2013

# Event-driven architecture: Message-framework

Afstudeerscriptie

T. Linschoten
Hogeschool Utrecht
27 mei 2013
Versie 1.0

# Afstudeerscriptie

**Tim Linschoten (1548807)**
Technische Informatica voltijd
27 mei 2013

**Hogeschool Utrecht**
Faculteit Natuur en Techniek
Nijenoord 1
3552 AS Utrecht

**Sogyo**
Landgoed Sandwijck
Utrechtseweg 301
3731 GA De Bilt

## Voorwoord:

Voor u ligt het afstudeerverslag van Tim Linschoten. Dit verslag is gemaakt aan de hand van de afstudeerstage bij Sogyo.

Ik wil de dhr. R. Wolter heel erg bedanken voor zijn begeleiding tijdens de stage. Hij heeft goede duidelijke begeleiding gegeven tijdens het project en was altijd bereikbaar voor eventuele vragen. Ook wil Ik dhr. P. van Rooijen heel erg bedanken voor zijn begeleiding vanuit de Hogeschool Utrecht. Hij heeft veel goede en duidelijke feedback gegeven op het plan van aanpak en op de afstudeerscriptie. Als laatste wil ik Sogyo als bedrijf bedanken voor de goede sfeer en leuke en leerzame afstudeerstageopdracht.

## Revisielijst

| Versienummer | Datum | Reden revisie |
|---|---|---|
| 0.1 | 19-april-2013 | Eest opzet verslag. |
| 0.2 | 10-mei-2013 | Feedback Peter van Rooijen verwerkt en eerste volledige versie van de scriptie. |
| 0.3 | 17-mei-2013 | Feedback Peter van Rooijen verwerkt. |
| 0.4 | 20-mei-2013 | Feedback Peter van Rooijen verwerkt. |
| 1.0 | 27-mei-2013 | Voorwoord geschreven en spelling en taal gecontroleerd. |

## Lijst van Afkortingen

| Afkorting | Term |
|---|---|
| EDA | Event-driven architecture |
| DTO | Data transfer object |
| WPF | Windows Presentation Foundation |

# Management samenvatting

Dit is de scriptie van de afstudeeropdracht van Tim Linschoten. De afstudeeropdracht is gemaakt voor het bedrijf Sogyo. Een van de werkzaamheden van Sogyo is klanten adviseren over de structuur en architectuur van ICT applicaties. Een van de onderwerpen waarin Sogyo geïnteresseerd is event-driven architecture, hierna EDA genoemd. De klanten van Sogyo hebben nog twijfels over het gebruik van EDA. Deze twijfels worden veroorzaakt door één van de eigenschappen van EDA. Bij een EDA applicatie is het lastig om de event stroom te achterhalen. Hiermee wordt bedoeld dat het lastig te zien is welke events er worden verstuurd en welke onderdelen van het systeem hier op reageren. Hierdoor is de applicatie lastig bij te werken en te onderhouden.

Om deze twijfel weg te nemen bij de klanten heeft Sogyo een project gestart dat bestaat uit meerdere afstudeeropdrachten. Dit overkoepelende project moet informatie leveren over EDA aan Sogyo en het moet de twijfels van EDA bij de klanten weg nemen.

Deze afstudeeropdracht is de eerste stap in dit overkoepelende project. De opdracht bestaat uit het opzetten van een framework waarin het mogelijk is om EDA applicaties te ontwikkelen. Met dit framework moet het ook mogelijk zijn om de event stroom van de applicatie te achterhalen. Hierbij moet de informatie gegeven worden om de event stroom te analyseren maar hoeft het analyseren zelf nog niet gedaan te worden. In de afstudeeropdrachten die volgen zal informatie die door het framework geleverd wordt gebruikt worden om de event stroom in beeld te brengen en te analyseren.

De werkzaamheden die zijn verricht om dit framework te realiseren bestaan uit een onderzoek naar de te gebruiken programmeertaal, het opzetten van de architectuur van het framework en het werkelijk implementeren van het framework. Nadat het framework was opgezet is er ook tijd besteed om de gebruiksvriendelijkheid van het framework te verbeteren. De laatste stap in het project was het ontwikkelen van een EDA applicatie in het framework. Deze applicatie dient als bewijs dat het mogelijk is een EDA applicatie te ontwikkelen in het framework.

Op basis van het ontwikkelde framework en de gestelde eisen aan het project kan geconcludeerd worden dat het project geslaagd is.

# Inhoudsopgave

# 1 Inleiding

De inleiding zal dienen als het eerste inhoudelijke hoofdstuk van de scriptie. Eerst zal de inhoud van de scriptie doorgenomen worden. Hierop volgt een beschrijving van Sogyo, het bedrijf waar de afstudeerstage heeft plaatsgevonden. Er zal ook verteld worden welke plaats de student in de organisatie inneemt. Aan het einde van het hoofdstuk is er een glossary te vinden met een aantal belangrijke termen die nodig zijn om de scriptie goed te begrijpen.

## 1.1 Leeswijzer

Het eerstvolgende hoofdstuk zal gaan over de projectbeschrijving. Hierin zal de projectopdracht beschreven worden. Er zal verteld worden waarom het project gestart is en wat het doel van het project is. Ook zal er verteld worden aan welke eisen het project moet voldoen om goed afgesloten te worden. Als laatste zal de projectmethodiek behandeld worden.

Het derde hoofdstuk zal gaan over de uitvoering van het project. Hierin zal de ontwikkeling van het framework en de andere onderdelen van het project zoveel mogelijk chronologisch behandeld worden. Hierin zal worden aangetoond dat de eisen die aan het project zijn gesteld zijn ingewilligd.

In het vierde hoofdstuk zal de conclusie van het project gegeven worden. Hierin zal verteld worden of het project wel of niet geslaagd is en waarom. Ook zullen er aanbevelingen gegeven worden over zaken die in de toekomst verbeterd kunnen worden.

Als laatste hoofdstuk zal een evaluatie van het project volgen. Hierin zal er een evaluatie van het project gegeven worden. Hierna zal de verkregen kennis van dit project worden toegelicht.

## 1.2 Beschrijving Sogyo

Sogyo is gelegen aan de Utrechtseweg in de Bilt in een omgebouwde boerderij. Sogyo houdt zicht vooral bezig met software innovaties. Er werken circa 80 mensen bij Sogyo en ongeveer de helft hiervan is aanwezig in de boerderij. De werkzaamheden van Sogyo zijn op te delen in drie pijlers:

- **Academy:** In de Sogyo Academy wordt er gewerkt aan de ontplooiing van IT-talent. In de Academy worden eigen medewerkers van Sogyo en de medewerkers van de klanten van Sogyo opgeleid.

- **Detachering & Consultancy:** De pijler Detachering & Consultancy werft, selecteert, begeleidt en ontwikkelt ambitieuze en talentvolle IT'ers. De inzet van Sogyo-professionals varieert van tijdelijke inzet op consultancy- of detacheringsbasis tot trajecten die tot een vast dienstverband bij hun opdrachtgevers leiden.

- **Development:** Bij de pijler Development wordt er gewerkt aan de ontwikkeling van IT-toepassingen.

Sogyo houdt zich veel bezig met het opleiden en adviseren. Veel van de klanten van Sogyo zijn bedrijven die zelf IT-toepassingen maken. Om deze bedrijven te kunnen adviseren moet er zo veel mogelijk kennis bij Sogyo aanwezig zijn. Hierom is het belangrijk voor Sogyo om zoveel mogelijk kennis te hebben over de nieuwe ontwikkelingen in het ICT veld.

Als stagiair ben ik geplaatst in de Academy van Sogyo. Hierdoor is er genoeg ruimte om te leren over de verschillende onderwerpen van de stage. Er is op de Academy ook altijd begeleiding aanwezig. De Academy is een plaats waar aan het ontplooien van IT-talent gewerkt wordt. Hierdoor is dit een perfecte plaats voor een stagiair.

## 1.3 Glossary

In deze glossary zullen een aantal termen worden toegelicht die nodig zijn om deze scriptie goed te begrijpen.

### 1.3.1 Event driven architecture

Event-driven architecture, hierna EDA genoemd, is een software architectuur pattern dat het gebruik van events promoot. Bij een systeem dat hierop gebaseerd is zal de communicatie tussen onderdelen van het systeem verlopen via events. Om EDA duidelijk uit te leggen moeten eerst ondersteunende begrippen worden toegelicht.

**Event**

Een event is een verandering van status in een onderdeel van een EDA systeem. Bij het optreden van een event wordt dit doorgegeven aan de andere onderdelen van het systeem die hierin geïnteresseerd zijn.

**Event emitters**

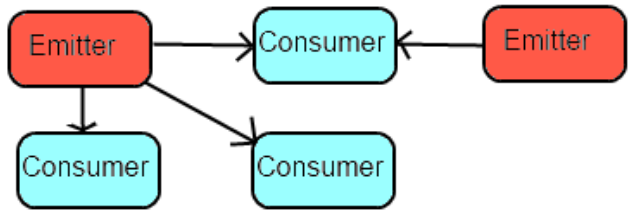
Een event emitter is een onderdeel van een EDA systeem dat events genereert en verstuurt. Als er een verandering van status optreedt in de agent dan wordt er een event gegenereerd.

**Event consumers**

Een event consumer is een onderdeel van een EDA systeem dat events ontvangt en verwerkt.

**Observer pattern**

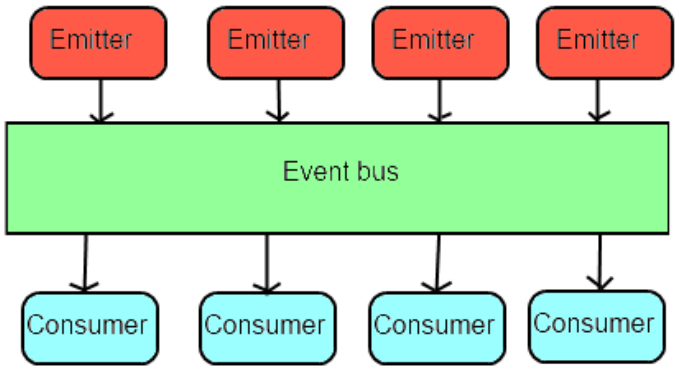
De event emitters en event consumers kunnen op verschillende manieren aan elkaar gekoppeld worden. Een mogelijkheid is gebruik te maken van het observer pattern. Hierbij meldt een event consumer dat het geïnteresseerd is in een bepaald event van een event emitter. Wanneer dit event optreedt zal de event transmitter de event consumer hiervan op de hoogte stellen.




Figuur 1: Event stroom bij het observer pattern

**Event bus**

Een andere manier om de event consumers en event emitters aan elkaar te koppelen is om gebruik te maken van een zogenoemde event bus. Hierbij laat de event consumer niet aan de event transmitter weten dat hij geïnteresseerd is in een bepaald event maar laat hij het weten aan de event bus. De event transmitter zal bij het optreden van het event de event bus hiervan op de hoogte stellen die op zijn beurt de event consumer hiervan op de hoogte stelt.




Figuur 2: Event stroom bij een Event bus

Bij event EDA is er een lage koppeling tussen de event emitters en event consumers. Dit komt omdat de componenten nooit direct met elkaar communiceren en hierdoor niet direct van elkaar op de hoogte zijn. Dit zorgt ervoor dat de verschillende onderdelen kunnen worden aangepast en vervangen zonder dat de andere onderdelen hier rekening mee hoeven te houden. Dit zorgt ervoor dat een EDA systeem gemakkelijk te onderhouden en uit te breiden is.

Een ander voordeel aan EDA is dat het heel responsief is. Bij het optreden van een event zal dit zo snel mogelijk afgehandeld worden. Het hele concept van EDA draait hier namelijk om. Om deze reden is EDA een goede oplossing voor systemen die heel responsief moeten zijn.

De losse koppeling van EDA zorgt ook voor een groot probleem. Omdat er geen directe koppeling is tussen onderdelen is het moeilijk om de event stroom in de applicatie te achterhalen. Hiermee wordt bedoeld dat het lastig is om te achterhalen welke events er optreden en welke consumers deze afhandelen. Dit zorgt ervoor dat als het systeem niet werkt zoals het hoort te werken het lastig is om te achterhalen wat er mis ging. Het is hierdoor lastig om problemen op te lossen. Ook is het hierdoor lastig om te achterhalen hoe sterk de koppeling is tussen de verschillende onderdelen in een EDA systeem.

# 2 Projectbeschrijving

In dit hoofdstuk zal de projectbeschrijving gegeven worden. Dit zal gedaan worden door eerst de aanleiding en probleemstelling voor het project te geven. Hierna zal de afstudeeropdracht gegeven worden en zullen de doelstellingen, requirements, deelopdrachten, de projectgrenzen en randvoorwaarden besproken worden. Als laatste zal de projectmethodiek behandeld worden.

## 2.1 Aanleiding & probleemstelling

Een van de bezigheden van Sogyo is het adviseren van software bedrijven over de te gebruiken tools, talen en architecturen. Sogyo ziet veel potentie in het maken van software gebaseerd op event-driven architecture. Toch hebben de klanten van Sogyo nog twijfels over het gebruik van EDA. Dit komt omdat door de losse koppeling van EDA er niet goed te achterhalen is hoe de event stroom is in een EDA applicatie. Dit zorgt ervoor dat een EDA applicatie moeilijk te debuggen is en het lastig te achterhalen is wat er aan de hand is binnen de applicatie.

Om de zorgen bij de klanten weg te nemen zal er een onderzoek naar EDA gedaan worden. Dit onderzoek moet laten zien dat het wel goed mogelijk is om EDA applicaties te ontwikkelen en te onderhouden. Om dit onderzoek te ondersteunen zal er een framework ontwikkeld worden. Dit framework moet het mogelijk maken om applicaties te ontwikkelen op basis van EDA waarbij de event stroom van de applicatie zichtbaar gemaakt kan worden.

## 2.2 Afstudeeropdracht

De afstudeeropdracht bestaat uit het ontwerpen, implementeren en documenteren van een framework. Het framework moet het mogelijk maken om applicaties te ontwikkelen op basis van EDA. Van een applicatie die gebouwd is op basis van het framework moet het mogelijk zijn om te achterhalen wat de informatie stroom is binnen in de applicatie.

Een applicatie die ontwikkeld is in het framework zal opgedeeld zijn in modules. Elke module is een onderdeel van het systeem dat events kan ontvangen. Zodra een event plaats vindt zullen de modules op de hoogte gesteld worden. De communicatie tussen de modules is de taak van het framework.

Het framework moet unit tests bevatten en moet volledig gedocumenteerd zijn. Naast de documentatie van de code moet er ook een tutorial geschreven worden hoe het framework gebruikt moet worden. De taal van het framework zal C# of F# zijn. Aan het begin van het project zal aan de hand van een vooronderzoek besloten moeten worden welke taal er gebruikt gaat worden.

## 2.3 Doelstellingen

De doelstelling van dit project is het bouwen van een framework. Het framework zal twee doelen moeten vervullen.

Het eerste doel van het framework is het onderzoek naar een EDA systeem te ondersteunen. Het onderzoek zal vooral naar voren moeten halen wat de voordelen en nadelen zijn van het bouwen van een applicatie met EDA. Het onderzoek valt buiten de scope van dit project. Desondanks is het onderzoek de primaire reden om het framework te ontwikkelen. Hierdoor moet tijdens de ontwikkeling van het framework het onderzoek altijd in het achterhoofd worden gehouden.

Het tweede doel van het framework is om de zorgen bij klanten weg te halen over EDA. Op dit moment wordt EDA niet gebruikt door de klanten omdat in een systeem dat hierop gebaseerd is de koppeling tussen onderdelen slecht te achterhalen is. Het framework moet een bijdrage leveren om te bewijzen dat deze koppelingen wel inzichtelijk te maken zijn. Dit zal gebeuren door nadat het framework gebouwd is een applicatie in het framework te ontwikkelen. Deze applicatie zal als voorbeeld dienen om te tonen dat de event stroom in beeld te brengen is bij EDA applicaties.

## 2.4 Requirements

Voor het framework zijn een aantal requirements vastgesteld door de opdrachtgever. Deze eisen moeten gevolgd worden om het project goed af te sluiten.

**1. Het framework moet de communicatie tussen modules regelen.**

De voornaamste taak van het framework is om de communicatie tussen modules te regelen. Een module is een onderdeel van het systeem dat op de hoogte van events wil worden gebracht. De communicatie tussen modules zal gebeuren met events. Modules kunnen events naar het framework sturen zodat andere modules deze events kunnen ontvangen.

**2. De events binnen het framework moeten tijdens de uitvoering van de applicatie in beeld worden gebracht.**

Om het onderzoek te steunen moet het mogelijk zijn om de events die binnen het framework plaatsvinden realtime weer te geven. Dit betekent dat tijdens de uitvoering van de applicatie de events die plaatsvinden weergegeven worden. Voor de huidige opdracht is het weergeven van de events in tekst vorm genoeg. In een vervolg opdracht zullen de events op een meer grafische manier worden weergegeven.

**3. Het moet mogelijk zijn om events te genereren.**

Het moet mogelijk zijn om events te generen. Dit zorgt ervoor dat alle event die plaats kunnen vinden ook door de gebruiker van het framework gegenereerd kunnen worden. Deze mogelijkheid zorgt ervoor dat alle modules los van elkaar getest kunnen worden.

**4. Het moet mogelijk zijn om dynamisch modules in en uit te laden.**

In het framework moet het mogelijk zijn om modules dynamisch in te laden. Hiermee wordt bedoeld dat een library-bestand wordt ingeladen tijdens de uitvoering van de applicatie. Als het library-bestand een module bevat zal deze worden aangemaakt. Hierna moet deze module direct al events kunnen ontvangen en versturen. Met het uitladen van modules wordt bedoeld dat modules verwijderd kunnen worden tijdens de uitvoering van de applicatie.

**5. Het moet mogelijk zijn dat een module een eigen front end heeft.**

Het moet mogelijk zijn dat een module een eigen front end heeft. Hiermee wordt bedoeld dat achter een module een user interface zit. Hieronder vallen niet alleen WPF en FORM interfaces maar ook web-based interfaces. Hierbij moet er rekening gehouden worden dat een module moet communiceren met processen die user interfaces aansturen.

**6. Het framework moet in de toekomst over verschillende systemen verspreid kunnen worden.**

In de toekomst moet het mogelijk zijn om het framework over verschillende systemen te verspreiden. Hiermee wordt bedoeld dat verschillende modules op verschillende systemen draaien. Dit is geen eis aan de huidige opdracht maar dit moet wel mogelijk zijn in de toekomst. Hier moet met het ontwerpen van het framework rekening gehouden worden.

**7. Het framework moet getest worden met behulp van unit tests.**

Het framework zal dienen als basis voor andere afstudeeropdrachten. Hierdoor is het van belang dat het framework volledig getest is. De code van het framework zal een testdekking van 90% moeten hebben. Dit moet getest worden met behulp van een test-coverage tool.

**8. Het framework moet volledig gedocumenteerd zijn.**

Het framework zal dienen als basis voor andere afstudeeropdrachten. Hierdoor is het van belang dat het framework volledig gedocumenteerd is.

## 2.5 Hoofdopdracht

De hoofdopdracht bestaat uit het ontwikkelen van een framework. Dit framework moet het mogelijk maken om applicaties te maken die werken met EDA. Het framework zal de communicatie tussen de verschillende onderdelen in het systeem afhandelen. Het framework zal hiernaast ook de mogelijkheid bieden om de communicatie van de verschillende onderdelen in beeld te brengen. Het framework zal aan requirements die gesteld zijn in hoofdstuk 2.4 moeten voldoen.

## 2.6 Deelopdrachten

Het project heeft naast de hoofopdracht de volgende deelopdrachten:

**1. Onderzoek naar de programmeertaal**

De eerste deelopdracht is een onderzoek naar de te gebruiken programmeertaal. Hierbij kan gekozen worden tussen C# en F#. Ik heb veel ervaring met C#, echter heb ik weinig ervaring met F#. Bij het kiezen van de taal van het framework moet hier rekening mee gehouden worden.

Bij dit onderzoek moet er ook gekeken worden naar de mogelijkheid om van het actor model te kunnen gebruiken. Actors zijn een mogelijke manier om modules te implementeren. Voordat de taal gekozen wordt moet er onderzocht worden wat actors precies zijn en of actors gebruikt gaan worden in het framework.

Het onderzoek naar de programmeertaal inclusief resultaten zal in een onderzoeksverslag verwerkt worden.

**2. Bouwen van een applicatie in het framework**

Nadat de basis van het framework gebouwd is zal er een applicatie in framework gebouwd worden. Deze applicatie zal als bewijs dienen dat het framework te gebruiken is en alle functionaliteit bevat. De ervaring die wordt opgedaan bij het bouwen van de applicatie zal gebruikt worden om het framework te verbeteren. Aangezien de applicatie alleen als bewijs dient dat het framework werkt hoeft de applicatie niet volledig werkend te zijn.

**3. Tutorial voor het gebruik van het framework**

Er moet een tutorial geschreven worden voor het gebruik van het framework. Deze tutorial moet goed beschrijven hoe het framework te gebruiken is. Deze tutorial zal waarschijnlijk in de vorm van een tekst document gemaakt worden.

## 2.7 Project grenzen & randvoorwaarden

Het framework zal dienen als basis en hulpmiddel bij een onderzoek naar EDA. Het onderzoek is geen onderdeel van dit project. Het project bevat het ontwerpen, implementeren, documenteren en testen van het framework. Ondanks dat het onderzoek buiten de scope van het project valt is het wel de belangrijkste reden voor het maken van het framework. Hierom is het belangrijk om rekening te houden met het onderzoek.

Het moet mogelijk zijn om applicaties te ontwikkelen met behulp van het framework. De verschillende onderdelen van deze applicaties zullen communiceren door middel van het framework. Omdat het framework primair dient als hulpmiddel bij het onderzoek en niet als middel om daadwerkelijk applicaties te ontwikkelen ligt de focus hier niet bij. Hierdoor hoeft er weinig tot geen rekening gehouden worden met beveiliging en efficiëntie van het framework.

Het project grenst aan minstens nog één project. Dit aangrenzende project zal de events grafisch weergeven. Om het vervolgproject te starten is er informatie nodig over de events die gegenereerd worden en welke modules er op reageren. Deze informatie moet door het framework gegeven worden. Voor het huidige project is het voldoende om de events in tekst vorm weer te geven.

Het project is geslaagd als het framework aan de eisen voldoet die in hoofdstuk 2.5 gesteld zijn en de applicatie die gemaakt is met het framework het bewijs levert dat het framework hieraan voldoet. Het framework moet ook alle onderdelen bevatten die nodig zijn zodat de vervolg stage-opdrachten begonnen kunnen worden. Hierbij is het belangrijk dat het framework de informatie geeft die nodig is om de communicatie tussen de verschillende modules in beeld te brengen.

## 2.8 Project methodiek

Bij dit project is er gebruik gemaakt worden van SCRUM als project methodiek. SCRUM is een agile project methodiek waarbij er gewerkt wordt met sprints. Een sprint is een periode van een vast aantal weken waarin een aantal doelen behaald zal worden. Er is bij dit project gewerkt worden met sprints van 2 weken.

Aan het begin van elke sprint is er bepaald worden wat er gedaan zal worden in die sprint. Dit is gedaan in de sprint planning. Elke sprint had een overkoepelend thema. Als thema's waren er bijvoorbeeld het vooronderzoek naar de programmeertaal of de het bouwen van de voorbeeld applicatie. Aan het einde van elke sprint is er gekeken of de doelen die aan het begin van de sprint vastgesteld waren behaald zijn. Ook kan er aan het einde van de sprint besloten worden om het project een andere richting in te sturen omdat er nieuwe informatie in de sprint naar voren is gekomen.

Elke sprint begon vaak met een onderzoek naar de mogelijke oplossingen voor de probleemstelling van de sprint. Hierna werd er onderzocht welke oplossing hiervan het beste was. Als laatste werd de oplossing geïmplementeerd. Als voorbeeld hiervan kan de sprint genomen worden waarin de gebruiksvriendelijkheid van het framework verbeterd werd. Eerst werd er onderzocht wat de mogelijkheden waren om dit te doen. Hierna werden de verschillende mogelijkheden naast elkaar gehouden om de beste eruit te halen. Hierna werd de oplossing geïmplementeerd.

Omdat het projectteam alleen bestaat uit een 1 stagiair zijn een aantal van de werkzaamheden die normaal bij Scrum gedaan worden achterwegen gelaten. Als voorbeeld zal er geen daily scrum gehouden worden. Dit is een meeting aan het begin van elke dag waarin wordt besproken wat elk lid van het team die dag gaat doen en of er nog problemen waren de dag ervoor. Dit is niet nodig omdat er geen andere project leden zijn die op de hoogte gesteld hoeven worden.

# 3 Project uitvoering

In dit hoofdstuk zal de uitwerking van het project beschreven worden. Dit zal gedaan worden door de verschillende onderwerpen die aan bod zijn gekomen tijdens het project te behandelen. Deze onderwerpen hebben vaak te maken met een bepaalde eis aan het framework of een deelopdracht die gemaakt is tijdens het project. De onderwerpen zullen zoveel mogelijk behandeld worden in de volgorde zoals ze tijdens het project aan bod kwamen. Dit lukt niet volledig omdat sommige onderwerpen over het gehele project verspreid waren.

### *Leeswijzer hoofdstuk 3*

**3.1** Het eerste onderwerp dat behandeld zal worden is het globale ontwerp van het framework. Hierin zal besproken worden wat het initiële ontwerp van het framework was. Er zal nog niet besproken worden hoe dit uiteindelijk uitgewerkt is, dit komt later aan bod.

**3.2** Het tweede onderwerp zal gaan over het onderzoek naar de programmeertaal van het framework. Hierbij is er ook gekeken of er bij het framework gebruik zal worden gemaakt van het Actor model.

**3.3** Het derde onderwerp zal gaan over hoe het framework uiteindelijk is gerealiseerd. Hierbij zal eerst de architectuur van het framework besproken worden. Hierna zullen ook een aantal ontwerpbeslissingen besproken worden. Als laatste zal er aan de hand van een simpel voorbeeld besproken worden hoe het framework gebruikt kan worden om een applicatie in te ontwikkelen.

**3.4** Het vierde onderwerp zal de Beheerders-interface behandelen. De Beheerders-interface is een module die gemaakt is om de werking van een applicatie gemaakt in het framework te analyseren. De Beheerders-interface kan modules in en uit laden en het laat de berichten zien die verstuurd worden in het framework. Hiernaast kunnen er berichten naar het framework verstuurd worden vanuit de Beheerders-interface.

**3.5** Het vijfde onderwerp dat besproken zal worden gaat over de applicatie die ontwikkeld is naast het framework. Hierbij zal eerst besproken worden wat de functie van de applicatie is en hoe de architectuur van de applicatie is opgezet. Hierna zullen de belangrijkste onderdelen van de applicatie besproken worden.

**3.6** Het laatste onderwerp dat besproken zal worden zal gaan over de opgeleverde documentatie en de unit tests. Onder de documentatie valt een gebruikershandleiding en de documentatie van de code van het Message-framework.

## 3.1 Globaal ontwerp van het framework

Aan het begin van het project stond het globale ontwerp van het framework al redelijk vast. Dit ontwerp was opgesteld door Ralf Wolter, de stage begeleider vanuit Sogyo. Het opgestelde ontwerp van het framework was nog niet uitgewerkt en was meer een richting van denken dan een uiteindelijk ontwerp. Tijdens de stage is dit ontwerp uitgewerkt en geïmplementeerd tot een framework. In dit gedeelte van het verslag zal het globale ontwerp van het framework besproken worden zoals hij aan het begin van de stage vast gelegd was. Hierna zal besproken worden hoe een EDA applicatie ontwikkeld kan worden met dit ontwerp van het framework.

### 3.1.1 Ontwerp framework

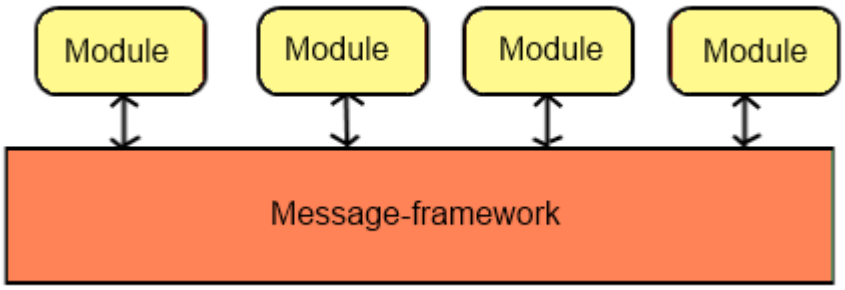
Het framework moet de communicatie regelen tussen de verschillende onderdelen van een applicatie. Deze verschillende onderdelen heten in het framework modules. De modules in het framework communiceren met elkaar met behulp van berichten die we messages zullen noemen. Omdat de communicatie in het framework volledig verloopt met behulp van messages zal het framework het Message-framework genoemd worden.

#### *Module*

Een module is een onderdeel van de applicatie met zijn eigen taak en functie. In het framewerk kennen modules elkaar niet. Elke module kent wel het framework. Communicatie tussen modules verloopt altijd door middel van het framework. Een module is geheel afgeschermd van de andere modules in het systeem. Het heeft een eigen proces en kan hierdoor geheel los van de andere onderdelen van de applicatie werken.

#### *Message*

De communicatie tussen de verschillende modules verloopt door middel van messages. Elke module heeft de mogelijkheid om messages naar het framework te versturen en te ontvangen. Na het versturen van een message wordt deze niet bij 1 specifieke module afgeleverd maar bij elke module. De modules moeten zelf bepalen of ze geïnteresseerd zijn in de message. Als dit het geval is dan zal de module de message afhandelen anders zal de module de message negeren.




Figuur 3: Het globale ontwerp van de event stroom in het Message-framework

Deze opzet maakt het mogelijk dat verschillende modules met elkaar communiceren zonder dat deze modules elkaar kennen. Er moet alleen bekend zijn welke messages verstuurd worden door een andere module zodat de module weet op welke message hij moet reageren. De enige koppeling die aanwezig is in het systeem is de koppeling tussen de modules en de messages die ze versturen en die ze willen ontvangen. Dit zorgt voor een lage koppeling en hoge cohesie voor de modules. Dit zorgt ervoor dat modules gemakkelijk aangepast en vervangen kunnen worden zonder dat de andere modules aangepast hoeven worden.

### 3.1.2 Het framework en EDA

Een van de doelen van het framework is om er EDA applicaties mee te kunnen ontwikkelen. Dit is met dit ontwerp mogelijk. Het Message-framework heeft veel weg van een event bus. In het framework stellen de modules de event emitters en de event consumers voor. Elke module kan namelijk events genereren en verwerken. De messages stellen de events voor die verstuurd worden.

Het verschil tussen een normale EDA applicatie en één die ontwikkeld is met het framework is dat een module in het framework niet luistert naar een event van een andere module maar naar een event in het algemeen. De herkomst van het event maakt voor een module niet uit in het framework terwijl bij een normale EDA applicatie de herkomst van een event vaak wel uitmaakt. Omdat de herkomst van een event niet uitmaakt is de koppeling tussen de module nog losser dan de koppeling tussen event transmitters en event consumers bij een normale EDA applicatie. Dit zorgt ervoor dat als een module wordt vervangen door een andere module die op dezelfde messages reageert en dezelfde messages verstuurt de rest van het systeem hier niet voor hoeft te worden aangepast.

Omdat alle events die gegenereerd worden altijd via het framework verstuurd worden is het mogelijk om in het framework alle events te loggen. Hierdoor is het mogelijk om de event stroom van een EDA applicatie in beeld te brengen.

## 3.2 Onderzoek programmeertaal

Nadat het globale ontwerp van het framework was vastgelegd kon er worden begonnen aan de eerste deelopdracht. Deze deelopdracht bestond uit een onderzoek waarin moest worden vastgesteld in welke programmeertaal het framework geschreven zou worden. De talen waaruit gekozen kan worden zijn C# en F#. Om deze keuze te maken is er een onderzoek gepleegd. Dit onderzoek is in een onderzoeksverslag verwerkt.

In deze scriptie zal niet het gehele onderzoeksverslag besproken worden maar de belangrijke onderdelen zullen hierin terug te vinden zijn. Eerst zal kort besproken worden tussen welke talen er gekozen kon worden en waarom. Hierna zal de onderzoeksopzet besproken worden. Hierna zal het resultaat van het onderzoek volgen.

### 3.2.1 Reden voor C# en F# als mogelijke talen

De eerste kandidaat taal C# is gekozen als mogelijke programmeertaal omdat dit mijn persoonlijke voorkeur is om in te programmeren. Dit komt omdat ik er veel ervaring mee heb en omdat de dotNET omgeving veel mogelijkheden biedt. Hiernaast werk ik graag met Visual studio.

De tweede kandidaat taal is F#. Ik had voor het project weinig ervaring met F#. Net als C# is F# een dotNET programmeertaal. F# is gekozen als mogelijke programmeertaal omdat de heer Wolter hierin interesse toonde. Dit komt omdat het met F# mogelijk is om met actors te werken. Actors tonen veel overeenkomsten met de modules in het framework en zijn hierdoor een mogelijk manier om het framework te implementeren. In het onderzoek moest bekeken worden welke voordelen het heeft om gebruik te maken van actors in het framework en of deze voordelen groot genoeg zijn om met een programmeertaal te werken waar weinig ervaring mee is.

### 3.2.2 Onderzoeksopzet

Het onderzoek is opgedeeld in drie delen.

In het eerste gedeelte van het onderzoek werd er gekeken naar de algemene voor- en nadelen van de twee programmeertalen. Hierbij is er vooral gekeken naar de verschillen tussen de twee talen en of er grote consequenties zijn aan het kiezen van een van de programmeertalen.

In het tweede gedeelte van het onderzoek is er gekeken naar de eisen die aan de programmeertaal worden gesteld. Deze eisen komen voort aan de eisen die aan het framework gesteld worden. Hierna is er gekeken of de twee talen aan deze eisen voldoen.

In het derde gedeelte van het onderzoek is er gekeken naar het actor model. De reden dat hiernaar gekeken is omdat dit mogelijk gebruikt zou worden in het framework. Het is bekend dat F# de mogelijkheid biedt om gebruik te maken van actors. Deze zitten in F# ingebouwd. In C# zitten deze niet ingebouwd. Hierom is er gekeken naar welke mogelijkheden C# heeft om met actors te werken.

Uit de informatie die voortgekomen is uit de verschillende deelonderwerpen van het onderzoek is uiteindelijk een conclusie gekomen.

### 3.2.3 Resultaten

In dit gedeelte van het verslag zullen kort de belangrijke resultaten van de deelonderzoeken besproken worden.

#### *Eerste en tweede gedeelte onderzoek*

Uit de eerste twee delen van het onderzoek is het volgende naar voren gekomen.

| | C# | F# |
|---|---|---|
| 1. Normale userinterfaces zijn mogelijk (WPF, FORMS) | Door dotNET libraries | Door dotNET libraries |
| 2. Web based user interfaces zijn mogelijk | Door dotNET libraries | Door dotNET libraries |
| 3. Bestanden kunnen ingeladen worden tijdens de uitvoering | Door dotNET libraries | Door dotNET libraries |
| 4. Klassen kunnen dynamisch worden ingeladen | Door dotNET libraries | Door dotNET libraries |
| 5. Het is mogelijk om objecten te serialiseren | Door dotNET libraries | Door dotNET libraries |
| 6. Het is mogelijk om de taal te documenteren | Mogelijk met Visual Studio | Mogelijk met Visual Studio |
| 7. Het is mogelijk om de taal te unit testen | Mogelijk met Visual Studio | Mogelijk met Visual Studio |
| 8. Cirkel dependencies zijn mogelijk | Mogelijk | Niet mogelijke door manier van compileren |
| 9. Er kan met F# projecten gewerkt worden | Omdat het tot een dotNET library gebouwd wordt | Omdat het tot een dotNET library gebouwd wordt |
| 10. Er kan met C# projecten gewerkt worden | Omdat het tot een dotNET library gebouwd wordt | Omdat het tot een dotNET library gebouwd wordt |

In deze lijst hierboven zijn de eisen verwerkt waar de taal aan moet voldoen en de andere eigenschappen waar de talen in verschillen. Zoals te kunnen F# en C# beide bijna hetzelfde. De belangrijkste ontdekkingen zullen nu worden toegelicht.

**Het maakt voor de gebruiker van het framework niet uit in welke taal het framework geschreven wordt.(9+10)**
Doordat C# en F# beide dotNET talen zijn maakt het voor de gebruiker niet uit in welke taal het framework geschreven wordt. Als een dotNET taal gecompileerd wordt dan wordt het een dotNET library. De gemaakt dotNET library kan door elke andere dotNET taal gebruikt worden. Hierdoor zal het voor de gebruiker van het framework niet uitmaken of het framework intern in C# of F# geschreven is.

**In F# zijn cirkel dependencies niet mogelijk. (7)**
Bij punt 7 is te zien dat in F# cirkel dependencies niet mogelijk zijn. Dit is ook gelijk het enige grote verschil tussen de twee talen. Met een cirkel dependency wordt bedoeld dat twee klassen naar elkaar refereren. Dit is in F# niet mogelijk. Dit komt door de manier hoe de F# code gecompileerd wordt. Het doet dit in een bepaald volgorde. Er kan geen referentie staan naar een klasse die niet gecompileerd is voordat die gerefereerd wordt. Dit is geen groot probleem omdat cirkel dependencies zo veel mogelijk voorkomen moeten worden. Als hiermee bij het ontwerp van het framework rekening mee wordt gehouden dan zou dit niet voor problemen moeten zorgen.

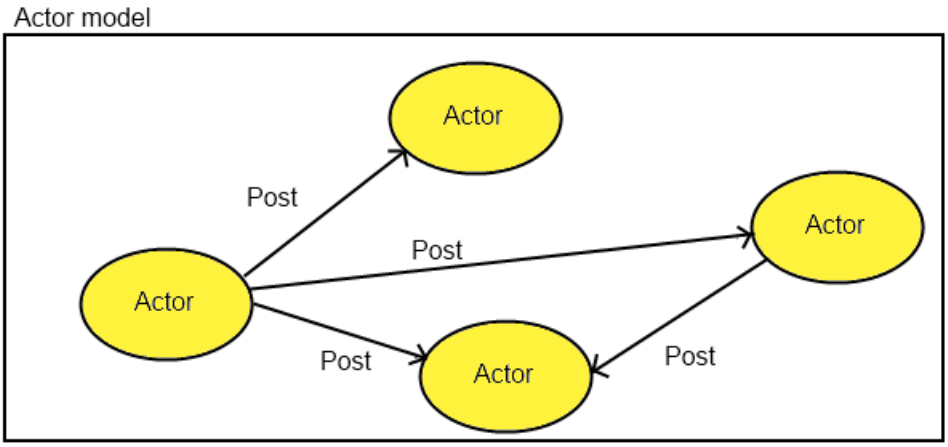
Uit de eerste twee gedeeltes van het onderzoek zijn geen belangrijke feiten naar voren gekomen die de een of de andere taal voorkeur geven.

## *Derde gedeelte onderzoek*

In het derde gedeelte van het onderzoek is er onderzocht wat het actor model inhoudt en of dit gebruikt zou worden voor het framework. Hierna is er gekeken wat de mogelijkheden zijn van het gebruik van het actor model in F# en C#.

**Wat is het actor model?**

Het actor model is een computer model dat stelt dat alles in het systeem een actor is. Het kan vergeleken worden met het object georiënteerd model waarin het systeem volledig bestaat uit objecten. Een actor is een processing entiteit die berichten van andere actoren kan afhandelen. De enige taak van een actor is berichten ontvangen en afhandelen. Voor het versturen van een bericht naar een actor moet een actor het adres weten van de andere actor.




Figuur 4: Een voorbeeld van een Actor model systeem

Bij het ontvangen van een bericht kan een actor de volgende dingen doen:

1. Een of meerdere berichten sturen naar andere actors.
2. Een of meerdere actors creëren.
3. De afhandeling van de volgende message die het ontvangt bepalen.

Over de afhandeling van berichten kan het volgende gezegd worden:

1. Als een bericht binnen komt tijdens de uitvoering van een ander bericht dan wordt het nieuwe bericht op de stack van de actor gezet. Deze wordt afgehandeld zodra de actor hier tijd voor heeft.
2. Het FIFO principe geld voor de afhandeling van berichten op de stack. De berichten worden afgehandeld in de volgorde waarin ze binnen komen.
3. Als een actor geen berichten afhandelt dan doet de actor helemaal niks. Dit betekent dat een actor geen processor kracht verspilt aan het wachten op berichten.

Een groot voordeel aan het actor model is dat het systeem verdeeld wordt in verschillende onderdelen die elk eigen stukken code kunnen uitvoeren. Dit zorgt ervoor dat een systeem dat werkt met actors automatisch parallel wordt uitgevoerd. Aangezien een actor geen processor kracht kost als hij aan het wachten is op berichten maakt het niet uit als er veel actors in het systeem aanwezig zijn.

(Hewitt, 2012)

**Het actor model en het framework**
Er is overwogen om het actor model te gebruiken in het framework omdat een module in het framework veel overeenkomsten heeft met een actor uit het actor model. De overeenkomsten zijn als volgt:

1. Beide zijn losse processing entiteiten.
2. Ze reageren alleen op berichten van buiten af.

Het grote verschil tussen een actor en een module is dat een actor berichten stuurt naar andere actors. Bij het framework stuur een module naar het framework. Bij het framework worden berichten dus niet verstuurd naar 1 specifieke module. Om deze reden is het actor model niet interessant voor het framework. Het concept van een actor echter wel. Als een taal actors ingebouwd heeft kunnen deze gebruikt worden om de modules te bouwen. Hierdoor zou de module automatisch multi-threaded zijn zonder dat dit zelf gebouwd hoeft te worden. De berichten worden automatisch 1 voor 1 afgehandeld en berichten worden automatisch op een stack gezet. Een andere groot voordeel is dat een actor geen processor kracht kost als het aan het wachten is op berichten.

Een van de problemen die wordt opgelost bij het gebruik maken van een actor klasse is dat er geen manier meer hoeft worden bedacht hoe een module berichten ontvangt van het framework. Een module moet een losse processing entiteit zijn en bezit hierdoor een eigen thread. De berichten van het framework komen van een andere thread af en hierdoor moet er tussen twee threads gecommuniceerd worden. Communicatie tussen threads is zeer beperkt mogelijk en zorgt bijna altijd voor problemen. Een mogelijke oplossing is bijvoorbeeld dat de thread van de module de hele tijd kijkt of er berichten zijn die hij moet afhandelen. Het nadeel aan deze methode is dat de module ook als hij niks doet processorkracht kost omdat hij controleert of er berichten zijn die hij moet afhandelen. Als er gebruikt wordt gemaakt van een actor klasse kost het wachten op berichten geen processorkracht.

Door gebruik te maken van een ingebouwde actor klasse om de module te implementeren wordt er veel werk uit handen genomen. Hierom zal er in het framework gebruik gemaakt worden van actors. Nu er besloten is dat er gebruik gaat worden van actors moet er bekeken worden wat de mogelijkheden zijn van de twee talen voor het gebruik van de actors.

**Wat zijn de mogelijkheden in F# voor actors**
In F# zitten actors in de taal zelf ingebouwd. Actors heten in F# mailboxen maar het principe erachter is hetzelfde. Actors in F# kunnen alles wat een actor moet doen. Elke mailbox heeft een adres waarna je berichten kan sturen. Dit gebeurt door de Post methode te gebruiken. Hieraan kunnen variabels worden meegeven.

```
let mailbox = new MailboxProcessor<Message>(fun inbox ->
    let rec loop count =
        async { printfn "Message count = %d. Waiting for next message." count
                let! msg = inbox.Receive()
                printfn "Message received. ID: %d Contents: %s" msg.ID msg.Contents
                return! loop( count + 1) }
    loop 0)
```

Figuur 5: Een nieuwe actor wordt aangemaakt

In de code hierboven wordt een simpele actor aangemaakt. Het is niet belangrijk om de code te begrijpen. Wel is belangrijk om te zien dat een actor met zeer weinig code aangemaakt kan worden. Hierbij is er gelijk voor gezorgd dat de actor een eigen thread heeft die geen processing kracht kost als het niks doet. Berichten die binnen komen tijdens de afhandeling van een bericht worden automatisch op een stack gezet en zo snel mogelijk afgehandeld. Hiernaast hoeft er geen code geschreven worden om de communicatie tussen threads te regelen. Een andere thread kan aan de actor door middel van het Post commando vragen of de actor het bericht wil afhandelen.

**Wat zijn de mogelijkheden in C# voor actors**
Er is ook onderzocht wat de mogelijkheden zijn van actors in C#. Uit dit onderzoek kwam naar voren dat het niet mogelijk was om actors te gebruiken zonder een externe library te gebruiken. Er zijn naar verschillende externe libraries gekeken maar deze waren vaak niet compleet of hadden andere nadelen. Als voorbeeld is er gekeken naar Concurrency and Coordination Runtime library van Microsoft. Om deze library te gebruiken moet de gehele Robotics runtime environment geïnstalleerd worden. De gebruiker van het framework zou deze ook moeten installeren. Dit zou het framework veel minder gebruiksvriendelijk maken.

Hierna is er gekeken naar een laatste optie namelijk zelf een actor library schrijven in F# en die te gebruiken in C# code. Hierdoor kan het framework geschreven worden in C# terwijl er wel gebruik wordt gemaakt van de F# actor klasse. Hiervoor is ook niet gekozen omdat dit voor het volgende probleem zorgde. Omdat er in F# geen cirkel dependencies kunnen zijn kan de F# library niet de C# library kennen. Dit is wel nodig om de twee goed te laten communiceren. Om dit toch mogelijk te maken moet er een derde project worden aangemaakt waar ze beiden naar refereren zodat de twee projecten met elkaar kunnen communiceren. Dit zou betekenen dat de gebruiker van het framework 3 libraries moet importeren om het framework te gebruiken. Hiernaast is het slecht te overzien wat voor andere problemen dit later in het project oplevert.

Uiteindelijk is er uit het onderzoek gekomen dat het niet goed mogelijk is om actors te gebruiken in C#. Hierdoor valt C# eigenlijk af als programmeertaal voor het framework.

### 3.2.4 Conclusie onderzoek
Uit de eerste twee delen van het onderzoek kwam naar voren dat beide programmeertalen veel gelijke mogelijkheden hebben en dat er geen grote nadelen kleefden aan het gebruik van de een of de andere programmeertaal. Het derde gedeelte van het onderzoek liet zien dat door gebruik te maken van actors er veel problemen opgelost worden voordat ze daadwerkelijk aan bod komen. Ook kwam er naar voren dat actors in F# goed gebruikt konden worden en in C# niet. Om deze reden is er gekozen om F# te gebruiken als programmeertaal voor het Message-framework.

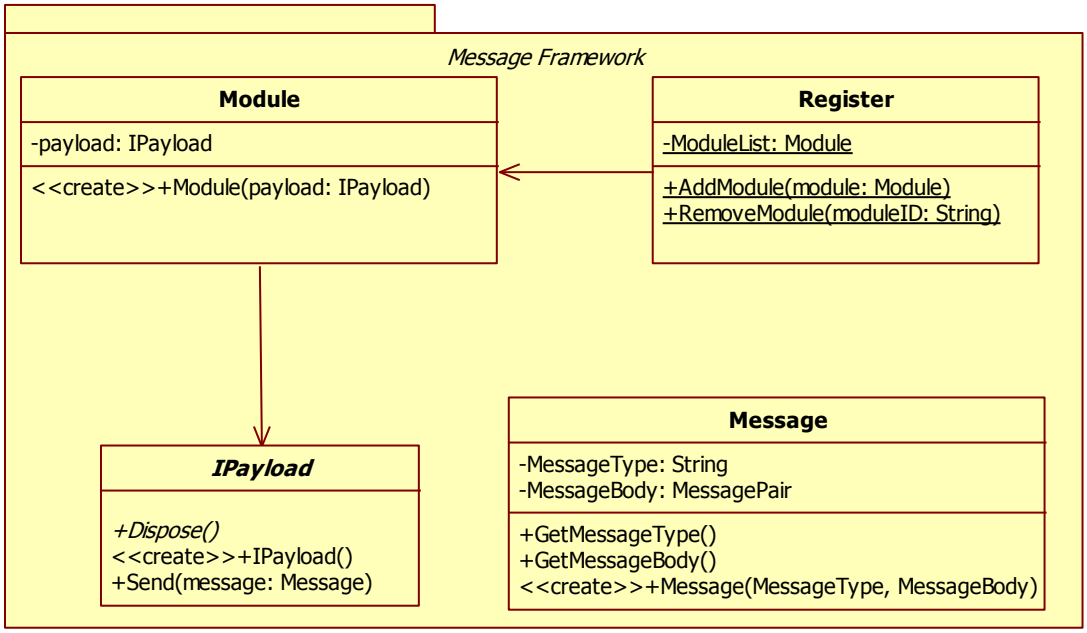
## 3.3 Het Message-framework

Dit gedeelte van het verslag zal gaan over het gebouwde Message-framework. Als eerste zal de architectuur van het framework besproken worden. Hierin zal gekeken worden naar hoe het systeem er nu uit ziet. Hierna zullen een aantal ontwerp beslissing besproken worden die tijdens het bouwen van het framework aan bod zijn gekomen. Als laatste zal er een kort voorbeeld gegeven worden van een simpele applicatie die ontwikkeld is in het framework.

### 3.3.1 Architectuur van het framework

Aan het begin van het project stond de globale werking van het framework al vast maar de interne architectuur van het Message-framework moest nog vastgelegd worden.

-



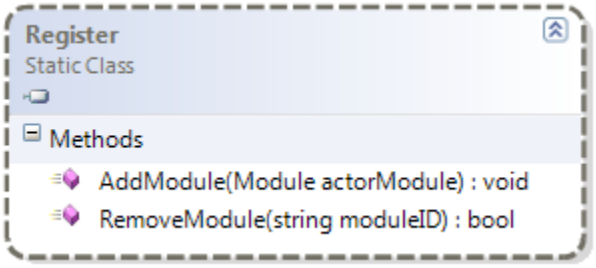

Figuur 6: De belangrijkste klasse van het Message-framework

De klassen in de afbeelding hierboven zijn de klassen die het hart van het framework vormen. Deze 4 klassen samen zorgen samen voor het overgrote deel van de functionaliteit van het framework. Deze klassen komen op de volgende manier overeen met het concept van het framework:

1. De module wordt vertegenwoordigd door de Module en IPayload klasse
2. De message wordt vertegenwoordigd door de Message klasse.
3. De taak van de Register is om alle modules in het framework bij te houden.

Er zal een meer uitgebreide uitleg gegeven worden bij elk van deze 4 klassen. Hierbij zullen vooral de functionaliteit van de klasse besproken worden. Het is hierbij van belang om te zeggen dat in dit verslag als er Module of Message met een hoofdletter staat hiermee de klasse wordt bedoeld. Als er module of message met een kleine letter staat wordt er hiermee het concept bedoeld zoals in het globale ontwerp is opgenomen.

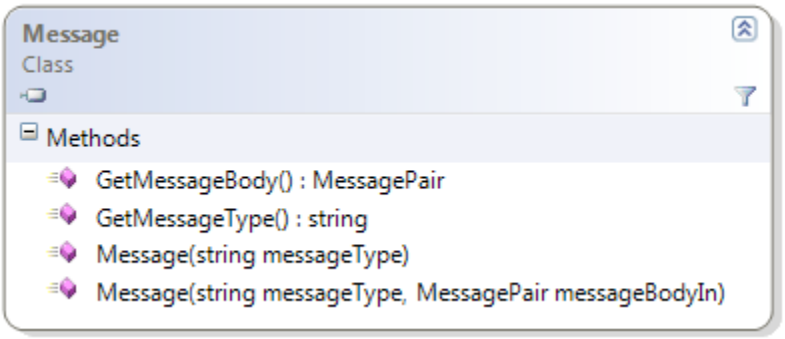
### *Register:*




Figuur 7: De Register statische klasse

De statische klasse Register heeft één taak. De taak is het om alle modules bij te houden die bestaan in het framework. Elke module die wordt aangemaakt moet geregistreerd worden bij de Register. Pas nadat een module zich bij de Register heeft geregistreerd zal de module messages kunnen versturen en ontvangen. Als een module niet bij de Register is toegevoegd kan de module geen messages versturen en ontvangen. Een module kan aan de Register worden toegevoegd door de AddModule operatie te gebruiken. Een module kan uit de Register verwijderd worden door de RemoveModule aan te roepen. Omdat de Register een statische klasse is kan het vanaf elke plaats in de code worden aangeroepen.

### *Message:*




Figuur 8: De Message klasse

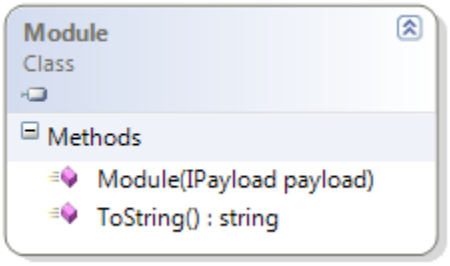
De Message klasse stelt de message voor die tussen de verschillende modules in het framework verstuurd kunnen worden. De Message klasse heeft twee taken. De eerste taak is om aan te geven om wat voor soort Message het gaat. De tweede taak van de Message klasse is om data door te geven tussen Modules. Hiervoor worden de volgende twee onderdelen van de Message gebruikt:

1. Message type
2. Message body

De Message type geeft aan om wat voor type message het gaat. Dit wordt weergegeven met een String. De Message type is het onderdeel van de Message dat wordt gebruikt om te bepalen of een module een Message moet afhandelen of niet.

De Message body is het onderdeel van de Message waarmee data doorgegeven kan worden met de Message. De invulling van de Message body zal in het gedeelte over de ontwerpbeslissingen besproken worden. Hierbij zal de implementatie en de redenering achter de implementatie besproken worden.

### *Module:*



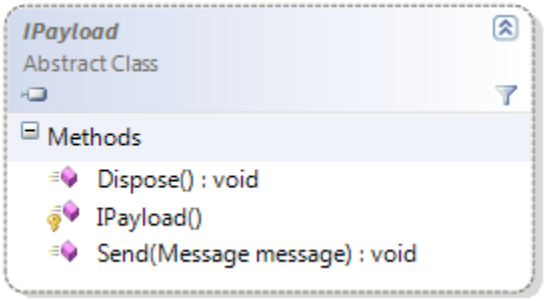

Figuur 9: De Module klasse

De Module klasse vertegenwoordigt samen met de IPayload een module in het framework. Voor elke module die er in de applicatie is moet er een Module worden aangemaakt. Om een Module aan te maken moet een implementatie van de IPayload interface worden meegegeven.

De taak van de Module klasse is om te bepalen of een Message die wordt verstuurd naar het framework afgehandeld moet worden door de module. Als dit niet het geval is zal de Message worden genegeerd. De Module kan zien of een Message interessant is voor de module door te kijken naar het type van de Message. In de eerste opzet van het framework moest er door middel van de methode AddModuleToListenTo aangegeven worden dat de Module naar een bepaalde Message moest luisteren. In een latere versie van het framework werd deze informatie in de implementatie van de IPayload geplaatst. Hierna kan de module door middel van de Reflectie library van dotNET deze informatie uit de IPayload halen. Dit wordt later bij de ontwerpbeslissingen verder behandeld.

De Module klasse maakt intern gebruik van de Actor klasse van F#. Zodra een Message bij de Module binnenkomt die de Module moet afhandelen wordt de Actor gebruikt om de Message af te handelen. Aan de Post methode van de actor wordt de methode die het moet uitvoeren om de Message af te handelen meegegeven. De Actor zal hierna de methode uitvoeren zodra hij hier tijd voor heeft. Door deze aanpak wordt er gegarandeerd dat er altijd maar één Message tegelijk wordt afgehandeld en worden Messages die binnen komen tijdens de afhandeling van een andere Message op de wachtrij gezet.

### *IPayload:*




Figuur 10: De IPayload interface

De IPayload is een interface die geïmplementeerd moet worden om een Module aan te maken. Een implementatie van de IPayload zullen we een Payload noemen. Met de Payload kunnen de volgende dingen gedaan worden:

1. Er kan gedefinieerd worden naar welke Messages er geluisterd moet worden door de Module en wat er moet gebeuren als die Message binnenkomt.
2. De naam van de Module kan gedefinieerd worden.
3. Het initialiseren en de-initialiseren van de onderdelen van de Module kan worden gedefinieerd.
4. De Payload kan Messages verzenden naar het framework.

Bij het implementeren van de IPayload definieert de gebruiker het gedrag van de bijbehorende module.

```
[ModuleName("ExampleModule")]
public class ExamplePayload : IPayload
{
    public ExamplePayload()
    {
        //Code to Start the parts of the module
    }

    public override void Receive(Message message)
    {
        //Code to Receive and handle the Message
    }

    public override void Dispose()
    {
        //Code to Dispose the parts of the Module
    }
}
```

Figuur 11: Een voorbeeld van een implementatie van de IPayload klasse.

**Definiëren van het luisteren naar Message**

Bij de eerste opzet van het framework werd er door de gebruiker bepaald hoe een Message werd afgehandeld door de Receive methode van de IPayload te implementeren. De Receive methode werd aangeroepen elke keer dat er een Message naar het framework werd verstuurd. De gebruiker moest zelf in de Receive functie de Message type controleren of deze overeenkomt met het type van een Message die hij wil ontvangen. Later in dit hoofdstuk wordt bij de ontwerpbeslissingen besproken hoe dit verbeterd is en waarom dit nodig was om te verbeteren.

**Definiëren van de naam van een Module**

Het is mogelijk om de naam van de module te definiëren in de Payload. Dit kan gedaan worden door gebruik te maken van een Attribute. Een Attribute is een onderdeel van het dotNET framework dat gebruikt kan worden om informatie in een klasse of methode te zetten. In het voorbeeld hierboven is een simpele implementatie van de IPayload te zien. Hierbij wordt de naam gezet boven het aanmaken van de klasse. De Module klasse kan hierna door middel van Reflection library van dotNET de naam van de module uit de Payload halen. De Reflection library kan gebruikt worden om de informatie van een klasse op te vragen. Onder deze informatie vallen ook de attributen die bij een klasse horen. Hierdoor kan de naam die bij de ModuleName attribute gezet wordt door de Module klasse opgehaald worden. De naam van de Module wordt gebruikt om te bekijken vanaf welke Module een bericht komt. Hiernaast wordt het gebruikt om Modules te verwijderen uit het framework.

**Definiëren van het initialiseren en de-initialiseren van de onderdelen van een Module**

Het definiëren van het initialiseren en de-initialiseren van de onderdelen van een Module is ook de taak van de Payload. Met de onderdelen van een Module worden alle klasse bedoeld die vallen onder de module. Het aanmaken van deze onderdelen moet worden gedaan in de constructor van de Payload. Dit zorgt ervoor dat de Module verantwoordelijk is voor zijn eigen resources. De Dispose methode van de Payload moet worden gedefinieerd om de resources van de module weer los te laten. In de Dispose moeten alle resources van de Payload worden vrijgegeven. De Dispose wordt aangeroepen door het framework zodra een Module wordt verwijderd.

**Het versturen van Messages**

Als laatste verantwoordelijkheid is de Payload ook de klasse vanwaar er Messages naar het framework kunnen worden verstuurd. Dit kan alleen vanuit de Payload omdat de Payload het enige is wat bekend is voor de code van een module. Hiernaast is het altijd nodig dat een Message verstuurd wordt vanuit een onderdeel van een module omdat er anders niet gezien kan worden hoe de berichten verlopen tussen de modules.

De IPayload heeft een Send operatie waarmee Messages verstuurd kunnen worden. Deze kan gebruikt worden maar het is aan te raden om gebruik te maken van de MessageCreator en de Send operatie daarvan te gebruiken. Hierbij wordt intern de Send van de IPayload aangeroepen. De MessageCreator wordt later in het hoofdstuk besproken.

### 3.3.2 Ontwerpbeslissingen

In dit gedeelte van het verslag zullen een aantal ontwerpbeslissingen besproken worden die te maken hebben met het framework. Deze ontwerpbeslissingen zijn genomen in de sprints nadat de eerste versie van het framework opgezet was.

De volgende ontwerpbeslissingen zullen besproken worden.

1. De invulling van de Message body.
2. Gebruiksvriendelijker maken van het aanmaken van een Message.
3. Gebruiksvriendelijker maken van het ontvangen van de Message.
4. De meta informatie geven over het framework.

De ontwerpbeslissingen zullen allemaal op dezelfde manier besproken worden. Eerst zal het probleem besproken worden. Hierna zullen de mogelijkheden die onderzocht zijn kort toegelicht worden. Als laatste zal de oplossing besproken worden.

#### *De invulling van de Message body*

**Het probleem:**

Nadat het framework globaal was opgezet voldeed het voor een groot deel aan de eisen aan het framework. Een van de functionaliteiten die er nog miste was dat er geen informatie aan een message meegegeven kon worden. Dit was wel nodig omdat modules anders niet goed kunnen communiceren. Als voorbeeld kan er gedacht worden aan een module die een persoon object bezit. Er is een nieuwe persoon aangemaakt en de module stuurt hiervoor een event message naar het framework dat er een persoon is aangemaakt. De andere modules kunnen hier niets mee tenzij de message ook daadwerkelijk de data van de persoon bevat. Om deze reden is de Message body ontwikkeld.

Bij het maken van de Message body waren er twee eisen:

1. De informatie moet geserialiseerd kunnen worden.

De eerste eis kwam naar voren vanuit een van de eisen aan het framework. Eis nummer 6 aan het framework stelt dat het framework later over verschillende systemen verspreid moet kunnen worden. Ondanks dat dit niet geïmplementeerd hoeft te worden in dit project moet dit wel mogelijk blijven. Dit zal later waarschijnlijk geïmplementeerd worden door de Messages die verstuurd worden te serialiseren. Hierdoor kunnen de Messages verstuurd worden tussen verschillende systemen. De beperking die naar voren komt uit deze eis is dat de gehele Message klasse geserialiseerd moet kunnen worden. Dit betekent dat de data in de Message body ook geserialiseerd moet kunnen worden. Om dit mogelijk te maken moet de data in de Message body alleen bestaan uit primitieve types. Als dit gegarandeerd wordt dan kan er ook gegarandeerd worden dat de Message in de toekomst geserialiseerd kan worden.

2. De ontvangende module moet weten wat de data voorstelt

De tweede eis aan de Message body is dat de ontvangende module moet kunnen zien wat de data die verstuurd is voorstelt. Als er van de data die verstuurd is niet duidelijk is wat er mee bedoeld wordt kan de ontvangende module er niet mee werken.

**De mogelijkheden:**
Om de Message body te implementeren zijn er een aantal mogelijke oplossingen onderzocht.

De eerste oplossing die bekeken is bestond uit het direct meegeven van objecten aan de Message body. Bij deze oplossing moest het object een DTO object zijn. Een DTO object ofwel een Data transfer object is een object dat gebruikt wordt om data tussen twee klasse door te geven. De twee modules moesten dan alle twee het DTO object kennen. Deze oplossing is niet valide omdat er een koppeling komt tussen modules doormiddel het DTO object. Hiernaast moet de module iets kennen wat buiten de module valt. Hierdoor is de cohesie van de module ook lager.

De tweede oplossing die bekeken is bestond uit het meegeven van primitieve types. De Message body zal dan bestaan uit een lijst van primitieve types. Elk van deze types geeft een waarde aan die doorgegeven moet worden. De volgorde zoals ze in de Message body staan zou dan bepalen wat ze voorstellen. Hierbij moet de ontvangende module dus precies weten in welke volgorde de data de Message body staat. Hierbij moet er dus goed afgesproken worden hoe de Message eruit ziet. Deze oplossing is ook niet ideaal omdat de koppeling tussen de Modules en de Message die ze sturen heel erg versterkt wordt.

De derde oplossing die bekeken is was gebruik maken van key-value paren. Een key-value paar is een paar van twee waardes die bij elkaar horen. De key geeft aan wat het paar voorstelt. De value geeft de waarde aan van het paar. Door gebruik te maken van key-value paren kan de ontvangende module aan de key zien wat er bedoeld word met het paar en de value geeft de werkelijke waarde van het paar. Als er dan ook nog gegarandeerd wordt dat de key-value paren alleen van primitieve types kunnen zijn dan kan er gegarandeerd worden dat de Message body volledig geserialiseerd kan worden. Omdat de oplossing van key-value paren volledig voldoet aan de gestelde eisen is deze oplossing gekozen om uit te werken.

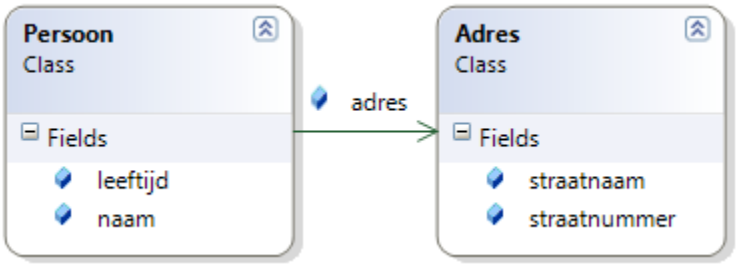
**De oplossing:**
Zoals al besproken is er gekozen om de Message body te implementeren als een lijst van key-value paren. De uiteindelijke implementatie van de Message body is in de vorm van een lijst van MessagePairs. De MessagePair klasse is een speciale klasse geschreven voor de Message body.

```
type public MessagePair =
    | StringValue of (string * string)
    | IntValue of (string * int)
    | DoubleValue of (string * double)
    | BoolValue of (string * bool)
    | MessagePairList of (string * List<MessagePair>)
```

Figuur 12: Een gedeelte van de code van de MessagePair klasse

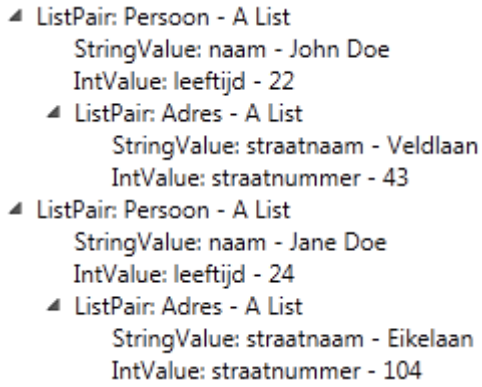
De MessagePair klasse is van het zogenoemde Discriminated Union type. Een Discriminated Union is een klasse die intern verschillende klasse kan zijn. In het voorbeeld hierboven is te zien dat de MessagePair klasse van de type StringValue, IntValue, DoubleValue, BoolValue en MessagePairList kan zijn. Dit zijn allemaal MessagePairs en bij het aanmaken van een MessagePair kan er gekozen worden van welk type het moet zijn. Elk van deze interne MessagePair objecten bestaat uit een combinatie tussen een key en een value. De key is altijd een string en deze geeft aan wat er bedoeld wordt met de MessagePair. De value verschilt per intern type. De value kan bij een MessagePair van het type string, int, double of boolean zijn. Dit zijn allemaal primitieve types. Hierdoor wordt er gegarandeerd dat de MessageBody geserialiseerd kan worden.

De MessagePair kan ook nog van een extra type zijn genaamd de MessagePairList. Dit is ook een combinatie van een key en een value. Hierbij is de value een lijst van MessagePairs. Dit zorgt ervoor dat een lijst van MessagePairs intern nog een lijst van MessagePairs kan bevatten.




Figuur 13: De Persoon klasse

Dit is een handige oplossing als er bijvoorbeeld de data van een persoon moet worden doorgegeven met de Message body. De persoon bevat een naam, een leeftijd en een adres. Het adres bevat op zijn beurt weer een straatnaam en een straatnummer. Om deze data mee te geven aan de Message body is het nodig dat de data van de Persoon bij elkaar blijft. Dit is nodig omdat het mogelijk moet zijn meerdere Persoon objecten mee te geven aan de Message body. Als deze niet bij elkaar blijven dan kan er niet goed gezien worden welk Adres bij welke Persoon hoort.




Figuur 14: De Message body van een voorbeeld Message

In het voorbeeld hierboven is de Message body te zien van een Message die twee Persoon objecten bevat. De data van de Persoon objecten zit in een interne MessagePair waardoor er goed te zien is welke data bij welk object hoort. De structuur van het werkelijke object kan gevolgd worden in de Message body. Ook kan de data van meerdere objecten worden meegegeven van hetzelfde type zonder dat de data door elkaar komt te staan.

### *Gebruiksvriendelijker maken van het aanmaken van een Message*

**Het probleem**

Het volgende probleem dat naar voren kwam was dat het aanmaken van een nieuwe Message veel code koste en heel onoverzichtelijk was. Dit kwam vooral omdat de gebruiker weet moest hebben van de interne werking van de Message klasse. Dit is nodig omdat er anders geen data aan de Message body kon worden toegevoegd. Als voorbeeld volgt hier de code om een simpele Message aan te maken te versturen.

```
MessagePair messageBody = MessagePair.NewMessagePairList(
    new Tuple<string, List<MessagePair>>("Message body", new List<MessagePair>()));
MessagePair persoonPair = MessagePair.NewMessagePairList(
    new Tuple<string, List<MessagePair>>("Persoon", new List<MessagePair>()));
MessagePair naamPair = MessagePair.NewStringValue(
    new Tuple<string, string>("naam", persoon.naam));
MessagePair leeftijdPair = MessagePair.NewIntValue(
    new Tuple<string, int>("leeftijd", persoon.leeftijd));
persoonPair.AddMessagePair(naamPair);
persoonPair.AddMessagePair(leeftijdPair);
messageBody.AddMessagePair(persoonPair);
Message message = new Message("newPersoonEvent", messageBody);
this.Send(message);
```

Figuur 15: Oude code voor het aanmaken en versturen van een Message

Om de Message aan te maken is er kennis nodig van de interne werking van de MessagePair klasse. Er is ook veel code nodig om weinig te doen. In de code hierboven wordt slechts een Message aangemaakt met de data van een Persoon erin. Hierin wordt alleen nog maar de naam en leeftijd meegegeven. De complexiteit van het aanmaken van een message doet de gebruiksvriendelijkheid van het framework geen goed. Hierom is dit versimpeld.

**De mogelijkheden**

Er zijn verschillende oplossingen onderzocht om het aanmaken van de Message te versimpelen.

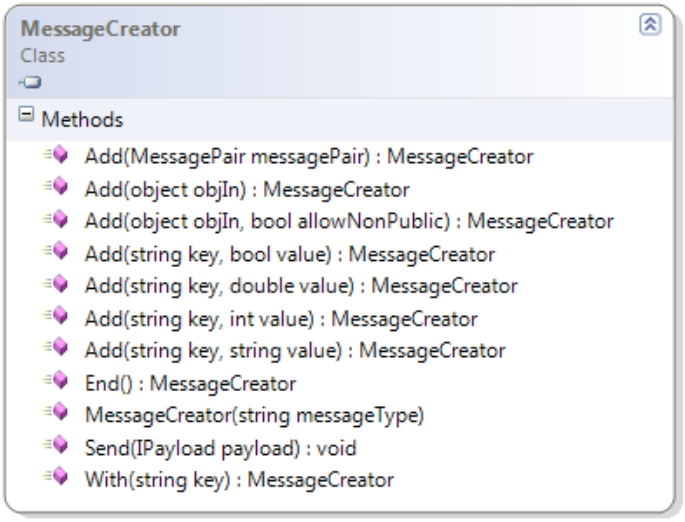
De eerste oplossing die bekeken is was om de Message body anders te implementeren dan met de MessagePair klasse. Dit is niet gekozen om te doen omdat de MessagePair klasse juist naar voren was gekomen als de invulling van de Message body na een onderzoek hiernaar.

Hierna was er gekeken naar een klasse die zou werken als tussenstap. Deze klasse zou de Message geheel afschermen van de gebruiker. Hierdoor zou het maken van een Message al een stuk makkelijker worden. Initieel is dit gedaan. In een later iteratie is hier nog een verbetering op gedaan. Namelijk door gebruik te maken van een fluent interface.

**De oplossing**

Om het aanmaken van een Message makkelijker te maken moest de Message van de gebruiker worden afgeschermd. Hiervoor is de MessageCreator klasse aangemaakt.

De eerste functionaliteit van de MessageCreator klasse is dat de Message klasse van de gebruiker afgeschermd wordt. Hierdoor hoeft de gebruiker niet te weten hoe de Message intern werkt. Met de Add operatie kunnen paren worden toegevoegd aan de MessageCreator en met de Send operatie kan de aangemaakte Message verstuurd worden. Hiervoor hoeven de Message klasse en de MessagePair klasse nooit gebruikt worden door de gebruiker. Deze worden intern wel gebruikt door de MessageCreator klasse maar de gebruiker hoeft geen weet te hebben hoe deze werken.




Figuur 16: De MessageCreator klasse

De tweede kracht van de MessageCreator klasse is dat het gebruik maakt van fluent interface. Fluent interface is een techniek om gemakkelijk objecten aan te maken met data erin. Het idee erachter is dat na elke methode een instantie van de klasse zelf wordt teruggegeven. Hierdoor kan er na elke operatie gelijk weer een operatie op de klasse aangeroepen worden. In het geval van de MessageCreator kan de Add methode keer op keer na elkaar worden aangeroepen.  Als laatste operatie wordt de Send van de operatie aangeroepen en wordt de gemaakte Message verzonden.

```
new MessageCreator("NewPersoonEvent")
    .With("Persoon")
        .Add("naam", persoon.naam)
        .Add("leeftijd", persoon.leeftijd)
    .End()
    .Send(this);
```

Figuur 17: Code voor het aanmaken en versturen van een Message met de MessageCreator

In het voorbeeld hierboven wordt dezelfde Message aangemaakt als aan begin alleen wordt hier gebruik gemaakt van de MessageCreator klasse. Er is veel minder code nodig om te hetzelfde gedaan te krijgen. Hiernaast is de code veel leesbaarder. Met de With operatie is het mogelijk om een intern paar aan te maken. Nadat de With operatie is aangeroepen zullen de Add operaties van de MessageCreator paren toevoegen aan het interne paar. Dit paar kan worden afgesloten met de End operatie. Hierna zal de Add operatie weer paren toevoegen aan de Message body. Met de MessageCreator klasse is er niet alleen veel minder code nodig om hetzelfde te doen maar de code is ook nog een stuk leesbaarder.

```
new MessageCreator("NewPersoonEvent")
    .Add(persoon)
    .Send(this);
```

Figuur 18: Een Message wordt verstuurd met alle public variables van het persoon object

Bij een latere iteratie is een extra functie aan de MessageCreator toegevoegd. Dit is de Add functie waarmee een object kan worden meegegeven. Deze operatie haalt alle public variables uit het object en zet deze als paren in de Message body. Dit zorgt ervoor dat de data van een object gemakkelijk aan een Message kan worden toegevoegd. Het is ook mogelijk om aan te geven dat private variables meegenomen moeten worden in de Message body. Dit is mogelijk omdat er gebruik wordt gemaakt van de Reflection library van dotNET.

### *Het gebruiksvriendelijker maken van het ontvangen van de Message*

**Het probleem**

Net als bij het aanmaken van een Message was het definiëren van het ontvangen van Messages een groot karwei voor de gebruiker van het framework. Bij het eerste ontwerp van het framework ontving de Payload alle messages. Dit ontving hij in de Receive methode van de Payload. Deze moest door de gebruiker worden geïmplementeerd in de Payload. De gebruiker moest hierna zelf in een if statement bepalen of hij het bericht wou afhandelen. Dit zorgde ervoor de Receive methode al snel groot werd. Hiernaast moest de gebruiker zelf de data uit de Message body halen. Dit deed hij door de Message body op te vragen aan de Message. Hierna moest de gebruiker controleren of de Message body de paren bezit die hij nodig heeft. Hierna moesten deze paren er ook nog uit gehaald worden. Hierna kon de gebruiker pas werkelijk iets doen met de code.

```
public void Receive(Message message)
{
    if (message.GetMessageType() == "ExpectedMessage")
    {
        MessagePair messageBody = message.GetMessageBody();
        MessagePair personPair = MessagePair.NewMessagePairList(new Tuple<string,List<MessagePair>>("Persoon", new List<MessagePair>()));
        personPair.AddMessagePair(MessagePair.NewStringValue(new Tuple<string,string>("naam", "")));
        personPair.AddMessagePair(MessagePair.NewIntValue(new Tuple<string,int>("leeftijd", 0)));
        //Check if the Message body contains the expected MessagePairs
        if (messageBody.GotExpectedKeysOfType(personPair))
        {
            //Get the values out of the Message body
            personPair = messageBody.GetMessagePair(personPair);
            String name = personPair.GetMessagePair(MessagePair.NewStringValue(new Tuple<string, string>("naam", ""))).GetString();
            int age = personPair.GetMessagePair(MessagePair.NewIntValue(new Tuple<string, int>("leeftijd", 0))).GetInt();
            //Code to do something with the age and name
        }
    }
    else if (message.GetMessageType() == "ExpectedMessage2")
    {
        //Code for handeling ExpectedMessage 2
    }
}
```

Figuur 19: Code van de Receive methode van een oude IPayload implementatie

De code hierboven is een voorbeeld van een Receive methode van een IPayload implementatie. In de code wordt de ExpectedMessage afgehandeld. Hierbij wordt als eerste gecontroleerd of de Message type van de Message ExpectedMessage is. Als dit zo is dan wordt de Message body gecontroleerd of deze de data bevat die nodig is om de message af te handelen. Als laatste wordt de data daadwerkelijk uit de Message body gehaald. Ook de ExpectedMessage2 wordt afgehandeld maar hier is de code nog niet uitgewerkt om de Message body te controleren en de data eruit te halen. Dit laat zien dat als dit voor meerdere Messages geschreven moet worden de Receive functie heel groot en onoverzichtelijk wordt.

De oude methode had twee grote nadelen. Ten eerste moest de gebruiker veel code typen om weinig gedaan te krijgen en dit was elke keer bijna dezelfde code. Ten tweede moest de gebruiker de interne werking van de Message klasse weten. Terwijl dit juist bij het aanmaken van de Message verborgen was voor de gebruiker door de MessageCreator klasse.

**De mogelijkheden**

Er is als mogelijkheid eigenlijk maar één optie bekeken. Dit was het gebruik van Attribute klasse in combinatie met de Reflection library van dotNET om dit voor de gebruiker te doen. Voor de naam van de Module was er al gebruikt gemaakt van de Attribute klasse en van de Reflection library. Hierdoor was bekend dat deze hiervoor goed gebruikt kon worden. Hoe deze oplossing precies intern werkt zal niet besproken worden. Het eind resultaat zal wel worden besproken om duidelijk te maken hoe groot de verbetering van de oplossing is.

**De oplossing**

De oplossing die geïmplementeerd is maakt gebruik van Attribute klasse in combinatie met de Reflection library. Met een Attribute kan er informatie in een klasse gezet worden. Deze informatie kan hierna met behulp van de Reflection library eruit gehaald worden.

```
[MessageTypeToListenTo("ExpectedMessage")]
public void ReceiveExpectedMessage(int leeftijd, string naam)
{
    //Code to do something with the age and name
}

[MessageTypeToListenTo("ExpectedMessage2")]
public void ReceiveExpectedMessage()
{
    //Code for handeling ExpectedMessage 2
}
```

Figuur 20: Code voor het ontvangen van de ExpectedMessage en de ExpectedMessage2

Met de oplossing die geïmplementeerd is is er veel minder code nodig om hetzelfde gedaan te krijgen. Hiernaast is het veel overzichtelijker wat er gebeurd. Het is niet meer nodig om alle Messages af te handelen in de Receive methode. De gebruiker kan zelf methodes aanmaken en de naam van de methodes kiezen. Door de MessageTypeToListenTo Attribuut boven de methode te zetten wordt er voor het framework duidelijk dat deze methode elke keer moet worden aangeroepen als die message binnen komt bij het framework. Hierdoor hoeft er niet meer door de gebruiker gecontroleerd worden in de methode of de message afgehandeld moet worden.

Dit is slechts de eerste stap van een methode met het MessageTypeToListenTo Attribuut. Als de methode onder de MessageTypeToListenTo attribuut parameters bevat word er door het framework gecontroleerd of de Message de waardes van de parameters bevat in de Message body. Hierin wordt de naam van de parameter tegenover de key van de paren gezet en de type van de parameter tegenover de type van de value. Als dit overeenkomt zal de methode worden aangeroepen met de waardes van de Message body.

```
new MessageCreator("NewPersoonEvent")
    .With("Persoon")
        .Add("naam", persoon.naam)
        .Add("leeftijd", persoon.leeftijd)
    .End()
    .Send(this);
```

Figuur 21: Het versturen van het NewPersoonEvent

```
[MessageTypeToListenTo("NewPersoonEvent")]
public void ReceiveNewPersoonEvent(
    string Persoon_naam, int Persoon_leeftijd)
{
    //Handle the NewPersoonEvent
}
```

Figuur 22: Het afhandelen van de NewPersoonEvent

In de afbeeldingen hierboven wordt links de NewPersoonEvent message verstuurd. In de afbeelding daarnaast is te zien dat een module luistert naar dit event doormiddel de ReceiveNewPersoonEvent methode. Zoals te zien is bevat de verstuurde message een intern paar genaamd persoon dat weer de paren van naam en leeftijd bevat. In de ReceiveNewPersoonEvent worden de waardes van deze paren uit de message gehaald doormiddel van de parameters van de methode. Het '_' karakter wordt gebruikt om het framework te laten weten dat de waarde in een intern paar staan. Zoals te zien is wordt door de combinatie van de MessageCreator en de MessageTypeToListenTo de gehele Message klasse van de gebruiker afgeschermd. Hierdoor is de gebruiksvriendelijkheid een stuk hoger.

Het MessageTypeToListenTo Attribute heeft ook nog een wildcard. Als er een '*' karakter wordt meegegeven zal de methode bij alle messages worden aangeroepen. Dit kan gebruikt worden om alle messages die naar het framework worden verstuurd op te vangen.

### *De meta informatie geven over het framework*

**Het probleem**

Een van de functionaliteiten van het framework is meta informatie geven over het framework. Hiermee wordt bedoeld dat het framework informatie geeft over de interne modules en messages in het framework. Deze informatie is nodig voor de volgende stagiair om de event flow duidelijk in beeld te brengen. In het huidige project is het ook van belang dat alle messages in beeld gebracht worden. Hiervoor is geen meta informatie nodig omdat dit gedaan kan worden door alle messages af te vangen en deze weer te geven. Voor de volgende stagiair is dit niet genoeg informatie. Voor die stagiair is het van belang om te weten welke modules reageren op de messages. Dit is alleen mogelijk als deze meta informatie ook door het framework geleverd wordt.

**De mogelijkheden**

Er zijn twee oplossingen voor dit probleem bekeken.

De eerste oplossing was het framework zo aan te passen dat de informatie die nodig is direct van het framework op te vragen was. Hierbij zou een Module deze informatie direct kunnen opvragen aan het framework. Bij deze oplossing is de geïnteresseerde partij verantwoordelijk om de data vaak genoeg op vragen aan het framework zodat het de nieuwste data heeft. Om deze oplossing te implementeren is het nodig dat het framework de verstuurde en ontvangen Messages opslaat. Dit zou ervoor zorgen dat als het framework lang genoeg draait hier mogelijk problemen mee krijgt. Hierbij kan gedacht worden aan het vollopen van de stack waar de informatie op staat en dat de informatie het onoverzichtelijk wordt omdat het zo veel is. Om deze redenen is deze oplossing niet de juiste oplossing.

De twee oplossing die bekeken is zijn zogenoemde Meta-Messages gebruiken. Meta-Messages zijn speciale Message die door het framework worden verstuurd. Deze Meta-Messages worden verstuurd door het framework zodra een module een message verstuurd en zodra een module een message afhandelt. Als een Module geïnteresseerd is in deze informatie hoeft hij alleen de Meta-Message op te vangen. Het grote voordeel aan deze oplossing is dat het framework deze informatie niet zelf hoeft op te slaan en hierdoor geen stack hoeft bij te houden met de informatie.

**De oplossing**

Er is gekozen om Meta-Message te gebruiken. Deze oplossing is gekozen omdat dit ervoor zorgt dat er geen grote extra aanpassingen aan het framework hoeven worden gedaan en dat er geen extra data hoeft worden opgeslagen in het framework. Hiernaast is het voor de gebruiker gemakkelijk om de nodige informatie te krijgen omdat het alleen de juiste Messages hoeft af te vangen in een module. Hierdoor kan de volgende stagiair gemakkelijk aan de juiste informatie komen. Er worden ook Meta-Messages verstuurd door het framework als er modules worden toegevoegd of verwijderd door de module.

De volgende Meta-Messages worden verstuurd:

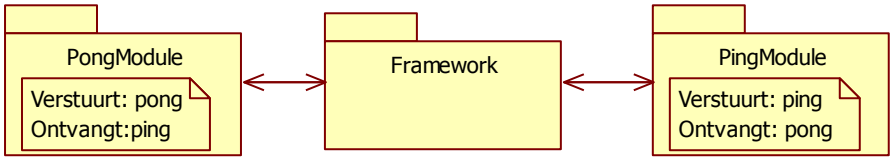
1. ModuleAdded: Wordt verstuurd als er een Module wordt toegevoegd aan de Register.
2. ModuleRemoved: Wordt verstuurd als er een Module wordt verwijderd uit de Register
3. MessagesSend: Wordt verstuurd als er een Message wordt verstuurd door een Module.
4. MessageReceived: Wordt verstuurd als er een Message wordt ontvangen en afgehandeld door een Module.

De MessageSend en MessageReceive Meta-Messages bevatten de data van de Message waar het om gaat. De ModuleAdded en ModuleRemoved Meta-Messages bevatten de data van de Module waar het om gaat.

### 3.3.3 Voorbeeld applicatie in het framework

Nu het framework is toegelicht zal er aan de hand van een kort voorbeeld worden laten zien hoe een simpele applicatie in het framework werkt. Dit voorbeeld dient vooral om te laten zien hoe gemakkelijk het is om een applicatie te ontwikkelen in het framework, het dient nog niet als bewijs dat het framework werkt. Hiervoor is er een uitgebreidere applicatie ontwikkeld die later in de scriptie behandeld wordt.

In dit voorbeeld zal er een applicatie ontwikkeld worden genaamd PingPong. De PingPong applicatie bestaat uit twee modules. De ping module en de pong module. De ping module stuurt het ping bericht als het een pong bericht bericht binnen krijgt en de pong module stuurt het pong bericht als het een ping bericht binnen krijgt.



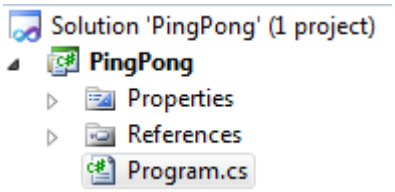

Figuur 23: De Module structuur van de PingPong applicatie

Om deze applicatie te maken moeten de volgende stappen ondernomen worden:

1. De solution moet aangemaakt worden.
2. Er moet een project worden toegevoegd per module.
3. Er moet een Payload aangemaakt worden voor iedere module.
4. De Modules moeten geïnitialiseerd worden en toegevoegd worden aan de Register in de Main.

**Het opzetten van het project**

De eerste stap is het opzetten van een nieuw project in Visual studio. In dit voorbeeld zullen we een Console applicatie opzetten genaamd PingPong. Na het opzetten van het project ziet het er als het goed is als volgt uit:




Figuur 24: Het nieuw aangemaakt project

De Program.cs zal uiteindelijk gebruikt worden om de modules aan te maken en toe te voegen aan de Register.

**Het aanmaken van de Modules**

Voor beide Module zal er nu een eigen package worden toegevoegd. Eerst zal er de Ping Module worden aangemaakt. Hiervoor moet er een nieuw Class library project worden toegevoegd. De reden dat elke Module een eigen project krijgt is om de codes van de Modules van elkaar af te schermen. Dit zorgt ervoor dat de Modules niet bij elkaars code kunnen. Dit garandeert dat alle communicatie tussen Modules verloopt via het framework. Om dit veder te garanderen is het aan te raden om alle methodes, klasse en attributen in een module op internal te zetten. Dit garandeert dat deze niet buiten de package en dus de module kunnen worden aangeroepen. Hierbij moet wel gezegd worden dat de Payload van de module public moet zijn om met het framework te communiceren.

### *Het aanmaken van de Payload*

Voordat de Payload aangemaakt kan worden in het project moet er een referentie worden toegevoegd naar het Message-framework. Nadat dit gedaan is kan de Payload aangemaakt worden.

```
[ModuleName("PingModule")]
public class PingPayload : IPayload
{
    [MessageTypeToListenTo("pong")]
    public void Receive()
    {
        System.Threading.Thread.Sleep(500);
        Console.Out.WriteLine("ping");
        new MessageCreator("ping").Send(this);
    }
}
```

Figuur 25: De code van de PingPayload

Hierboven is de gehele code van de PingPayload. De PingPayload luistert naar het bericht pong. Als het bericht pong binnenkomt bij de Ping Module zal de Receive functie worden aangeroepen. Hierna zal er een halve seconde gewacht worden. Dit is gedaan zodat de berichten elkaar niet te snel opvolgen. Hierna zal er “ping” op de Console geschreven worden en zal het ping bericht gestuurd worden.

De Ping Module is nu volledig klaar. Voor het aanmaken van de Pong Module zijn de stappen precies hetzelfde als van de Ping Module. Hierom zal dit niet veder worden toegelicht.

### *Het opstarten van het project*

Nu de Modules opgezet zijn moet de Main geschreven worden van het project. In de Main moeten de Modules aangemaakt worden en toegevoegd worden aan de Register.

```
static void Main(string[] args)
{
    //Het aanmaken en toevoegen van de twee Modules
    IPayload pingPayload = new Ping.PingPayload();
    Module pingModule = new Module(pingPayload);
    Register.AddModule(pingModule);
    IPayload pongPayload = new Pong.PongPayload();
    Module pongModule = new Module(pongPayload);
    Register.AddModule(pongModule);
    //Het versturen het eerste pong bericht
    new MessageCreator("pong").Send(pongPayload);
    Console.ReadKey();
}
```

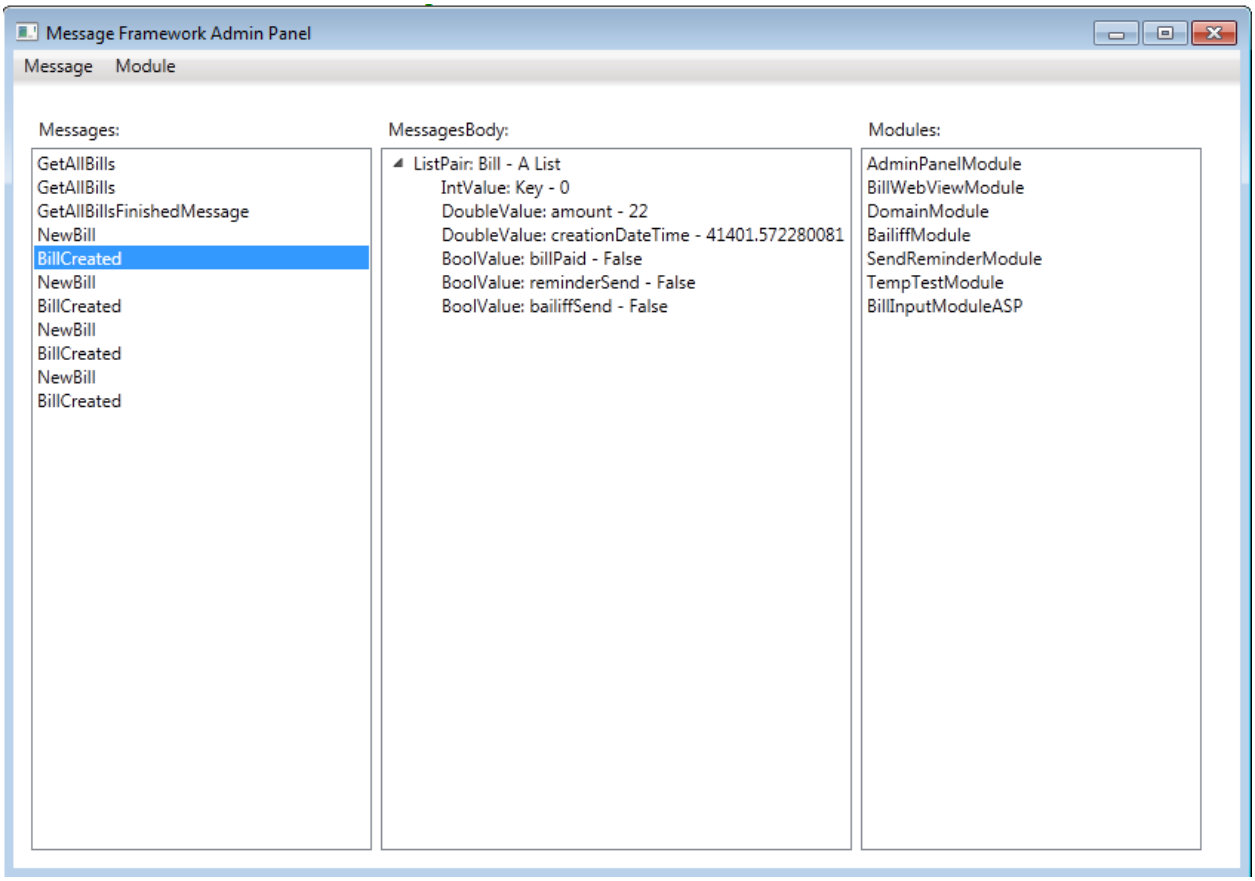
Figuur 26: De code van de Main

In de code hierboven wordt een Ping Module en een Pong Module aangemaakt. Nadat deze zijn aangemaakt worden ze toegevoegd aan de Register. Hierna wordt er een pong Message verstuurd. Deze Message start de ketting van messages. Omdat een Message altijd moet worden verstuurd vanaf een Payload wordt hiervoor de pongPayload gebruikt. Het resultaat van deze code is dat er elke halve seconden er ping of pong op de console gezet wordt.

Dit voorbeeld project is heel simpel maar laat goed zien welke stappen er ondernomen moeten worden om een applicatie te ontwikkelen in het framework. Als de applicatie ingewikkelder wordt zijn er niet meer stappen nodig om de communicatie tussen de Modules te regelen.

## 3.4 Beheerders-interface

De Beheerder-interface ook wel de Admin Panel genoemd is een losse Module die ontwikkeld is tegelijk met het framework. De Beheerders-interface is een module met een eigen Windows Presentation Foundation interface waarin informatie over het framework te zien is. Een Windows Presentation Foundation ofwel WPF interface is een manier om interfaces te maken met Visual studio. Het kan vergeleken worden met Windows FORMS.




De Figuur 27: De Beheerders-interface

De Beheerders-interface is gebouwd om aan de volgende eisen van het project te voldoen:

1. De events binnen het framework moeten tijdens de uitvoering van de applicatie in beeld worden gebracht.
2. Het moet mogelijk zijn om events te generen.
3. Het moet mogelijk zijn om dynamisch modules in en uit te laden

De Beheerders-interface kan worden gebruikt om te bekijken wat er in het framework gebeurt. Het kan ook gebruikt worden om modules te testen. Om dit te doen bevat de Beheerders-interface de volgende functionaliteiten:

1. Alle Messages die verstuurd worden met het framework worden weergegeven.
2. Alle Modules die in het framework zijn geregistreerd worden weergegeven.
3. Er kunnen Messages verstuurd worden vanuit de Beheerders-interface
4. Er kunnen Modules dynamisch worden ingeladen in het framework
5. Er kunnen Modules worden verwijderd uit het framework

De functionaliteit van de Beheerders-interface maakt het mogelijk om gemakkelijk een applicatie die ontwikkeld is in het framework uit te testen. Dit komt omdat alle berichten die verstuurd worden door de modules in beeld gebracht worden en omdat modules onafhankelijk in en uit geladen kunnen worden.

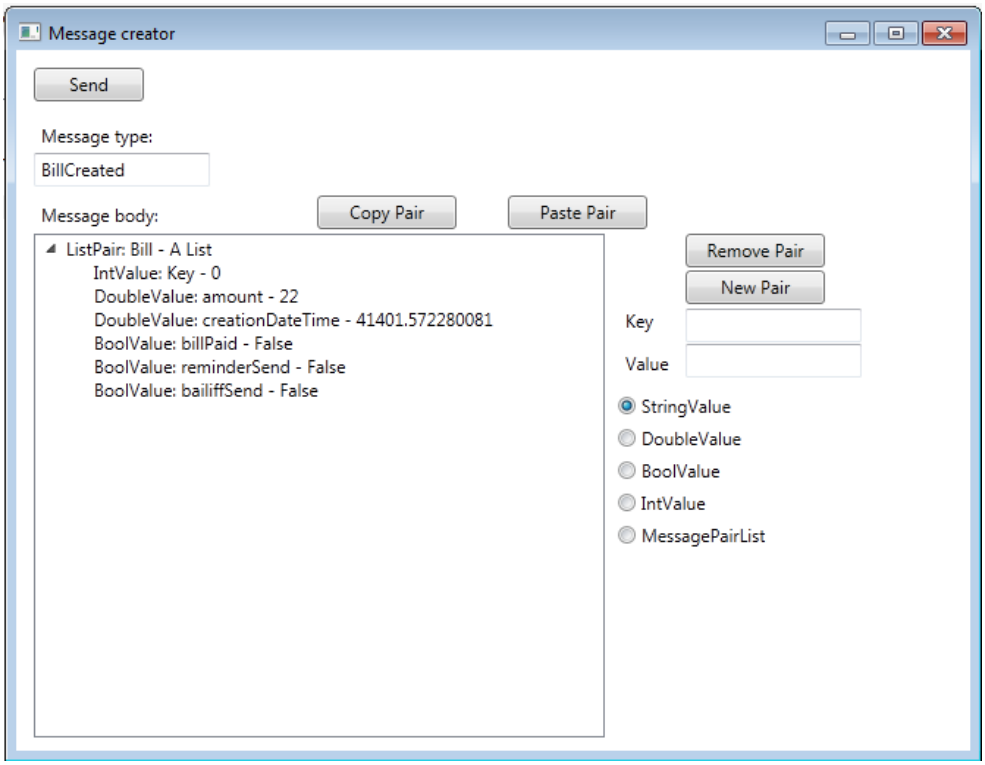
### *Het weergeven van de Messages en de Modules*

Een van de functionaliteiten van de Beheerders-interface is het weergeven van alle Messages en alle Modules. De Beheerders-interface komt aan deze informatie door naar alle Messages te luisteren. In de beheerders-interface worden alleen alle Messages weergegeven. Hierbij wordt er niet weergegeven welke Module er daadwerkelijk de Messages afhandelen. Door op een Message te klikken wordt de Message body van de Message getoond in het middelste scherm.

In het rechter scherm worden alle Modules weergegeven die ingeladen zijn in het framework. De Beheerders-interface komt aan deze informatie door naar de Meta-messages te luisteren die laten weten wanneer een module is toegevoegd of verwijderd.

### *Aanmaken en versturen van een Message*

Het is mogelijk om Messages aan te maken en te versturen met behulp van de Beheerders-interface. Wanneer een nieuwe Message wordt aangemaakt wordt het volgende scherm geopend.




**Figuur 28: De Message creator interface**

Het is mogelijk om de Message type en de Message body van de Message te definiëren. Door op de Send knop de drukken wordt de Message verstuurd naar het framework. Hierbij is de IPayload van de Beheerders-interface de verstuurder van het bericht.

### *Modules dynamisch in en uit laden*

Het is ook mogelijke om met de Beheerders-interface Modules dynamisch in en uit te laden. Dit kan gedaan worden door een Assembly bestand in te laden waarin een IPayload implementatie staat. Doormiddel van de Reflection library van dotNET worden alle IPayload implementaties gezocht in de gekozen Assembly. Deze gevonden Payloads worden hierna aangemaakt en gebruikt om Modules te maken. Dit is mogelijk omdat alle informatie die nodig is om een Module aan te maken in de IPayload implementatie te vinden is.

Het is ook mogelijk om een ingeladen Module te verwijderen uit het framework met behulp van de Beheerders-interface. Nadat de Module uit het framework wordt verwijderd zal de Dispose van de Payload worden aangeroepen. Dit zorgt ervoor dat alle resources die de Module had worden losgelaten.

## 3.5 De EDA applicatie

De tweede deelopdracht van het project was het bouwen van een applicatie in het Message-framework. Het bouwen van deze applicatie heeft twee doelen.

1. Het Eerste doel is aan tonen dat het mogelijk is om een EDA applicatie te ontwikkelen in het framework.
2. Het tweede doel is bewijzen dat er aan de vijfde eis van het framework voldaan is. Namelijk dat het mogelijk is dat een module een eigen front end heeft.

Om het eerste doel te behalen zal de applicatie ontwikkeld worden volgens het EDA principe. Hierbij moet de communicatie tussen de verschillende modules volledig bestaan uit events. De modules moeten hierbij de event emitters en de event consumers voorstellen en de messages moeten de events voorstellen.

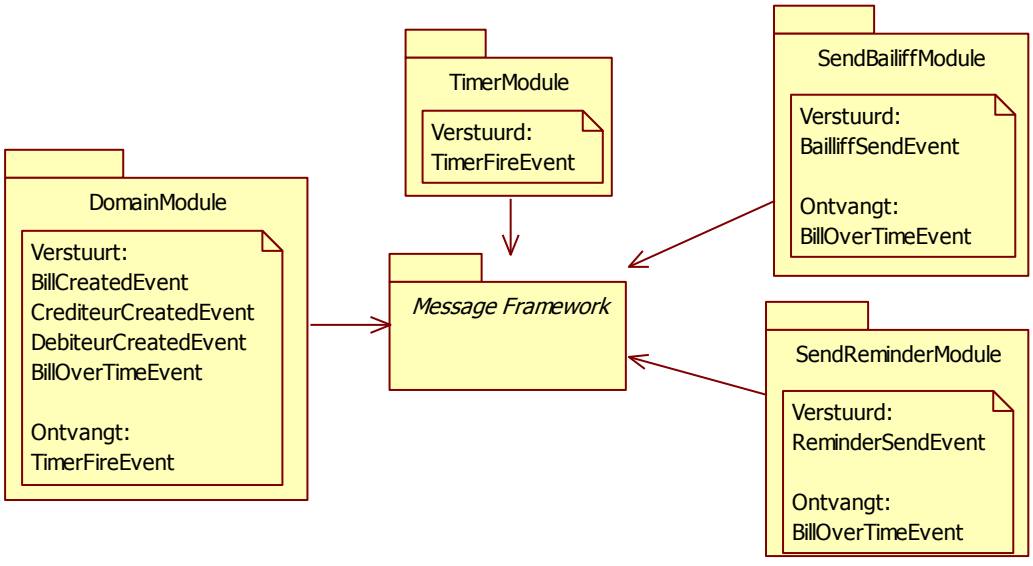
Het tweede doel zal behaald worden door modules te ontwikkelen met een eigen front end. Om te bewijzen dat er aan de eis is voldaan moet er aangetoond worden dat een module ontwikkeld kan worden die een normale gebruikersinterface bevat en dat het mogelijk is een module te ontwikkelen die een web interface bevat.

### 3.5.1 De incasso case

De gekozen applicatie is een applicatie die wordt gebruikt in een opdracht van de academie van Sogyo. Het gaat over een digitale incasso case. De beschrijving van de incasso case is als bijlage 3 toegevoegd. Het idee achter het systeem is dat er rekeningen kunnen worden toegevoegd aan het systeem. Als de rekening na één week niet betaald is zal er automatisch een waarschuwing verstuurd worden. Als na twee weken de rekening nog niet betaald is zal er automatisch een deurwaarder gestuurd worden.

### 3.5.2 De architectuur

Om de applicatie volgens het EDA principe te ontwikkelen zal de architectuur van de applicatie worden vastgelegd door de modules te bepalen en vast te leggen welke messages deze modules versturen. De modules zullen allemaal event emitters en/of event consumers voorstellen en de messages stellen allemaal events voor.




**Figuur 29: De Modules in de applicatie en de events die ze versturen en die ze ontvangen.**

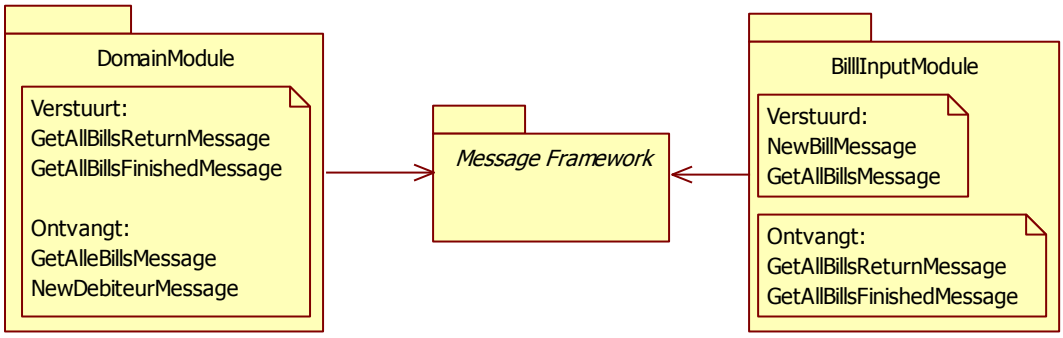
Zoals te zien is zijn er in de applicatie 4 modules aanwezig. Deze Modules versturen en ontvangen verschillende Messages. Alle Messages die in de applicatie verstuurd worden stellen events voor, dit is te zien doordat ze allemaal op Event eindigen. Het is niet van belang om door te nemen wat deze modules precies doen of hoe de module intern werken. Het is wel van belang om te vertellen dat de modules volledig volgens het EDA principe communiceren.

### 3.5.3 WPF interface module

Om aan te tonen dat het mogelijk is een module te ontwikkelen met een normale gebruikersinterface als front end is de BillInputModule module ontwikkelend. De BillInputModule is een module met een WPF interface waarin nieuwe rekeningen aangemaakt kunnen worden en alle rekeningen bekeken kunnen worden.

De eerste taak van de BillInputModule is het aanmaken van nieuwe rekeningen. Een nieuwe rekening wordt aangemaakt door een NewBillMessage message te versturen naar het framework. De DomainModule zal hierna de rekening aanmaken. Zoals de naam al aangeeft is de NewBillMessage message geen event. Dit komt omdat er geen verandering van status plaatsvindt in de BillInputModule waardoor de NewBillMessage niet als een event gezien kan worden. De NewBillMessage is een aanvraag aan de DomainModule om een nieuwe rekening aan te maken. Dit gaat tegen het EDA principe in. Toch is dit toegestaan omdat het hierbij om gebruikers interactie gaat. Bij gebruikers interactie is het niet mogelijk om met events te werken tussen de modules. De rest van de interactie tussen de modules stellen wel events voor en hierom is het nog steeds een EDA applicatie.

De tweede taak van de BillInputModule is het weergeven van alle rekeningen. Hierbij is het probleem dat de BillInputModule deze informatie niet bevat. Deze informatie bevindt zich in de de DomainModule en deze twee modules kunnen niet direct met elkaar communiceren. Een module kan een message sturen naar een andere module maar er kan niet gereageerd worden door de andere module. Hierdoor is het niet mogelijk om direct informatie op te vragen aan een andere module.




**Figuur 30: De modules en de messages die te maken hebben met de BillInputModule**

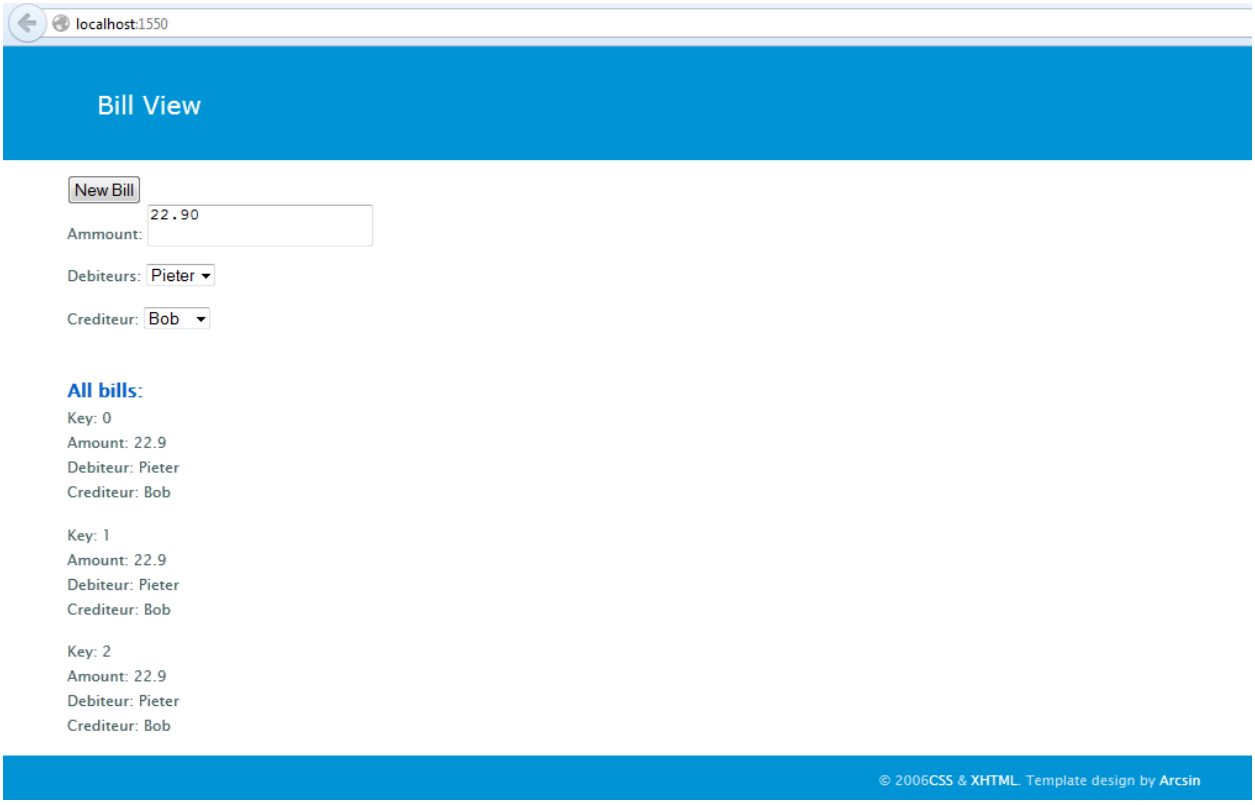
Dit probleem is opgelost door gebruik te maken van een moduleID bij messages. De moduleID is een unieke identifier van elke module. Als dit meegegeven wordt aan een message kan dit gebruikt worden om messages terug te sturen. Omdat het moduleID in de return message staat weet de eerste module dat de return message voor hem bedoeld is. Door gebruik te maken van de moduleID is het dus wel mogelijk dat twee modules met elkaar communiceren. Hierbij wordt de verbinding tussen de twee modules dynamisch opgezet en hebben de modules nog steeds geen sterke koppeling met elkaar.

### 3.5.4 Web Modules

Om aan het tweede doel van de applicatie te voldoen moet er ook een module ontwikkeld worden die een web interface heeft. Ook voor deze module is er gekozen om een module te maken waarin de rekeningen te zien zijn en waarin rekeningen aan te maken zijn. Deze module heet de BillWebModule.

De BillWebModule heeft dezelfde functionaliteit als de BillInputModule behalve dat de interface een web interface is. In de BillWebModule moet een webserver gestart worden en worden afgesloten. Om de webserver aan te maken zijn er verschillende technieken uitgeprobeerd. Hieronder vallen het zelf aanmaken van een webserver en het gebruik maken van de ingebouwde Visual studio web server. Uiteindelijk is er gekozen om gebruik te maken van een webserver genaamd Nancy.

Nancy is een lichtgewicht framework dat het mogelijk maakt om http gebaseerde services te bouwen voor dotNET. Nancy is volledig gratis en open source. Het is mogelijk om Nancy op te starten in de module en af te sluiten in de module. Hierna kunnen er met behulp van Nancy http aanvragen worden afgehandeld. Met behulp van Nancy is het mogelijk om web modules te ontwikkelen in het framework.




Figuur 31: De website van de BillWebModule

### 3.5.5 Applicatie doelen

De applicatie die ontwikkeld is werkt intern bijna volledig volgens het EDA principe. Er wordt alleen van het EDA principe afgestapt om de user interactie af te handelen. Dit is nodig omdat het anders niet mogelijk is om aan de juiste informatie te komen voor de gebruikersinterface. Dit haalt niets weg van het feit dat de rest van de communicatie tussen de modules volledig verloopt volgens het EDA principe. Hierdoor is de applicatie het bewijs dat het mogelijk is een applicatie te ontwikkelen volgens het EDA principe in het framework.

Het tweede doel is ook behaald. De BillWebModule en de BillInputModule laten zien dat het mogelijk is om modules te ontwikkelen die een eigen front end hebben. Hierbij kan er gekozen worden om met WPF te werken of met web interfaces. Het is ook mogelijk om met andere manieren een front end toe te voegen zoals FORMS. Het is niet nodig om deze ook te ontwikkelen omdat de gemaakte modules al genoeg bewijs leveren dat het mogelijk is om een module te ontwikkelen met een eigen front end.

## 3.6 Gebruikers handleiding, documentatie en Unit tests

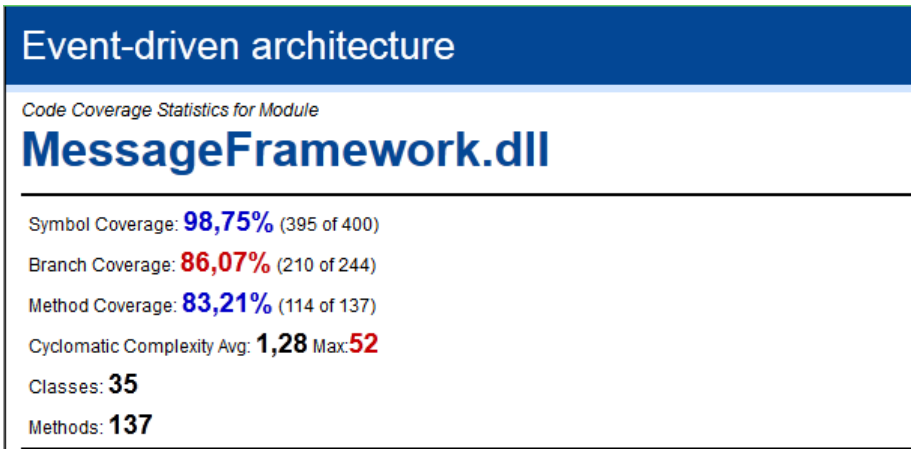
Nadat alle onderdelen van het project af waren is de gebruikershandleiding geschreven. De code documentatie en test dekking zijn tijdens het bouwen van het project bijgehouden. Wel is dit aan het einde van het project allemaal nagekeken en uitgebreid.

### 3.6.1 Gebruikershandleiding

De gebruikershandleiding neemt de taak van tutorial van het framework op zich. De gebruikershandleiding is geschreven met een gebruiker van het Message-framework in gedachten. Het is niet bedoeld als overdrachtsdocument van het framework zelf. Hier is een apart document voor geschreven. De gebruikershandleiding legt eerst de globale werking van het framework uit. Hierna worden aan de hand van voorbeelden de verschillende mogelijkheden van het framework behandeld.

### 3.6.2 Test dekking

Een van de eisen aan het framework was dat de test dekking boven de 90% procent moest zitten. Het framework bevat een testdekking van bijna 100%. Dit is gemeten met Ncover. Het is met het project niet mogelijk om 100% testdekking te krijgen omdat Ncover sommige delen van F# niet meetelt in de testdekking.




Figuur 32: Uitvoer van Ncover voor het framework

De missende dekking van de code is vooral te danken aan de MessagePair klasse die gebruik maakt van de f# Discrimination Union klasse. Deze wordt niet goed gemeten door Ncover en hierdoor mist een deel van de test dekking.

### 3.6.3 Code Documentatie

De gehele code van het Message-framework is gedocumenteerd. Dit is gedaan door gebruik te maken van XML documentatie van Visual Studio. Visual Studio kan hiervan gebruik maken om informatie te geven over het Message-framework bij het gebruik ervan. Hiernaast kan er automatisch een XML document aangemaakt worden bij het compileren van de code. Dit XML document bevat alle documentatie van het framework.

# 4 Conclusie & aanbevelingen

In dit hoofdstuk zal er eerst de conclusie besproken worden. Hierna zullen een aantal aanbevelingen gegeven worden over wat er in de toekomst verbeterd kan worden aan het framework.

## 4.1 Conclusie

Tijdens het project zijn alle eisen die aan het framework zijn gesteld behaald. Dit is gedaan door het bouwen van het framework, het bouwen van de Beheerders-interface en het bouwen van de applicatie. Hier volgt nog een korte herhaling hoe dit gedaan is.

1. **Het framework moet de communicatie tussen modules regelen.**

Het ontwerp van het framework zorgt ervoor dat alle Messages die verstuurd worden altijd door het framework worden verstuurd naar de andere modules. Dit zorgt ervoor dat de gehele communicatie tussen de modules door het framework wordt afgehandeld. Hierbij moet de gebruiker van het framework er zelf voor zorgen dat er geen andere communicatie plaats vindt tussen de modules.

2. **De events binnen het framework moeten tijdens de uitvoering van de applicatie in beeld worden gebracht.**

De messages en dus de events die in het framework plaatsvinden worden door de beheerders-interface weergegeven. De Meta-Messages die worden gegenereerd door het framework maken het mogelijk om in een latere fase van het overkoepelende project de message visueel in beeld te brengen.

3. **Het moet mogelijk zijn om events te generen.**

Vanuit de beheerders-interface is het mogelijk om messages en dus events te generen.

4. **Het moet mogelijk zijn om dynamisch modules in en uit te laden.**

De Beheerders-interface maakt het mogelijk om modules in en uit te laden. Dit is mogelijk omdat de architectuur van het framework zo is opgebouwd dat alle informatie die nodig is om een module op te starten en af te sluiten te vinden zijn in de IPayload implementatie.

5. **Het moet mogelijk zijn dat een module een eigen front end heeft.**

De incasso case applicatie laat twee voorbeelden zien waarin een module een front end heeft. Hierdoor is het bewijs geleverd dat het mogelijk is dat een module een eigen front end heeft.

6. **Het framework moet in de toekomst over verschillende systemen verspreid kunnen worden.**

De berichten die verstuurd moeten worden tussen de verschillende systemen zullen geserialiseerd worden. Het is daarom belangrijk dat de Messages klasse geserialiseerd kan worden. Dit is mogelijk omdat de Message intern alleen gebruik maakt van basis typen die geserialiseerd kunnen worden.

7. **Het framework moet getest worden met behulp van unit tests.**

Het framework voldoet ruim aan de eis m.b.t. de unit tests. Het bewijs hiervoor is de uitvoer van NCover.

8. **Het framework moet volledig gedocumenteerd zijn.**

Het framework is volledig gedocumenteerd. Dit is gedaan door gebruik te maken van XML documentatie.

Nadat de verplichte eisen aan het framework zijn ingewilligd is er ook veel aandacht besteed aan het gebruiksvriendelijk maken van het framework. Voor de gebruiker kan er met weinig stappen een applicatie ontwikkeld worden die met het framework werkt. Dit is een van de grote krachten van framework.

Alle deelopdrachten zijn ook voltooid. Het onderzoek naar de programmeertaal is verwerkt in een onderzoeksverslag. De incasso case applicatie laat zien dat het mogelijk is om een applicatie te ontwikkelen in het framework en de gebruikershandleiding is ook opgeleverd. Doordat alle eisen van het framework ingewilligd zijn en alle deelproducten zijn opgeleverd is het project geslaagd.

## 4.2 Aanbevelingen

Mijn eerste aanbeveling voor de toekomstige ontwikkeling van het framework is om de efficiëntie van het framework te verbeteren. Tijdens de ontwikkeling had de efficiëntie van het framework een lage prioriteit. Een van de punten die verbeterd kan worden is de code waarin de module bepaalt of een message door de module moet worden afgehandeld. Bij het schrijven van de code voor dit stuk is er geen rekening gehouden met efficiëntie en hierdoor geloof ik dat er zeker nog verbeteringen zijn door te voeren in deze code.

In de volgende stage opdracht zal er gewerkt worden aan het visueel weergeven van de message stroom in het Message-framework. De informatie om dit te doen wordt al gegeven door het framework in de vorm van de Meta-Messages. Bij de volgende stage is het daarom aan te raden om een module te ontwikkelen die de Meta-Message opvangt. Hierna kan er direct begonnen worden om deze visueel weer te geven zonder dat er aan het framework gewerkt hoeft te worden.

# 5 Evaluatie

In deze evaluatie zal er eerst het project worden geëvalueerd. Hierin zal eerst bekeken worden wat er goed ging en wat er fout ging in het project. Ook zal er gekeken worden hoe Scrum beviel in een project team van 1 persoon. Hierna zal de verkregen kennis in dit project kort besproken worden.

## 5.1 Evaluatie van het project.

Het project is zoals in de conclusie besproken goed afgerond. Het project is redelijk voorspoedig verlopen zonder echte problemen. Ik geloof dat dit mede kwam door het gebruik van Scrum en de goede begeleiding vanuit de Hogeschool Utrecht en vanuit Sogyo.

### 5.1.1 Werken met F#

In het project is er gewerkt met F#. Voordat er aan deze opdracht was begonnen had ik nog geen ervaring met F#. Ik heb hiermee om leren gaan tijdens het project. Dit is gedaan door eerst een aantal tutorials online te volgen. Er zijn geen grote problemen geweest met het werken met F#. Dit komt omdat Visual Studio een goede integratie heeft met F#. Hiernaast is de online documentatie van F# goed bijgewerkt door Microsoft.

Ik heb geleerd dat F# is programmeertaal is met veel potentie. Het heeft de volledige kracht van het dotNET framework achter zich en heeft hierdoor een aantal sterke libraries. Hiernaast heeft het ook nog een aantal voordelen van zichzelf. Voorbeelden hiervan zijn de ingebouwde actor klasse en de Discriminated Union klasse.

### 5.1.2 Werken met Scrum

Het werken met Scrum beviel goed. Ondanks dat er veel onderdelen van Scrum afvallen in een 1-persoons projectteam zijn de onderdelen die overbleven heel geschikt voor een afstudeeropdracht. Het werken met sprints heeft het grote voordeel dat je per sprint je richting van het project kan aanpassen en je prioriteiten opnieuw kan evalueren. Dit was in dit project heel handig omdat het lastig was een langere planning te maken. Dit kwam omdat bij dit project veel informatie die nodig was voor de planning pas tijdens het project naar voren kwam.

Voor de Scrum tool is er gebruik gemaakt van Redmine. Redmine beviel goed. Er kunnen gemakkelijk nieuwe sprints en user stories worden toegevoegd.

## 5.2 Verkregen kennis

Tijdens het project heb ik ervaring opgedaan met verschillende technieken en onderwerpen.

1. Event driven architecture
2. Het Actor model
3. Werken met F#
4. Werken met Nancy als webserver
5. Werken met fluent interfaces
6. Werken met de Reflection library van dotNET
7. Werken met Scrum

Van deze onderwerpen kan er vooral gezegd worden dat het actor model en de Reflection library veel indruk op mij gemaakt hebben. Het actor model is al een oud model maar het wordt steeds logischer om er gebruiken van te maken. Dit komt omdat de huidige computersystemen steeds meer processen tegelijk kunnen draaien. Door de automatische opdeling in losse processing eenheden bij het actor model wordt hier goed gebruik van gemaakt.

De Reflection library heeft een sterke indruk op mij gemaakt omdat het vele mogelijkheden biedt die ik eerst niet voor mogelijk hield. Het inladen van Assemblies en het analyseren van klasse op de namen van methodes en variables zijn hier voorbeelden van. Zelfs private en internal variables van een klasse kunnen opgehaald worden met behulp van de Reflection library.

# 6 Bronnen

*Actor model Wikipedia*. (sd). Opgehaald van Wikipedia: http://en.wikipedia.org/wiki/Actor_model

Bolognese, L. (sd). *An introduction to Microsoft F#*. Opgehaald van Channel 9: http://channel9.msdn.com/blogs/pdc2008/tl11

Hewitt, C. (2012, April 9). Hewitt, Meijer and Szyperski: The Actor Model (everything you wanted to know, but were afraid to ask). (E. Mijer, Interviewer)

Microsoft. (sd). *CCR Introduction*. Opgehaald van Msdn Microsoft: http://msdn.microsoft.com/en-us/library/bb648752.aspx

Microsoft. (sd). *Control.MailboxProcessor<'Msg> Class (F#*. Opgehaald van Msdn Microsoft: http://msdn.microsoft.com/en-us/library/ee370357.aspx

Microsoft. (sd). *Object Serialization in the .NET Framework*. Opgehaald van Msdn Microsoft: http://msdn.microsoft.com/en-us/library/ms973893.aspx

Microsoft. (sd). *SandCastle codeplex*. Opgehaald van http://sandcastle.codeplex.com/

Microsoft. (sd). *System.reflection*. Opgehaald van msdn Microsoft: http://msdn.microsoft.com/en-us/library/system.reflection%28v=vs.100%29.aspx

Microsoft. (sd). *XML Documentation (F#)*. Opgehaald van Msdn Microsoft: msdn.microsoft.com/en-us/library/dd233217.aspx

*Nancy home page*. (sd). Opgehaald van http://nancyfx.org/

# 7 Bijlage

## 7.1 Bijlage 1: Plan van aanpak

# Plan van aanpak

### Revisielijst

| Versienummer | Datum | Reden revisie |
|---|---|---|
| 0.1-0.5 | 1/02/2013 - 27/02/2013 | Eest opzet verslag & inhoudelijke uitwerking |
| 0.6 | 4/03/2013 | Opmerkingen Ralf Wolter verwerkt |
| 1.0 | 6/03/2013 | Opmerkingen Peter van Rooijen verwerkt |

### Lijst van Afkortingen

| Afkorting | Term |
|---|---|
| EDA | Event-driven architecture |

# Inhoudsopgave

# Inleiding

Dit plan van aanpak is geschreven naar aanleiding van het afstuderen in opdracht van de Hogeschool Utrecht. De afstudeeropdracht wordt uitgevoerd bij Sogyo.

Het plan van aanpak bestaat uit vijf delen. In het eerste hoofdstuk zal de achtergrond van de betrokken partijen toegelicht worden. In het tweede hoofdstuk zal de context van het project worden beschreven. Het derde hoofdstuk zal over de projectopdracht gaan. Hierin zal de opdracht worden toegelicht. De doelstellingen, deelopdrachten, project grenzen, randvoorwaarden, op te leveren producten en risico's zullen ook worden toegelicht. Het vierde hoofdstuk zal uitleg geven over de projectactiviteiten en de planning van het project. Het vijfde hoofdstuk zal de risico's benoemen en vertellen wat de maatregelen bij de risico's zijn. Het laatste hoofdstuk zal gaan over de te gebruiken methodes tijdens het project. Hieronder vallen de te gebruiken kwaliteitsbewaking en de tools.

Als bijlage is het afstudeercontract bijgevoegd dat ondertekend moet worden door de betrokken partijen als het plan van aanpak is goedgekeurd.

# 1 Achtergrond

In dit hoofdstuk zal er meer verteld worden over de betrokken partijen. Er zal meer verteld worden over de opdrachtnemer, opdrachtgevers en over de begeleiders bij het project. De contactgegevens van alle betrokken partijen zijn aan het einde van het verslag terug te vinden.

## 1.1 Betrokken partijen

In dit gedeelte van het verslag zal er toelichting gegeven worden over de betrokken partijen bij dit project.

**Opdrachtnemer**
De opdracht zal worden uitgevoerd door Tim Linschoten, student Technische Informatica aan de Hogeschool Utrecht.

**Opdrachtgever**
De afstudeeropdracht zal plaats vinden bij Sogyo. Sogyo is een bedrijf gelegen aan de Utrechtseweg in de Bilt. Het is gelegen op landgoed Sandwick in een omgebouwde boerderij. Sogyo houdt zicht vooral bezig met software innovaties. Hieronder vallen innovaties op het gebied van software architectuur, software ontwikkeling en software integratie. Er werken circa 80 mensen bij Sogyo en ongeveer de helft hiervan is aanwezig bij de vestiging waar de stage plaats vindt. De werkzaamheden van Sogyo zijn op te delen in drie pijlers: de Opleidingen, Detachering & Consultancy en Development. In de Sogyo Opleidingen wordt er gewerkt aan de ontplooiing van IT-talent. Sogyo leidt niet alleen hun eigen medewerkers op maar biedt ook in-company opleidingen aan voor hun opdrachtgevers. De pijler Detachering & Consultancy werft, selecteert, begeleidt en ontwikkelt ambitieuze en talentvolle IT'ers. De inzet van Sogyo-professionals varieert van tijdelijke inzet op consultancy of detacheringsbasis tot trajecten die tot een vast dienstverband bij hun opdrachtgevers leiden. Bij de pijler Development wordt er gewerkt aan de ontwikkeling van IT-toepassingen.

**Bedrijfsbegeleider**
De bedrijfsbegeleider is Ralf Wolter. De heer Wolter is senior engineer en afstudeerbegeleider bij Sogyo. Hij heeft een afgeronde WO opleiding, richting CKI en een master e-technology. Hij is bereikbaar via mail op rwolter@sogyo.nl.

**Contactpersoon**
De contactpersoon voor de stage is Brenda Klever. Zij is de HR-consultant bij Sogyo. Zij is te bereiken via mail op bklever@sogyo.nl of telefonisch op 0302202216.

**Docentbegeleider**
De rol van docentbegeleider vanuit de Hogeschool Utrecht zal worden vervuld door de heer Peter van Rooijen. Hij is te bereiken op peter.vanrooijen@hu.nl

# 2 Context

In dit hoofdstuk zal de context rondom het project worden toegelicht. Eerst zal de aanleiding tot het project besproken worden, hierop volgt de probleemstelling van het project.

## 2.1 Aanleiding

Een van de bezigheden van Sogyo is het adviseren van software bedrijven over de te gebruiken tools, talen en architecturen. Sogyo ziet veel potentie in het maken van software gebaseerd op *event-driven architecture*, hierna EDA genoemd. Toch hebben de klanten van Sogyo nog twijfels over het gebruik van EDA.

Event-driven architecture is een software architectuur pattern dat het gebruik van events promoot. Bij een systeem dat hierop gebaseerd is zal de communicatie voornamelijk verlopen via events. Een event kan gezien worden als een verandering van status. Als er onderdelen van een systeem geïnteresseerd zijn in een event van een ander onderdeel dan kan hij hier luisteraar van worden. Wanneer een event optreedt, zullen de geïnteresseerde onderdelen van het systeem hiervan op de hoogte gesteld worden.

Doordat componenten nooit direct met elkaar communiceren is er een hele losse koppeling tussen deze componenten. De losse koppeling zorgt ervoor dat de verschillende onderdelen van een systeem elkaar niet hoeven te kennen. Dit biedt het voordeel dat de onderdelen in het systeem makkelijker kunnen worden aangepast of vervangen zonder dat de rest van het systeem hier last van heeft.

De losse koppeling van EDA zorgt ook voor een groot probleem. Omdat er geen directe koppeling is tussen onderdelen is het moeilijk om de informatie stroom in de applicatie te achterhalen. Hiermee wordt bedoeld dat het lastig is om te achterhalen welke communicatie tussen de verschillende componenten plaats vindt. Dit komt omdat het niet direct te achterhalen is welke events er optreden en welke componenten op deze events reageren. Dit is een groot probleem en de voornaamste reden dat klanten van Sogyo zorgen hebben over het gebruik van EDA.

Om de zorgen bij de klanten weg te nemen zal er een onderzoek naar EDA gedaan worden. Dit onderzoek moet laten zien dat het wel goed mogelijk is om EDA applicaties te ontwikkelen en te onderhouden. Om dit onderzoek te ondersteunen zal er een framework ontwikkeld worden. Dit framework moet het mogelijk maken om applicaties te ontwikkelen op basis van EDA waarbij wel de informatie stroom van de applicatie zichtbaar gemaakt kan worden.

## 2.2 Probleemstelling

Sogyo wil zijn klanten adviseren om gebruik te maken van EDA in hun producten. Het probleem is dat de klanten hier nog zorgen over hebben. Deze zorgen zijn er vooral omdat bij een applicatie die gebouwd is aan de hand van EDA de koppeling tussen onderdelen laag is. Deze lage koppeling is er omdat de verschillende onderdelen van het systeem elkaar niet hoeven te kennen. Ze hoeven alleen te reageren op de events binnen in het systeem. Dit zorgt ervoor dat het lastig te achterhalen is hoe de informatie stroom binnen de applicatie is.

Om deze twijfels weg te nemen zal er een onderzoek plaats vinden naar EDA. Dit onderzoek zal ondersteund worden door een framework. Dit framework moet het mogelijk maken om de informatie stroom binnen een applicatie die hierop gebaseerd is in beeld te brengen.

# 3 Projectopdracht

In dit hoofdstuk zal de projectopdracht worden toegelicht. Als eerste zal de afstudeeropdracht besproken worden. Hierop volgen de doelstellingen, hoofdopdracht, deelopdrachten, project grenzen, randvoorwaarden, op te leveren producten en de risico's.

## 3.1 Afstudeeropdracht

De afstudeeropdracht bestaat uit het ontwerpen, implementeren en documenteren van een framework. Het framework moet het mogelijk maken om applicaties te ontwikkelen op basis van EDA. Van een applicatie die gebouwd is op basis van het framework moet het mogelijk zijn om te achterhalen wat de informatie stroom is binnen in de applicatie.

Een applicatie die ontwikkeld is in het framework zal opgedeeld zijn in modules. Elke module is een onderdeel van het systeem dat events kan ontvangen. Zodra een event plaats vindt zullen de modules op de hoogte gesteld worden. De communicatie tussen de modules is de taak van het framework.

Het framework moet unit tests bevatten en moet volledig gedocumenteerd zijn. Naast de documentatie van de code moet er ook een tutorial geschreven worden hoe het framework gebruikt moet worden. De taal van het framework zal C# of F# zijn. Aan het begin van het project zal aan de hand van een vooronderzoek besloten moeten worden welke taal er gebruikt gaat worden.

## 3.2 Doelstellingen

De doelstelling van dit project is het bouwen van een framework. Het framework zal twee doelen moeten vervullen. Het eerste doel van het framework is het onderzoek naar een EDA systeem te ondersteunen. Het onderzoek zal vooral naar voren moeten halen wat de voordelen en nadelen zijn van het bouwen van een applicatie op EDA. Het onderzoek valt buiten de scope van dit project. Ondanks dat het onderzoek buiten de scope van het project valt moet niet vergeten worden dat het onderzoek de primaire reden is om het framework te ontwikkelen. Hierdoor moet tijdens de ontwikkeling van het framework het onderzoek altijd in het achterhoofd worden gehouden.

Het tweede doel van het framework is om de zorgen bij klanten weg te halen over EDA. Op dit moment wordt EDA niet gebruikt door de klanten omdat in een systeem dat hierop gebaseerd is de koppeling tussen onderdelen slecht te achterhalen is. Het framework moet een bijdrage leveren om te bewijzen dat deze koppelingen wel inzichtelijk te maken zijn. Dit zal gebeuren door nadat het framework gebouwd is een applicatie in het framework te ontwikkelen. Deze applicatie zal als voorbeeld dienen om te tonen dat de informatie stroom in beeld te brengen is bij EDA applicaties.

## 3.3 Requirements

Het project zal aan de volgende eisen moeten voldoen:

**9. Het framework moet de communicatie tussen modules regelen.**

De voornaamste taak van het framework is om de communicatie tussen modules te regelen. Een module is een onderdeel van het systeem dat op de hoogte van events wil worden gebracht. De communicatie tussen modules zal gebeuren met events. Modules kunnen events naar het framework sturen zodat andere modules deze events kunnen ontvangen.

**10. De events binnen het framework moeten tijdens de uitvoering van de applicatie in beeld worden gebracht.**

Om het onderzoek te steunen moet het mogelijk zijn om de events die binnen het framework plaatsvinden realtime weer te geven. Dit betekent dat tijdens de uitvoering van de applicatie de events die plaatsvinden weergegeven worden. Voor de huidige opdracht is het weergeven van de events in tekst vorm genoeg. In een vervolg opdracht zullen de events op een meer grafische manier worden weergegeven.

**11. Het moet mogelijk zijn om events te generen vanuit een beheerders-interface.**

Het moet mogelijk zijn om events te generen vanuit een beheerders-interface. Dit zorgt ervoor dat alle event die plaats kunnen vinden ook vanuit de beheerders-interface gestuurd kunnen worden. Deze mogelijkheid zorgt ervoor dat alle modules getest kunnen worden vanuit de beheerders-interface.

**12. Het moet mogelijk zijn om dynamisch modules in en uit te laden.**

In het framework moet het mogelijk zijn om modules dynamisch in te laden. Hiermee wordt bedoeld dat een library-bestand wordt ingeladen tijdens de uitvoering van de applicatie. Als het library-bestand een module bevat zal deze worden aangemaakt. Hierna kan deze module direct al events ontvangen en sturen. Met het uitladen van modules wordt bedoeld dat modules kunnen worden verwijderd tijdens de uitvoering van de applicatie.

**13. Het moet mogelijk zijn dat een module een eigen front end heeft.**

Het moet mogelijk zijn dat een module een eigen front end heeft. Hiermee wordt bedoeld dat achter een module een user interface zit. Hieronder vallen niet alleen WPF en FORM interfaces maar ook web-based interfaces moeten mogelijk zijn. Er moet rekening gehouden worden dat een module moet communiceren met processen die user interfaces aansturen.

**14. Het framework moet in de toekomst over verschillende systemen verspreid kunnen worden.**

In de toekomst moet het mogelijk zijn om het framework over verschillende systemen te verspreiden. Hiermee wordt bedoeld dat verschillende modules op verschillende systemen draaien. Dit is geen eis aan de huidige opdracht maar dit moet wel mogelijk zijn in de toekomst. Hier moet met het ontwerpen van het framework rekening gehouden worden.

**15. Het framework moet getest worden met behulp van unit tests.**

Het framework zal dienen als basis voor andere afstudeeropdrachten. Hierdoor is het van belang dat het framework volledig getest is. Het gehele project zal een test dekking van 90% moeten hebben.

**16. Het framework moet volledig gedocumenteerd zijn.**

Het framework zal dienen als basis voor andere afstudeeropdrachten. Hierdoor is het van belang dat het framework volledig gedocumenteerd is.

## 3.4 Hoofdopdracht

De hoofdopdracht zal bestaan uit het ontwikkelen van een framework. Dit framework moet het mogelijk maken om applicaties te maken die werken met EDA. Het framework zal de communicatie tussen de verschillende onderdelen in het systeem afhandelen. Het framework zal hiernaast ook de mogelijkheid bieden om de communicatie van de verschillende onderdelen in beeld te brengen. Het framework zal aan requirements die hierboven zijn gesteld moeten voldoen.

## 3.5 Deelopdrachten

Het project heeft naast de hoofopdracht de volgende deelopdrachten:

**4. Onderzoek naar de programmeertaal**

De eerste deelopdracht is een onderzoek naar de te gebruiken programmeertaal. Hierbij kan gekozen worden tussen C# en F#. Ik heb veel ervaring met C#, daarentegen heb ik weinig ervaring met F#. Bij het kiezen van de taal van het framework moet hier rekening mee gehouden worden.

Bij dit onderzoek moet er ook gekeken worden naar de mogelijkheid van gebruik van het actor model. Actors zijn een mogelijke manier om modules te implementeren. Voordat de taal gekozen wordt moet er onderzocht worden wat actors precies zijn en of actors gebruikt gaan worden in het framework. Als actors gebruikt gaan worden moet onderzocht worden wat de mogelijkheden zijn van de talen voor het gebruik van actors. Er is al bekend dat F# actors als klasse in de taal aanbiedt. C# heeft deze mogelijkheid niet.

Het onderzoek naar de programmeertaal met de resultaten zal in een onderzoeksverslag verwerkt worden.

**5. Bouwen van een applicatie in het framework**

Nadat het framework gebouwd is zal er een applicatie in framework gebouwd moeten worden. Deze applicatie zal als bewijs dienen dat het framework te gebruiken is en dat het mogelijk is om de communicatie tussen de verschillende onderdelen van het systeem in beeld te brengen. Het is nog niet bekend wat voor applicatie er gebouwd gaat worden in het framework. Aangezien de applicatie alleen als bewijs dient dat het framework werkt hoeft de applicatie niet volledig werkend te zijn.

**6. Tutorial voor het gebruik van het framework**

Er moet een tutorial geschreven worden voor het gebruik van het framework. Deze tutorial moet goed beschrijven hoe het framework te gebruiken is en hoe het framework intern werkt. Deze tutorial zal waarschijnlijk in de vorm van een tekst document gemaakt worden.

## 3.6 Project grenzen & randvoorwaarden

Het framework zal dienen als basis voor een onderzoek naar EDA. Het onderzoek is geen onderdeel van dit project. Het project bevat het ontwerpen, implementeren, documenteren en testen van het framework. Ondanks dat het onderzoek buiten de scope van het project valt is het wel de belangrijkste reden voor het maken van het framework. Hierom is het belangrijk om rekening te houden met het onderzoek.

Het moet mogelijk zijn om applicaties te ontwikkelen met behulp van het framework. Deze verschillende onderdelen van deze applicaties zullen communiceren doormiddel van het framework. Omdat het framework primair dient als hulpmiddel bij het onderzoek en niet als middel om daadwerkelijk applicaties te ontwikkelen ligt de focus hier niet bij. Om deze reden is het niet nodig dat applicaties die ontwikkeld zijn met behulp van het framework daadwerkelijk gedistribueerd kunnen worden. Hierdoor hoeft er weinig tot geen rekening gehouden worden met beveiliging en efficiëntie van het framework.

Het project grenst aan minstens nog één project. Dit aangrenzende project zal de events grafisch weergeven. Om het vervolg project te starten is het opvangen en weergeven van de events nodig. Dit zal gebeuren in het huidige project. Voor het huidige project is het voldoende om de events in tekst vorm weer te geven.

## 3.7 Op te leveren producten

De volgende producten moeten worden opgeleverd:

1. Het plan van aanpak.
2. Een onderzoeksverslag over het onderzoek naar de programmeertaal.
3. Een gedocumenteerd en getest framework dat aan de gestelde eisen voldoet.
4. Een applicatie gebouwd met behulp van het framework.
5. Een tutorial document hoe het framework te gebruiken is.
6. De afstudeerscriptie.

# 4 Projectactiviteiten & Planning

In dit hoofdstuk zullen de projectorganisatie, projectactiviteiten en de planning van deze activiteiten worden toegelicht. Ook de mijlpalen planning zal gegeven worden.

## 4.1 Project organisatie

Bij dit project zal er gebruik gemaakt worden van SCRUM als project methodiek. SCRUM is een agile project methodiek. Bij SCRUM wordt er gewerkt in sprints. Een sprint is een periode van 1 of 2 weken waarin een aantal doelen behaald zal worden. Er zal bij dit project gewerkt worden met sprints van 2 weken. Er zal nu kort uitgelegd worden hoe een sprint verloopt.

**Sprint planning**

Aan het begin van een sprint zal de sprint planning gehouden worden. Hierin zal bepaald worden welk werkt gedaan gaat worden in de sprint. Dit zal bepaald worden in overleg met Ralf Wolter.

**Daily scrum**

Bij Scrum is het de gewoonte dat er tijdens een sprint elke dag een daily scrum gehouden wordt. Dit is een vergadering aan het begin van elke dag waar wordt verteld wat iedereen de vorige dag gedaan heeft en wat hij die dag gaat doen. Aangezien het projectteam maar bestaat uit 1 lid zal dit niet nodig zijn voor dit project. Wel zal er elke dag bepaald worden welke taken er voltooid zullen worden die dag en gekeken worden hoeveel de dag ervoor gedaan is.

**Afsluiting sprint**

Aan het einde van elke sprint zal er kort gepresenteerd worden aan Ralf Wolter wat er die sprint gedaan is. Dit zorgt ervoor dat de heer Wolter weet hoe het project vordert en dit geeft hem de mogelijkheid om zo nodig het project een andere richting in te sturen.

## 4.2 Projectactiviteiten

De volgende activiteiten zullen plaats vinden tijdens het project:

**1. Plan van aanpak**

De eerste activiteit van het project was het schrijven van het plan van aanpak.

**2. Onderzoek naar de programmeertaal**

De tweede activiteit van het project is een onderzoek naar de te gebruiken programeertaal. Hierbij kan gekozen worden tussen C# en F#. Bij het onderzoek zal eerst gekeken worden naar de eisen waaraan de taal moet voldoen zodat het framework gebouwd kan worden. Hierna zal gekeken worden of C# en F# hieraan voldoen. Ook zal er gekeken worden naar de voor- en nadelen van het gebruik van C# en F#. Hierna zal onderzocht worden wat actors zijn en of deze gebruikt gaan worden voor het project. Als actors gebruikt gaan worden in dit project moet er onderzocht worden wat de mogelijkheden van F# en C# zijn met het gebruik van actors.

**3. Schrijven onderzoeksverslag**

Tijdens en na het onderzoek naar de programmeertaal zal er een onderzoeksverslag geschreven worden van het onderzoek. Dit onderzoeksverslag moet goed beschrijven wat onderzocht is en waarom er gekozen is voor de programmeertaal.

**4. Project omgeving voorbereiden**

Voordat er aan het framework gewerkt kan worden moet de omgeving opgezet worden. Hieronder vallen het opzetten van de mappenstructuur, het voorbereiden van de source control en het opzetten van de continuous integration build.

**5. Ontwerpen van het framework**

Nadat de project-omgeving is opgezet zal het framework ontwerpen worden. Hierbij zal de globale architectuur van het framework bepaald worden.

**6. Implementeren, documenteren en testen van het framework**

Nadat de globale architectuur van het framework is vastgelegd zal het framework geïmplementeerd, gedocumenteerd en getest worden. Het schrijven van de unit tests van het framework zal tegelijk gebeuren met het ontwikkelen van het framework. De documentatie zal ook tijdens de ontwikkeling geschreven en bijgewerkt worden. Tijdens de ontwikkeling van het framework zal het ontwerp verder uitgewerkt worden.

**7. Bouwen van een applicatie in het framework**

Nadat het framework grotendeels is gebouwd zal er een applicatie ontwikkeld worden in het framework. Deze applicatie zal gebruikt worden om aan te tonen dat het mogelijk is om EDA applicaties te ontwikkelen in het framework. Ook zal het als bewijs dienen dat het mogelijk is om de informatie stroom binnen een EDA applicatie in beeld te brengen. Er is nog niet bekend wat voor applicatie er ontwikkeld zal worden. De applicatie zal niet volledig werkend hoeven te zijn omdat het als bewijs moet dienen en niet daadwerkelijk gebruikt gaat worden. Tijdens het bouwen van de applicatie in het framework zal ook het framework verder uitgewerkt worden.

**8. Tutorial voor het gebruik van het framework**

Er moet een tutorial voor het gebruik van het framework gemaakt worden. De tutorial zal waarschijnlijk in de vorm van een tekst document worden gemaakt. Deze tutorial moet duidelijk uitleggen hoe het framework te gebruiken is.

**9. Schrijven afstudeerscriptie**

Nadat het framework is opgeleverd zal de afstudeerscriptie geschreven worden.

## 4.3 Activiteiten planning

Er zal gewerkt worden met sprints van twee weken. Dit is ook terug te vinden in de planning. Er zal per sprint bepaald worden wat er gedaan gaat worden. De huidige planning is hierom globaal en zal waarschijnlijk veranderen tijdens het project.

| | Week -> | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Activiteit** | Sprint -> | | | 0 | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | |
| Plan van aanpak | | 30 | 20 | 10 | | | | | | | | | | | | | | |
| Onderzoek naar programmeertaal | | 10 | 20 | 10 | | | | | | | | | | | | | | |
| Onderzoeksverslag schrijven | | | | 10 | 10 | | | | | | | | | | | | | |
| Omgeving voorbereiden | | | | 10 | 10 | | | | | | | | | | | | | |
| Ontwerpen van het framework | | | | | 20 | 10 | | | | | | | | | | | | |
| Bouwen van het framework | | | | | | 25 | 35 | 15 | 15 | 15 | 15 | 15 | 15 | 20 | | | | |
| Bouwen van een applicatie in the framework | | | | | | | | 20 | 20 | 20 | 20 | 20 | 20 | | | | | |
| Tutorial voor gebruik van het framework | | | | | | | | | | | | | | 15 | 10 | | | |
| Scriptie | | | | | | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 30 | 40 | | |
| Uitloop | | | | | | | | | | | | | | | | | | |

## 4.4 Mijlpalen planning

| | **Geplande datum** | **Uiterlijke datum** |
|---|---|---|
| PVA ondertekenen | 4 maart | 14 maart |
| PVA Inleveren afstudeercommissie | 8 maart | 15 maart |
| Onderzoeksverslag opleveren | 1 maart | 8 maart |
| Opleveren applicatie gemaakt in het framework | 27 april | 17 mei |
| Opleveren framework | 3 mei | 24 mei |
| Scriptie inleveren | 20 mei | 28 mei |

# 5 Hoofdstuk 5: Risico's

In dit hoofdstuk zullen de risico's van het project worden besproken.

| Risico | Toelichting | Maatregel |
|---|---|---|
| **Tijdsnood** | De producten komen niet op tijd af waardoor de einddatum in gevaar komt. Bijvoorbeeld door ziekte of verkeerde inschatting tijd. | Door twee buffer weken in de planning op te nemen. |
| **Verkeerd gecommuniceerd** | Er is een verkeerde opvatting over wat gedaan moet worden. | Doordat er elke twee weken tijdens de sprint planning samen met de heer Wolter bepaald wordt wat er gedaan gaat worden is er ook gelijk een controle of de opdracht nog in de goede richting gaat. Aan het einde van de sprint zal er aan de hand van een presentatie worden verteld wat er die sprint gedaan is. Dit is weer een moment waarop gecontroleerd zal worden of de opdracht nog in de goede richting gaat. |

# 6 Methodes

In dit hoofdstuk zullen de methodes die worden gebruikt in dit project worden toegelicht. Als eerste zal de te gebruiken kwaliteitsbewaking besproken worden. Hierop volgen de tools die worden gebruikt tijdens het project.

## 6.1 Kwaliteitsbewaking

Om de kwaliteit van de code te bewaken zal er gebruik gemaakt worden van unit tests, source control en van een continuous integration server. De unit tests zorgen ervoor dat er zekerheid is dat de code doet wat het moet doen. De source control zorgt ervoor dat de code niet kwijt kan raken en dat er terug kan worden gegaan naar een vorige versie. De continuous integration server zal de code automatisch bouwen en testen. Hierdoor is er snel te zien als de code gebroken is.

## 6.2 Tools

**Redmine**

Redmine is de projectmanagement-tool die gebruikt wordt bij Sogyo. Deze tool zal gebruikt worden om de sprints in te plannen en de back log bij te houden.

**Subversion**

Subversion zal gebruikt worden als source control bij dit project. Sogyo heeft een Subversion server die hiervoor gebruikt kan worden.

**Jenkins**

Jenkins is de continuous integration server die gebruikt zal worden voor dit project. Jenkins zal ervoor zorgen dat elke keer dat de code op Subversion wordt gezet de code gebouwd zal worden en de unit tests uitgevoerd worden. Dit zorgt ervoor dat het snel duidelijk is als de code gebroken is. Nadat de code gebouwd is en de unit test uitgevoerd zijn zal Ncover uitgevoerd worden om de unit test dekking te bepalen.

**Visual studio 2010**

Het programeerwerk zal gebeuren in Visual Studio 2010. Visual Studio 2010 biedt de mogelijkheid om in F# en C# te werken en is hierom zeer geschikt.

# 7 Persoon/bedrijf gegevens

**Gegevens student:**
Voornaam: Tim
Achternaam: Linschoten
Telefoon: 06 29527231
E-mail: tim.linschoten@student.hu.nl

**Gegevens Docentbegeleider:**
Voornaam: Peter
Achternaam: van Rooijen
E-mail: peter.vanrooijen@hu.nl

**Gegevens contactpersoon:**
Voornaam: Brenda
Achternaam: Klever
Functie: HR-consultant
Telefoon: 030 2202216
E-mail: bklever@sogyo.nl

**Gegevens bedrijfsbegeleider:**
Voornaam: Ralf
Achternaam: Wolter
Functie: Senior engineer
Telefoon: 030 2202216
E-mail: rwolter@sogyo.nl

**Gegevens Bedrijf**
Naam: Sogyo
Voertaal: Nederlands
Website: http://www.sogyo.nl/
Adres: Landgoed Sandwijck
Utrechtseweg 301
3731 GA De Bilt
Aantal medewerkers: +-80
Aantal medewerkers werkzaam op dit adres: +-40

## 7.1 Bijlage 2: Persoonlijke evaluatie van het eigen functioneren

Hierop volgt een korte persoonlijke evaluatie over mijn eigen functioneren tijdens de afstudeerstage.

Als ik terug kijk op de stage dan ben ik erg tevreden. Alle opdrachten zijn op tijd afgemaakt en opgeleverd. Hiernaast ben ik erg tevreden met de uiteindelijke producten die ik oplever. Het framework voldoet niet alleen aan de benodigde eisen maar is ook nog gemakkelijk in gebruik. Hierdoor voelt het framework ook echt als een framework aan.

Bij de stage is gebruik gemaakt van Scrum. Dit zorgde ervoor dat de stage opgedeeld was in sprints. Ik zelf vond dit erg fijn werken. De eerste paar sprints zijn gebruikt om het framework de benodigde functionaliteit te geven. De iteraties erna zijn gebruikt om het framework te verbeteren. Het iteratieve werken heeft ervoor gezorgd dat het framework elke sprint beter geworden is. De sprints zorgden er ook voor dat er elke sprint een onderwerp gekozen kon worden waarop geconcentreerd kon worden. Als voorbeeld kan gedacht worden aan de applicatie in het framework. Hiervoor is één sprint gebruikt. Omdat tijdens de sprint alleen rekening gehouden hoefde te worden met de onderwerpen van die sprint bleven mijn gedachten goed geordend en wist ik precies wat ik moest doen. Ik geloof echt dat het werken met Scrum ervoor heeft gezorgd dat de opdracht beter is verlopen dan wanneer ik met een andere projectmethodiek gewerkt had.

Tijdens de afstudeerstage is er niet iets groots misgegaan. Hierdoor zijn er geen grote momenten aan te wijzen waarbij ik mijn werkwijze heb moeten veranderen. Er kan dus gezegd worden dat ik tijdens de stage geen grote leermomenten als gevolg van het herstellen van gemaakte fouten heb gehad. Wel heb ik gezien dat beslissingen die ik genomen heb tijdens de stage goed zijn uitgepakt.

De eerste beslissing die goed uitgepakt is was dat ik extra tijd had ingepland om aan de verslagen te werken. Ik wist van tevoren dat het schrijven van de verslagen niet mijn sterkste kant was en heb daarom hier meer tijd voor ingepland. Dit heeft ervoor gezorgd dat ik mijn scriptie en plan van aanpak vaker heb kunnen laten nakijken en hierdoor vaker heb kunnen verbeteren. Hierdoor geloof ik sterk dat de verslagen van hogere kwaliteit zijn dan wanneer dit niet gedaan was.

Een andere beslissing die goed uitgepakt is was het kiezen voor F# als programmeertaal voor het framework. Het kiezen van de programmeertaal voor het framework was vroeg in de stage gedaan. Hiervoor is een onderzoek gedaan maar tijdens dit onderzoek was het niet zeker te weten of dit ook echt werkelijk goed zou uitpakken. Uiteindelijk zijn er geen problemen naar voren gekomen met het werken met F# en zijn er een aantal features naar voren gekomen die erg handig waren. Hiernaast ben ik erg tevreden dat ik ervaring heb opgedaan met een programmeertaal die ik niet kende.

## 7.2 Bijlage 3: Incasso case specificatie

# Incasso case

Incassobureau 'Gimme the Money' is een jong, nieuw bedrijf met een andere kijk op incasseren van rekeningen. Met behulp van onze beproefde methode van 'vragen, waarschuwen, handelen' hebben we een hoge betalingratio voor onze opdrachtgevers gerealiseerd. Om nog meer bang-for-the-buck te creëren, willen we dit proces verder automatiseren.

Van het versturen van de eerst correspondentie tot en met het inschakelen van onze deurwaarders moet volledig georganiseerd kunnen worden vanuit de nieuw te bouwen applicatie. Onze klanten moeten dus elektronisch de gegevens kunnen aanleveren, waarna we automatisch de eerste brieven met gespecificeerde rekening de deur uit moeten laten gaan.

Als cliënten op tijd betalen is er niks aan de hand en moet in de database verwerkt worden dat ze betaald hebben. Zo niet, dan gaat de waarschuwing de deur uit naar ze.

Hebben ze de waarschuwing ook niet op tijd door, dan wordt het tijd voor de deurwaarders. Gelukkig hoeven we dat niet al te vaak te doen, de dreiging er mee is gelukkig al genoeg om de meeste mensen snel te doen betalen.

### Acceptatie

Hier bij acceptatie nemen we de aanleveringen van onze klanten in behandeling. De aanleveringen bestaat uit een verzameling rekeningen die we invoeren. Als er bij een rekening bij zit van een debiteur die we dubieus achten accepteren we die rekening niet. Ons grootste werk is het koppelen van de rekeningen aan de debiteuren. Om het credit rapport van de debiteur goed te houden, moeten we namelijk zorgen dat alle rekeningen van dezelfde persoon ook onder de zelfde debiteur opgeslagen worden.

### Financiën

Financiën is het belangrijkste onderdeel van ons bedrijf. Hier gaan we namelijk om met het geld. Onze grootste taak is verwerken van de betalingen die binnen komen, deze moeten gekoppeld worden aan een uitstaande rekening. De meeste debiteuren vullen het kenmerk in, dat maakt het koppelen eenvoudig, maar het gebeurt ook dat dit vergeten wordt en dan moet op naam, woonplaats en bedrag gekoppeld worden. Nadat een aanlevering is geaccordeerd word 50% van het totale bedrag als voorschot uitgekeerd.

Ook betalen we de crediteuren als een rekening betaald wordt. Hier wachten we meestal een week mee om nog wat rente te trekken en daarna maken we het binnengekomen bedrag van de rekeningen naar de crediteur over. Voor verstuurde herinneringen brengen we nog 2 euro per herinnering in rekening. Open staande rekeningen van minder dan drie euro boeken we meteen af. Indien er een deurwaarder langs gestuurd moet worden, markeren we de debiteur als dubieus en accepteren we geen verdere schulden meer van deze debiteur.

## Debiteuren beheer

Het debiteuren beheer is de kern van het bedrijf. Hier onderhouden we het contact met de schuldenaars. We sturen de rekeningen en herinneringen en zetten eventueel de deurwaarder in. Ook staan we de schuldenaars telefonisch te woord. Dit gaat meestal over dubbele rekeningen of niet verwerkte betalingen.

In het geval van een deurwaarder worden de kosten van dit proces op de schuldenaar verhaald. Dit is op dit moment 135 euro en 5% van de totale som.

Ook kan het zijn dat schuldenaar niet kunnen betalen. Dan kan een betaal regeling opgesteld worden. Dit kan uitstel zijn of betaling in termijnen. Voor uitstel wordt er in overleg een periode vast gesteld tot maximaal een half jaar. Hiervoor worden kosten van 2,5 % in rekening gebracht. Betaling in termijnen kan in maximaal 12 termijnen van een maand en gaan de volgende kalender maand in. Dit kost 5% van de originele som.