frameworks*; ACM**;** Special Issue on Object-Oriented Application Frameworks, Vol. 40, No. 10, October 1997; www.cs.wustl.edu/~schmidt/CACM-frameworks.html;

Dit artikel is een 'globale' beschrijving over OO framewerken zoals MacApp, ET++, Microsoft's MFC and DCOM, RMI, en CORBA. Deze voordelen en de context van deze framewerken worden toegelicht.

[FLORIJN 97] G. Florijn; *Software engineering en software architectuur*; 1997;

[FOLD 96] D. Howe (ed.), "The Free On-line Dictionary of Computing", http://wombat.doc.ic.ac.uk/.

[FOSTER 01] I. Foster, C, Kesselman, S. Tuecke The Anatomy of the Grid, Enabling Scalable Virtual Organizations; 2001;

[GAMMA 94] E. Gamma, R. Helm, R. Johnson, J. Vlissides; *Design patterns: elements of reusable object oriented software*; Addison-Wesley; 1994.

Een van de eerste boeken over design patterns en meteen een belangrijk boek op het gebied van OO design patterns. In dit boek worden 23 OO design patterns beschreven en uitgelegd aan de hand van abstracte voorbeelden.

[GARLAN 93] D. Garlan and M. Shaw; *An Introduction to software architecture. In Advances in Software Engineering and Knowledge Engineering;* pagina 1-39; Singapore; 1993; World Scientific Publishing Company.

[GARLAN 00] D. Garlan; *Software Architecture: a Roadmap*; pagina 91-101; 2000; Addison-Wesley

[GSOAP 1] http://www.genivia.com/Products/gsoap/changelog.html

[HARVEY 98] D. Harvey; The Role of Patterns in Enterprise Architecture; 1998; http://www.davethehat.com/articles/pda.htm;

Uitleg waarom design patterns gebruikt moeten worden in architecturen. Er komt naar voren waarom design patterns handig zijn en hun toepassingsgebieden.

[JACKSON 95] M. Jackson; The world and the machine; ACM; 1995

De relatie tussen de wereld en de machine (software) is niet makkelijk. Deze paper beschrijft de vier facetten waar rekening gehouden mee moet worden bij het opstellen van requirements van software. Daarnaast worden problemen genoemd waardoor het opstellen van requirments toch mis kunnen gaan.

[JAFFAR 03] J. Jaffar, A. E. Santosa, R. H. C. Yap, K. Q. Zhu; Scalable Distributed Depth-First Search with GreedyWork Stealing; 2003

[KIPFER 99] P. Kipfer, P. Slusallek; *Transparent Distributed Processing For Rendering*; ACM; 1999;

In deze paper wordt uitgelegd hoe een transparante infrastructuur kan worden opgezet voor renderen. Dit gebeurd met technieken als design patterns en CORBA. Er wordt uitgelegd hoe de distributed architectuur is opgezet en hoe de technieken zijn toegepast hierin. Ondanks dat de context van de distributed architectuur renderen is wordt wel in de paper gesteld dat de architectuur simpel te gebruiken is voor andere distributed oplossingen in andere contexten.

[KING 04] B. King; Simplify Distributed System Design Using the Command Pattern, MSMQ, and .NET; From the September 2004 issue of MSDN Magazine; 2004; msdn.microsoft.com/msdnmag/issues/04/09/CommandPattern/;

In dit artikel wordt het gebruik van de CommandPattern beschreven in een distributed omgeving. Met behulp van een voorbeeld en modellen wordt toegelicht hoe de pattern werkt.

[KRUCHTEN 95] P. Kruchten; *The 4+1 view model of architecture*; IEEE; 1995;

Er bestaan verschillende views binnen software architectuur. Binnen het Kruchten worden de naar de auteur inziens de vier belangrijkste views in een model beschreven. Met behulp van scenario's worden de vier verschillende architectuur views gecontroleerd.

[LAINE 01] P. K. Laine; The role of SW architecture in solving fundamental problems in object-oriented development of large embedded SW systems; IEEE; 2001

[MAAT] M. Maat en H. Vogt; Migratie - Het legacy probleem aangepakt;

[MCCONNELL 04] S. McConnell; Code complete second edition; Microsoft; 2004;

Een praktisch naslagwerk over de best practices over het ontwikkelen van software. Verschillende 'belangrijke' aspecten zoals testen, patterns, codestandaarden, ontwikkeltrajecten, onderhoud etc. komen aan bod in dit boek.

[MENGOTT 04] T. Mengotti; GPU a framework for distributed computing over Gnutella; 2004;

In dit betoog staat een framewerk beschreven voor gedistribueerd computing. Met behulp van dit framewerk kunnen gebruikers ervan berekening laten uitvoeren op meerdere computers tegelijk. De technieken, architectuur en plugins met grafische output worden in deze paper besproken.

In dit betoog staan problemen en technieken beschreven die tijdens mijn onderzoek ook van toepassing zijn. Een goed voorbeeld is het gebruiken van plugins die tijdens het gebruik ervan een grafisch output geven. Ook mijn applicatie zal te maken krijgen met dit soort plugins. Deze paper en zijn referentie kan mij ondersteunen tijdens het opzetten van een architectuur.

[METTALA 92] E. Mettala and M. H. Graham; *The domain-specific software architecture program;* 1992;

[MONROE 97] R. T. MONROE, A. KOMPANEK, R. MELTON, and D. GARLAN; Architectural Styles, Design Patterns, and Objects; IEEE; 1997;

Na een korte introductie over software architectuur wordt in deze paper ingegaan op de voordelen van gebruik hiervan. Dit gebeurd aan de hand van verschillende voorbeelden.

[MORISWA 01a] Y. Morisawa, K. Torii; An Architectural Style of Product Lines for Distributed Processing Systems, and Practical Selection Method; 2001;

Negen beschrijvingen over gestandaardiseerde architectuur stijlen toegespitst op distributed applicaties. Er wordt uitgelegd hoe deze verschillende stijlen werken.

[MORISWA 01b] Y. Morisawa, K. Torii; An Architectural Style of Product Lines for Distributed Processing Systems, and Practical Selection Method; 2001;

Negen beschrijvingen over gestandaardiseerde architectuur stijlen toegespitst op distributed applicaties. Ook worden de zwakke en sterke punten van elke architectuur stijl toegelicht en wordt een selectie methode beschreven voor het selecteren van een architectuur stijl.

[MULLER 97] Hausi A. Muller; Reverse Engineering Strategies for Software Migration; Department of Computer Science, University of Victoria, 1997;

[O'CALLAGHAN 97] Alan O'Callaghan; Object Technology Migration is not just ReverseEngineering; Object Expert 2 (1997), nummer 1, pagina 16-19; 1997;

[ORACLE 1] http://oracle.ittoolbox.com/documents/document.asp?i=1292

[ORACLE 2] http://www.lc.leidenuniv.nl/awcourse/oracle/appdev.920/a96590/adg11rtn.htm#1001153

[PAGE 03] A. Page, T. Keane,, J. Waldron; MultiTiered distributed computing Platform; 2003;

[PETKOV 03] N. Petkov, J. van den Berg, E. Duval; Grid computing en e-science; 2003;

[PERRY 92] D. E. Perry and A. L. Wolf; *Foundations for the study of software architecture; ACM SIGSOFT Software Engineering Notes;* 1992;

[PYARALI 98] I. Pyarali, T. Harrison, D. C. Schmidt, T. D. Jordan; *Proactor - An Object Behavioral Pattern for Demultiplexing and Dispatching Handlers for Asynchronous Event*; 1998;

Proactor pattern is een design pattern welke toegepast kan worden in concurrent applicaties. Een toelichting op de voor en nadelen van dit design patterns en een c++ voorbeeld van een mogelijke implementatie van de proactor pattern.

[ROELVINK 03] A. Roelvink; Softwarerenovatie Oplossing voor legacy probleem; 2003;

[RUGABER 04] Spencer Rugaber, Kurt Stirewalt; Model-Driven Reverse; IEEE; 2004;

[SADOSKI 97] D. Sadoski; Client/Server Software architectures – an overview; SEI; 1997;

[SCHMIDT 98] D. C. Schmidt; *Reactor - An Object Behavioral Pattern for Demultiplexing and Dispatching Handles for Synchronous Events*; 1998;

De reactor pattern is een beschrijving voor het afhandelen van concurrent service requests van client applicaties. De pattern zorgt ervoor dat elke service op zich zelf staat en elke service wordt gemanaged door zijn eigen dispatching manager. Ook worden de voor en nadelen van de Reactor pattern besproken en wordt er een c++ voorbeeld gegeven.

[SHAW 89] M. Shaw; *Larger scale systes require higher level abstractions*; ACM; 1989;

[SHAW 94] M Shaw; Comparing Architectural design styles; IEEE; 1994;

[SHAW 96] M. Shaw, D. Garlan; Software architecture: perspective on an emerging discipline; Prentice Hall; 1996;

[SOMMERVILLE 00] I. Sommerville; Legacy Systems boek – legacysys; 2000;
Een introductie op legacy systemen, er wordt een beschrijving gegeven wat legacy sytemen karakteriseert en wat de problemen zijn met het vervangen van legacy systemen.

[SONI 95] D. Soni, R. L. Nerd, C. Hofmeister; Software Architecture in Industrial Applications;

[STANKOVIC 02] N. Stankovic, k. Zhang; *A distributed parallel programming framework*; IEEE; 2002

[THAM] C. Tham Distributed Client/Server Architectures and Transaction Processing on Open Systems; *AT&T Global Information Solutions;* members.value.com.au/christie/oow94.htm

[WU 97] B. Wu, D. Lawless, J. Bisbal, R. Richardson, J. Grimson, V. Wade, D. O'Sullivan; The Butterfly Methodology : A Gateway-free Approach for Migrating Legacy Information Systems; IEEE; 1997;

8 Bijlagen,

8.1 Bijlage 1, Reverse engineering en design recovery

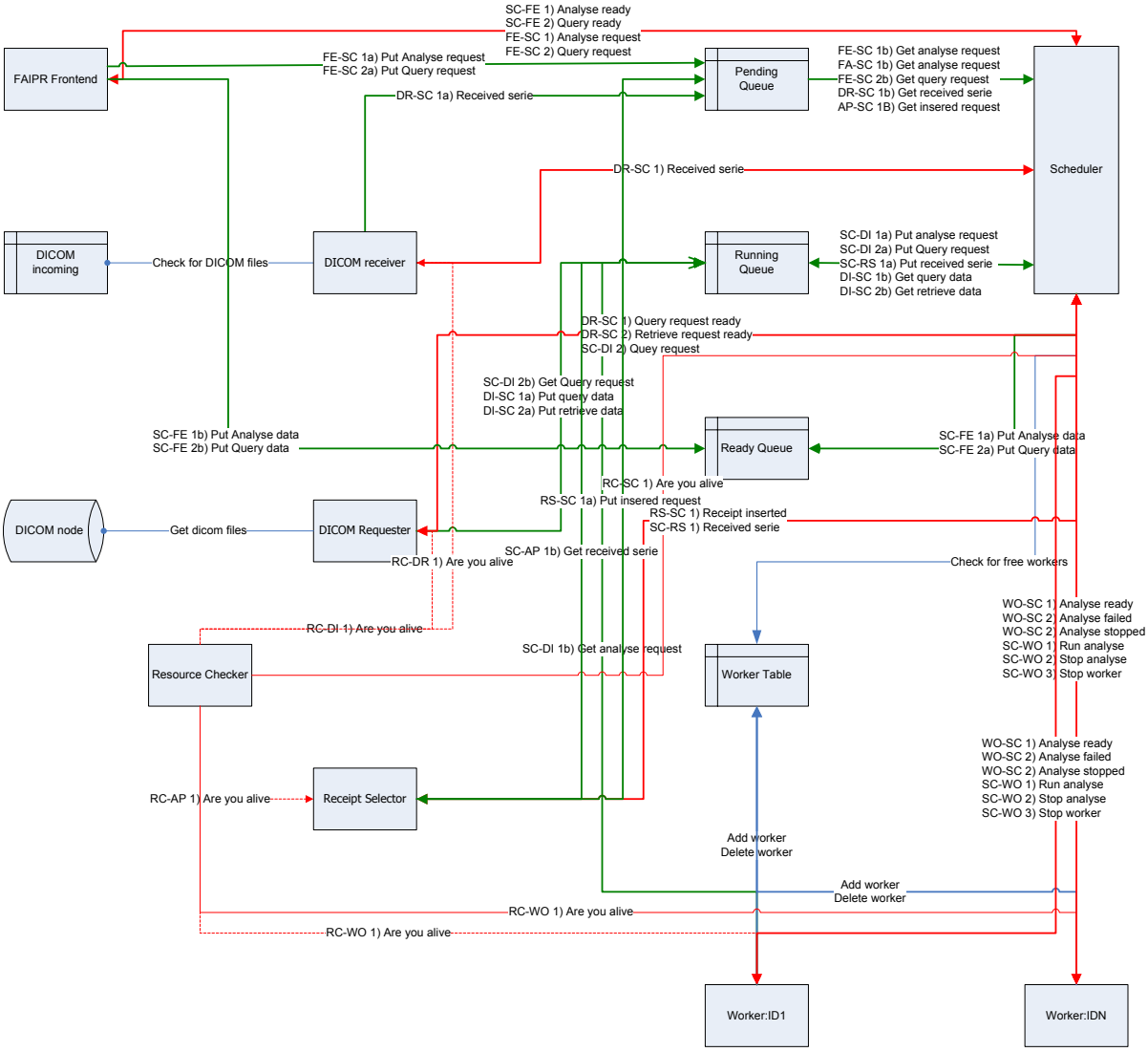
Om succesvol te kunnen bepalen welke strategie gekozen moet worden voor het vervangen van de legacy applicatie, zal er eerst een inventarisatie en specificatie van de applicatie plaats moeten vinden. Dit kan gebeuren door het bestuderen en analyseren van de bijbehorende ontwerp documentatie. In de praktijk blijkt echter dat veel van deze documentatie weg, achterhaald of nooit gemaakt is. Hierdoor is de source code nog de enige bron waaruit de ontwerp documentatie gehaald kan worden.

[CHIKOFSKY 90] Met behulp van de techniek ‘forward engineering’ wordt een abstract ontwerp op hoog niveau omgezet naar een implementatie (bijv. source code) van dit ontwerp. Het omgekeerde hiervan is ‘reverse Engineering’, hier wordt vanuit een implementatie naar een abstract op hoog niveau ontwerp gegaan. Voor beide technieken geldt dat de weergaven op hoog niveau gepresenteerd worden door middel van grafische presentaties en de implementatie is vaak de feitelijke applicatie. Voor beide technieken geldt dat toevoegingen en of veranderingen buiten de scope valt.

‘Redocumentation’ concentreert zich op het maken van een semantisch gelijke beschrijving van het software systeem. Deze techniek wordt vaak toegepast wanneer er geen documentatie meer aanwezig of niet betrouwbaar is. Bij ‘design recovery’ wordt domein kennis en externe informatie gebruikt om een abstracte beschrijving te maken van een software systeem. Bij ’design recovery’ wordt er meer informatie gebruikt dan alleen de source code. Deze informatie kan bijvoorbeeld worden verkregen door domein experts. Met behulp hiervan wordt de structuur van de source code opnieuw in kaart gebracht.

‘Restructuring’ richt zich op het transformeren van een systeem beschrijving naar een andere beschrijving toe, maar blijft op hetzelfde abstractie niveau. Belangrijk is dat het semantisch gedrag van een systeem gelijk blijft na de transformatie, dit houdt in dat er geen veranderingen of nieuwe functionaliteiten worden door gevoerd. Het doel van ‘reengineering’ of renovatie is het maken van een nieuwe specificatie aan de hand van het software systeem. Eventuele nieuwe functionaliteiten mogen worden toegevoegd aan de specificatie waarna het systeem opnieuw wordt ontwikkeld.

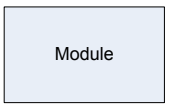
8.2 Bijlage 2, Architectuur - Module view



Legenda

Module

Een module verzorgt een bepaalde service binnen het FAIPR systeem. De module 'scheduler' staat groter afgebeeld dan de anderen waardoor hij beter opvalt. De reden hiervoor is dat dit het centrale coördinatie punt is van al het communicatie verkeer.



Rode lijn

De rode lijnen geven de SOAP communicatie weer. Binnen deze view wordt onderscheidt gemaakt tussen twee type:

- De gestippelde lijnen geven weer dat deze behoren tot de module 'resource checker', die controleert of een bepaalde module nog actief is.
- De rechte lijnen geven de inter-module communicatie weer.

Groene lijn

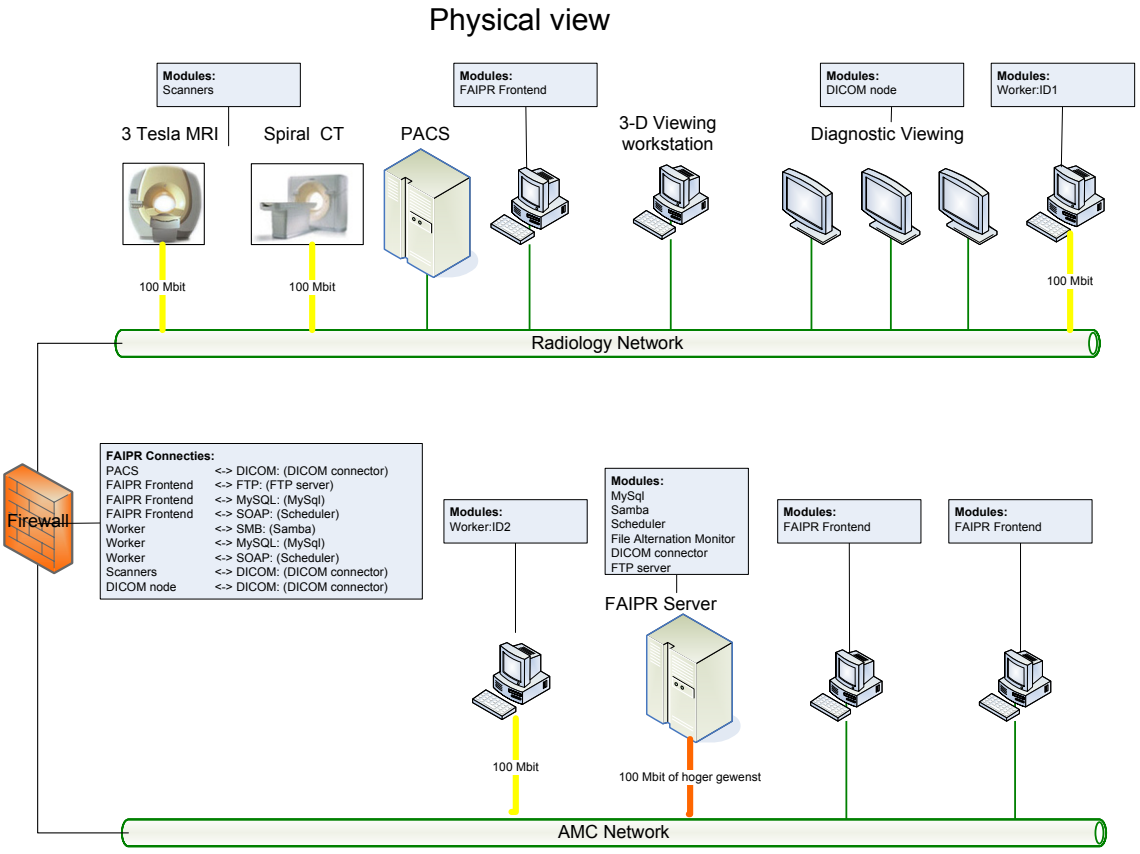
Elke rode lijn wordt vooraf gegaan aan een groene die staat voor de communicatie naar de database toe.

Blauwe lijn

Communicatie naar de database zonder SOAP notificatie.

8.3 Bijlage 3, Architectuur - Physical view

Om goed te kunnen communiceren en discussiëren met het systeembeheer van de afdeling radiologie, is er besloten om een physical view op te stellen. Hierin staan verschillende elementen die op een of andere manier betrokken zijn bij de nieuwe FAIPR.



8.4 Bijlage 4, Systeemspecificatie document

1. Kwaliteitsattributen

In overleg met de stakeholders zijn de onderstaande kwaliteitsattributen belangrijk voor het nieuwe FAIPR systeem:

Availability

Om ervoor te zorgen dat dit systeem snel door de gebruikers gebruikt kan worden, dient er een korte response tijd te zijn in de uitvoering van de verschillende scenario's. Hieronder staat wat de maximale response tijden dienen te zijn:

Scenario	Response tijd (seconden)
AA-1, Autonome analysis	Standaard, instelbaar
AA-2, Autonome analysis	Standaard, instelbaar
QR-1, Query request	0,2

Scenario AA-1 en AA-2 zijn niet gebonden aan bepaalde response tijden. Dit omdat een analyse zelf al minstens 10 minuten duurt. Wel moet per analyse de maximale response instelbaar zijn. Hierdoor kan voorkomen worden dat een worker permanent wordt bezet door een analyse die wellicht al is vastgelopen.

Dependability

Het systeem wordt ingezet t.b.v. het analyseren van medische beelden van patiënten, dit ter ondersteuning voor het maken van beslissingen over de toestand van een patiënt. Wanneer dit systeem klinisch wordt toegepast moet een arts kunnen vertrouwen op dit systeem.

Het kwaliteitsattribuut 'dependability' is relatief moeilijk te waarborgen. Het systeem testen is een mogelijkheid, maar volledig testen is te kostbaar en daardoor vrijwel onmogelijk. Om toch de betrouwbaarheid van het systeem te kunnen waarborgen moet tijdens de ontwikkeling aan het volgende worden gehouden:

1. Werken volgens een code standaard
2. Code reviewing
3. Alleen gebruik maken van third party bibliotheken die aan onderstaande eisen voldoen:
    - De bibliotheek moet tenminste 2 jaar in ontwikkeling zijn.
    - Website waar product te downloaden is.
    - Goede documentatie in het engels.
    - Werkende voorbeelden.
    - Tenminste 1 grote (beursgenoteerd of 2000+ medewerkers) organisatie moet van de bibliotheek gebruik maken.

Portability

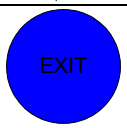
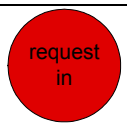
De nieuwe FAIPR moet werken onder de volgende besturingssystemen:

- Windows XP
- Linux, Debian 3.1

Scalability

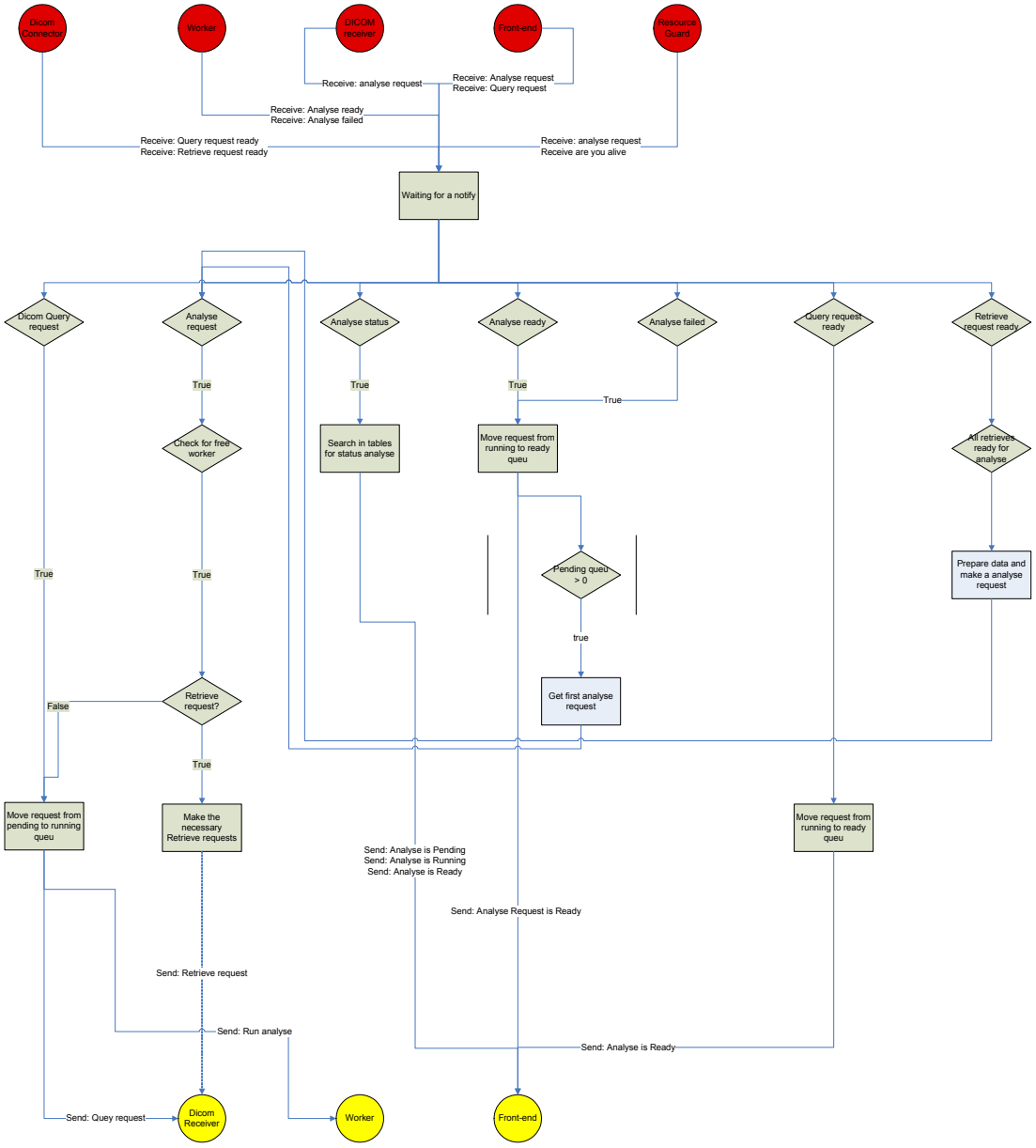
Kijkend naar de toekomst waarin het systeem een groot aantal analyse applicaties ondersteunt, zal dit systeem ook gebruikt gaan worden door een bredere gebruiksgroep. Hierdoor moet de applicatie de mogelijkheid ondersteunen om flexibel uitgebreid te worden.

Legenda

	Er kunnen tegelijkertijd meerdere 'uitvoeringen' plaats vinden. Met behulp van dit teken wordt aangegeven op welke plaats dit mogelijk is.
	Een module die in deze toestand komt wordt beëindigd.
	Binnen elke module worden meerdere SOAP requests onderscheiden. De binnenkomende staan bovenaan en worden weergegeven met een rode cirkel met daarin de naam van de request.
	Uitgaande soap request staan aan de onderkant van elke module. Een uitgaande request is altijd gekoppeld aan een inkomende request.
	Binnen deze stap in het model wordt een bepaalde taak uitgevoerd. Dit kan bijvoorbeeld een rekentaak zijn, beslissing maken of gegevens uit een database verkrijgen.
	Aan de hand van bepaalde criteria wordt een beslissing genomen.

2. Module: Scheduler

De module scheduler verzorgt alle interactie tussen de overige modules en kan gezien worden als een soort router of mediator. Hierdoor is hij erg belangrijk voor het FAIPR systeem, zonder weten de overige modules niet wie waar is. Derhalve is het belangrijk dat hij stabiel en snel werkt.

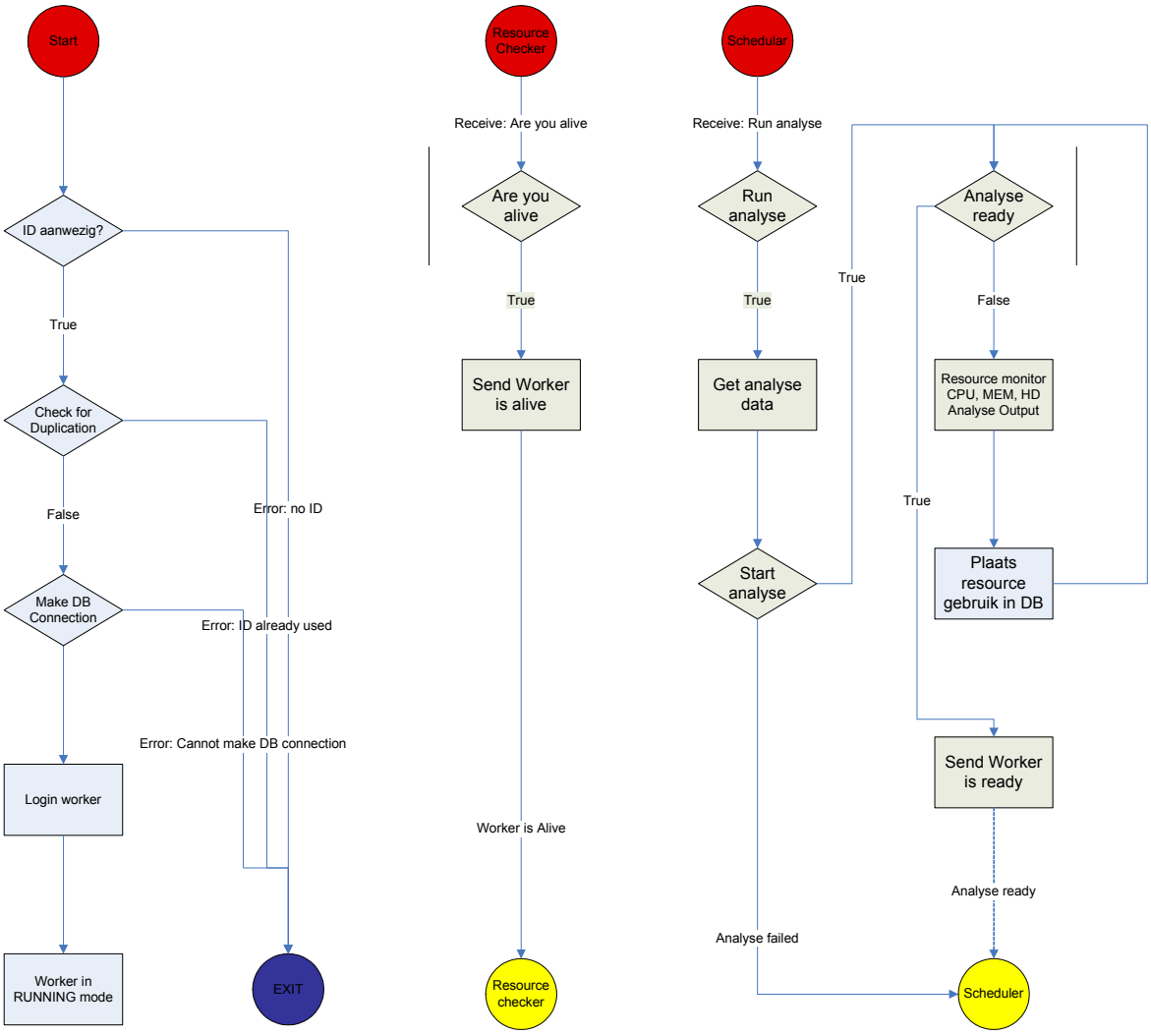


Figuur 4, Sceduler

2.1. De scheduler moet simultaan verschillende services aanvragen kunnen verwerken.
2.2. Ondersteuning van de communicatie in hoofdstuk “Communication module to module”.
2.3. Ondersteuning van de scenario’s in hoofdstuk “Scenario’s”.
2.4. Eens in een bepaalde tijd controleren of alle pending opdrachten al behandeld zijn.
2.5. Afhankelijk van de type analyse moet er bekeken worden welke worker hiervoor in aanmerking moet komen.
2.6. 2.5 moet gebeuren op basis van de volgende resources: harddisk, geheugen, CPU.
2.7. Wanneer er een recept in de pending queue geplaatst is moet de scheduler bekijken welk onderdeel uitgevoerd moet worden van het recept.

3. Module: Worker

De worker module bestaat uit twee gedeelten. De eerste is de initialisatie procedure. Hierin moet de worker zich aanmelden zodat andere modules gebruik kunnen maken van zijn diensten. Hierna zorgt de worker ervoor dat analyses uitgevoerd worden.



3.1. Moet vanaf één centrale plek gestart kunnen worden vanaf een netwerkschijf.
3.2. Identificatie gebeurd door middel van een ID nummer.
3.3. Het ID nummer uit 3.2 moet meegegeven worden in de command line als de worker wordt gestart.
3.4. Er moet gecontroleerd worden of een andere worker al is ingelogd onder hetzelfde ID nummer.
3.5. Als 3.4 positief is meldt de worker zich aan.
3.6. Er moet een applicatie (lees als analyse) kunnen worden gestart.
3.7. De standaard output (stdout) en standaard error (stderr) van de te starten applicatie uit 2.6 moet geplaatst worden in de database.
3.8. Ondersteuning van de communicatie in hoofdstuk “Communication module to module”
3.9. Ondersteuning van de scenario’s in hoofdstuk “Scenario’s”

4. Computersystemen en netwerk

4.1. Het netwerk is onderling verbonden via TCP/IP.
4.2. Het systeem moet minimaal door een totaal van 20 werknemers gelijktijdig gebruikt kunnen worden.
4.3. Het systeem moet 32 werker nodes ondersteunen.
4.4. In 4.4 en 4.3 wordt uitgegaan van een server tenminste met de volgende specificaties (Pentium 4 3.06 GHz, 1024MB intern, 120GB HD, Debian 3.1).
4.5. Het systeem moet draaien op verschillende interne en externe netwerken.
4.6. In 4.5 wordt er vanuit gegaan dat het systeem- en netwerkbeheer de nodige aanpassingen hiervoor aanbrengt in de desbetreffende interne en externe netwerken.

5. Message, Warning and Error logging

5.1. Het loggen is opgedeeld in drie klassen namelijk: Message, Warning en Error.
5.2. In ‘error’ komen log berichten te staan die kritische zijn voor de continuïteit van FAIPR systeem in zijn geheel.
5.3. In ‘warning’ komen log berichten te staan die niet direct een gevaar zijn voor het functioneren van het FAIPR systeem, maar wel echte foutmeldingen zijn.
5.4. In ‘message’ komen standaard proces berichten te staan.
5.5. De modules halen vanaf één centraal punt af of de verschillende log klassen naar de database, scherm en/of file toe moeten worden geschreven.

6. Communicatie

De communicatie tussen onderlinge modules bestaan uit notificaties. Elke notificatie heeft zijn eigen betekenis voor het FAIPR systeem. Hieronder een toelichting op de verschillende notificaties.

- Run analyse, een worker wordt genotificeerd dat er een analyse uitgevoerd moet worden.
- Stop analyse, een worker moet de huidige analyse afbreken.
- Stop worker, een worker moet eventueel zijn analyse afbreken en zich zelf deactiveren.
- Received serie, de Dicom Receiver geeft hiermee aan dat hij een serie heeft ontvangen die geanalyseerd kan worden.
- Query request, er moet een bepaalde query worden uitgevoerd op een DICOM node.
- Query request ready, de qeury request is succesvol uitgevoerd.
- Retrieve request ready, de retrieve request is succesvol uitgevoerd.
- Analyse ready, analyse is klaar en de worker kan nieuw werk ontvangen.
- Analyse failed, analyse uitvoeren is gefaald en de worker kan nieuw werk ontvangen.
- Analyse stopped, analyse is succesvol stop gezet en de worker kan nieuw werk ontvangen.
- Receipt inserted, er is een receipt in de pending table geplaatst. Het is zaak aan de scheduler om de eerst volgende service uit te voeren van het receipt.
- Are you alive, de desbetreffende module geeft aan dat hij nog werkt.

Communicatie module naar module

Binnen het nieuwe FAIPR systeem gebeurd de onderlinge communicatie door middel van notificaties. Hiermee geven modules aan dat iets moet gebeuren of wordt verwacht. Voor de duidelijkheid wordt er onderscheidt gemaakt tussen daadwerkelijke “systeem” notificaties en “controle” notificaties. Hieronder een overzicht van alle notificaties:

FAIPR notificaties

Code	Omschrijving	Service naam
SC-WO	Scheduler to Worker:	
1	Run analyse	runAnalyse
2	Stop analyse	stopAnalyse
3	Stop worker	stopWorker
SC-RS	Scheduler to Receipt Selecter	
1	Received serie	serieReceived
SC-DR	Scheduler to Dicom Requester	
1	Quey request	queryRequest
2	Retrieve request	retrieveRequest
DR-SC	Dicom Requestor to Scheduler	
1	Query request ready	queryRequestReady
2	Retrieve request ready	retrieveRequestReady
WO-SC	Worker to Scheduler:	
1	Analysis ready	analysisReady
2	Analysis failed	analysisFailed
3	Analysis stopped	analysisStopped
DR-SC	Dicom Receiver to Scheduler	
1	Received serie	serieReceived
RS-SC	Receipt selector to Scheduler	
1	Receipt updated	receiptUpdated

Resource Checker

RC-WO, Resource Checker to Worker

- Are you alive

RC-SC, Resource Checker to Scheduler

- Are you alive

RC-RS Resource Checker to Receipt selector

- Are you alive

RC-DR, Resource Checker to Dicom Requestor

- Are you alive

RC-DR, Resource Checker to Dicom Receiver

- Are you alive

Scenario's

Hieronder worden drie scenario's beschreven waarin de verschillende services die het FAIPR systeem zal bieden, uit een worden gezet.

AA-1, Autonome analysis (Nog niet alle series binnen):

Een autonome analyse wordt getriggerd vanuit de DICOM receiver, de serie die daar is binnengekomen wordt bekend gemaakt aan de analyse preparer. Doordat er bijvoorbeeld nog een serie mist kan de scheduler niet verder totdat de andere serie er bij is.

From: Module	SOAP signal	To: Module	Action
DICOM receiver	Received serie	Scheduler	Send notify
Scheduler	Received serie	Analyse Preparer	Send notify
Analyse Preparer	Inserted receipt	Scheduler	Check dependency's, **false**
----------~~~~some time passed~~~~----------			
DICOM receiver	Received serie	Scheduler	Send notify Check dependency's, **true** Check for free worker, **true**
Scheduler	Run analyse	Worker	Nothing
----------~~~~some time passed~~~~----------			
Worker	Analyse failed	Scheduler	Send notify
Scheduler	Analyse failed	Front-end	

AA-2, Autonome analysis (geen werker beschikbaar):

Een autonome analyse wordt getriggerd vanuit de DICOM receiver, de serie die daar is binnengekomen wordt bekend gemaakt aan de analyse preparer. De scheduler kijkt welke opdrachten verwerkt kunnen worden en ziet dat er een analyse plats moet vinden. Hierop volgend raadpleegt hij de worker tabel of er een worker vrij is. Er zijn op het moment van raadplegen geen workers vrij waardoor de scheduler moet wachten totdat er een worker aangeeft dat hij beschikbaar is.

From: Module	SOAP signal	To: Module	Action
DICOM receiver	Received serie	Scheduler	Send notify
Scheduler	Received serie	Analyse Preparer	Send notify
Analyse Preparer	Inserted request	Scheduler	Check dependency's, **true** Check for free worker, **false**
----------~~~~some time passed~~~~----------			
Worker	Analyse ready	Scheduler	Check for new work, **true** Send notify
Scheduler	Run analyse	Worker	Nothing
----------~~~~some time passed~~~~----------			
Worker	Analyse ready	Scheduler	Send notify
Scheduler	Analyse ready	Front-end	

QR-1, Query request

De front-end kan queries uitvoeren op een DICOM node, door middel van de query request notificaties wordt door de scheduler de juiste service aangesproken en wordt de query uitgevoerd.

From: Module	SOAP signal	To: Module	Action
Front-end	Query request	Scheduler	Send notify
Scheduler	Query request	Dicom requester	Get dicom headers Send notify
Dicom requester	Query request ready	Scheduler	Send notify
Scheduler	Query request ready	Front-end	