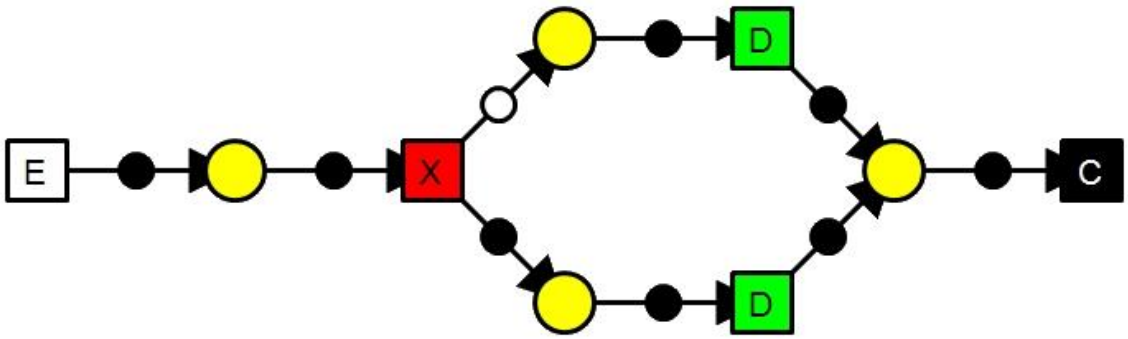
Petrinet Editor

Scriptie

Naam:	Max Aalderink
Studentennummer:	1520641
Email:	maxaalderink@gmail.com
Afstudeerbegeleider:	Joop Kaldeway
Bedrijf:	Flexposure
Bedrijfsbegeleider:	Wouter Lindehof
Documentversie:	1.0
Datum:	19-12-2012



HOGESCHOOL UTRECHT

Voorwoord

Deze scriptie gaat over het resultaat van mijn afstudeer stage project dat plaats vond van Augustus 2012 tot Januari 2013 bij Flexposure.
Met deze scriptie rond ik mijn bachelor opleiding Technische Informatica bij de Hogeschool Utrecht af.

Ik wil graag iedereen bedanken die het voor mij mogelijk hebben gemaakt om deze scriptie te schrijven. Ten eerste wil ik graag mijn begeleiders bedanken, Wouter Lindenhof van Flexposure en Joop Kaldeway van de Hogeschool Utrecht. Zonder de begeleiding en feedback die ik van hun heb gekregen was deze scriptie niet mogelijk.
Ook wil ik graag Dieter Lentjes bedanken die als Operationeel Manager van Flexposure de stage positie beschikbaar heeft gesteld en Mathijs Hendriks van StudentenBureau voor het helpen met het vinden van deze stage plaats.

Ook wil ik graag mijn waardering uiten naar Flexposure voor het voorzien van een erg fijne werkplaats en omgeving met alle fijne collega's die daar bij horen. Met name de collega's van R&D die mij maar al te graag te hulp zijn geschoten wanneer nodig wil ik bedanken, in het bijzonder, Wouter, Quinten, John en Gert-Jan.

Tot slot wil ik graag de Hogeschool Utrecht en mijn klasgenoten bedanken voor de fijne ervaringen tijdens de opleiding die ik heb gevolgd ten aanleiding van deze scriptie.
In het bijzonder Arno Derks en Marleen Schilt die mij zeer nuttige feedback hebben gegeven en hebben geholpen met deze scriptie.

Utrecht, 19 December, 2012

Max Aalderink

Afkortingen

Hieronder zijn de verschillende afkortingen inclusief de betekenis hiervan uitgeschreven die in deze scriptie gebruikt zal worden.

SMA	Security Management Application
WFM	WorkFlow Management
WML	Workflow Modeling Language
CCTV	Closed Circuit TeleVision
CCD	Charge Coupled Device
UML	Unified Modeling Language
XML	Extensible Markup Language
DLL	Dynamicly-Linked Library
R&D	Research & Development

Management Samenvatting

In dit hoofdstuk wordt een korte samenvatting gegeven over het verloop van mijn stage vanuit een management oogpunt. Hierin zal ik niet in gaan op technische details of de achtergrond behandelen die later wel in deze scriptie aan bod komen.

Opdrachtgever/Opdrachtnemer

Ik, Max Aalderink, ben een vierdejaars student Technische Informatica aan de Hogeschool Utrecht. Flexposure is een bedrijf dat Security Management Applications (SMA) levert voor grote bedrijven zoals de Amsterdam Arena en Q-Park parking. Denk hierbij aan het managen van de intercoms, centraal aansturen van beveiliging camera's of het automatiseren van toegang of alarmen.
Ik heb gekozen om bij Flexposure mijn afstudeer stage te volgen omdat ik veel affiniteit heb met domotica (thuis automatisering). Hoewel het automatiseren van beveiliging niet exact hetzelfde is als automatisering in huis komt het in grote lijnen wel overeen. Ook denkt Flexposure er aan om de markt op te gaan voor particulieren.

Doel van het project

Voor het bepalen van de logica van de SMA FlinQ 4.0 wordt de WorkFlow Language (WFL) Petrinet gebruikt. Hiervoor bestaat alleen nog geen gebruikersvriendelijke editor die op maat gemaakt is voor de functies van het product van Flexposure.
Het doel van dit project is dus om die editor te maken zodat de medewerkers van Flexposure gemakkelijker op maat gemaakte logica kunnen creëren voor hun klanten.

Omgeving van het project

Tijdens mijn stage heb ik gewerkt binnen de R&D tak van het bedrijf. Deze bestaat uit drie andere programmeurs en één business relationship manager/tester allemaal in dezelfde kantoorruimte. Binnen deze ruimte had ik beschikking tot mijn eigen bureau met computer en twee monitoren.
Tevens is er een gedeelde opslag server waar onder andere een (Mecurial) version control server op draait voor het bijhouden van wijzigingen in de code.

Het hele bedrijf bestaat uit ongeveer 20 man. Gezien het formaat van het bedrijf is de sfeer erg open en is er geen grote complexe hiërarchische structuur.
Omdat de software waarvoor ik een tool moet gaan maken nog flink in ontwikkeling is en we met een klein team werken wordt SCRUM toegepast zodat we snel en flexibel kunnen plannen.

Activiteiten

Omdat SCRUM toegepast werd was er geen strakke planning die gevolgd moest worden. In grote lijnen zijn er drie stadia geweest. Het eerste stadium was het onderzoek en prototype stadium waarin ik onderzocht hoe het huidige product werkt en wat er mogelijk is voor de editor.

Het tweede stadium bestaat uit korte sprints van 2 tot 4 weken waarin een aantal toevoegingen of verbeteringen in gerealiseerd moeten worden. Aan het begin van deze sprints is er een lang overleg over de planning waarin ik sterk betrokken wordt en invloed op kan uitoefenen. Tussentijds overleg en wijzigingen in de doelstellingen van de sprint zijn ook niet zeldzaam.
De planning wordt bijgehouden in een gedeeld Google Docs document waarin de status, commentaar, geschatte benodigde uren en wie deze taak is toebedeeld staat.
Documentatie van wijzigingen in de code gebeurt aan de hand van commentaar bij commits naar de version control server.
In het derde stadium zijn alle van te voren vastgestelde eisen van de editor voltooid en wordt de editor getest door het publiek waarvoor het bedoeld is, de service afdeling. Een korte handleiding zal geschreven worden die zowel de WFL Petrinet als de specifieke functies van de editor uitleggen. Alle bugs die aangegeven worden of commentaar dat gegeven wordt zal worden besproken en verwerkt.

Producten

Voor het bedrijf:
-De Petrinet editor voor FlinQ 4.0 werkend onder Microsoft Windows
-Een instructie gids voor de editor
-De software voor de FlinQ 4.0 server om de output van de editor in te lezen

Voor school:
-Plan van aanpak
-Een scriptie over mijn stage opdracht
-Een presentatie over mijn stage opdracht

Conclusies

Achteraf kan ik zeggen dat ik tevreden ben met het eind resultaat. De editor voldoet aan de eisen en ik ben zeer te spreken over de samenwerking met het team en hoeveel zij mij met de keuzes in de ontwikkeling hebben betrokken.
Hoewel ik erg tevreden ben over het eind product heb ik wel commentaar op de werkwijze. Er is goed te merken dat de collega's gewend zijn om in een zeer klein team te werken en vanuit hun opleiding niet gewend zijn om gedetailleerde UML schema's of planning te maken zoals ik dat wel ben.
Achteraf kan ik zeggen dat het beter was geweest, voor zowel de R&D afdeling als mijzelf, als ik meer het touw in eigen handen had proberen te nemen en voor had gesteld dat er meer UML diagrammen van te voren gemaakt zouden worden.

Inhoudsopgave

1. Inleiding en Context

In dit hoofdstuk zal kort uitleg gegeven worden over de branche waarin Flexposure werkt, zal omschreven worden wat het bedrijf precies zelf doet en zal ik mijn taak binnen het bedrijf beknopt uitleggen. Daar aan vooraf een korte uitleg over de structuur van deze scriptie

1.1 Structuur van deze scriptie

Vooraf aan dit hoofdstuk is een korte samenvatting gegeven over dit project vanuit een management perspectief. Dit fungeert tevens als samenvatting van deze scriptie.

In dit hoofdstuk wordt een beeld geschept van de branche en de producten van Flexposure. Hiermee wordt duidelijker gemaakt waarom dit project gedaan moet worden.

In hoofdstuk 2 is het Plan van Aanpak verwerkt dat een paar weken na de start van de afstudeerstage is gemaakt om het doel en de planning van het project duidelijker vast te stellen. Ook wordt hier uitgelegd wat een Petrinet is.

In hoofdstuk 3 wordt de uiteindelijk gekozen aanpak besproken na aanleiding van het eerste prototype en de grotere beslissingen die genomen zijn.

In hoofdstuk 4 zijn de meer specifieke eigenschappen en design beslissingen behandeld.

Hoofdstuk 5 is de persoonlijke evaluatie over de verloop van het afstudeerproject.

Tot slot is in hoofdstuk 6 de literatuurlijst te vinden en in hoofdstuk 7 de gegevens van het bedrijf, de begeleiders en de student.

1.2 Branche

Dankzij de groeiende behoefte voor veiligheid vanuit de samenleving is er een steeds grotere vraag naar bewaking en surveillance van verschillende omgevingen. Gebaseerd op deze behoefte is de markt voor Sercurity Managment Applications (SMA) tot stand gekomen.

De evolutie SMA’s begon met op enkel video gebaseerde surveillance systemen die analoge technieken gebruikte, in het bijzonder Closed Circuit Television (CCTV). Deze systemen, welke tegenwoordig worden beschouwd als de eerste generatie surveillance systemen, bestonden uit een aantal camera’s op verschillende plaatsen en waren verbonden met een aantal monitoren die vaak geplaatst werden in een enkele, centrale controle ruimte.

In latere jaren gingen camera's vaker de digitale techniek Charge Coupled Device (CCD) gebruiken om beelden te registreren en werden analoge technieken alleen nog gebruikt om de beelden te distribueren en op te slaan.

Met de toenemende kracht van volledig digitale systemen die door computers aangestuurd worden konden naast het simpelweg opnemen van beeld nog meer geavanceerde features gerealiseerd worden. Voorbeelden hiervan zijn nummerplaat detectie en real-time event afhandeling. Deze technologische verbeteringen hebben geleid tot de ontwikkeling van semiautomatische en intelligente systemen, nu bekend als de tweede generatie surveillance systemen. Deze zijn nog steeds voornamelijk gebaseerd op video.

Een hedendaagse SMA wordt gerekend tot de derde generatie surveillance systemen. Deze zijn ontworpen om meerdere clients aan te kunnen sturen en om een groot aantal camera's en andere sensoren of actoren aan te kunnen sturen. Ook wordt er een intuïtieve user interface verwacht.
Om dit te realiseren is een autonoom en intelligent systeem nodig dat tevens schaalbaar is naar de wensen van de gebruiker. Daar bovenop is vereist dat er één universele, geïntrigeerde oplossing is voor het bedienen van vele soorten sensoren en actoren die met verschillende protocollen werken.

Voorbeelden hiervan zijn;

- pas-lezers
- bewegingssensoren
- brandmelders
- hefbomen
- intercom
- dome-camera's

Hedendaagse SMA's doen veel meer dan simpel en passief monitoren en beslissingen over laten aan een operator. Automatisering die minimale menselijke interactie vereist wordt verwacht. Hiernaast moet een SMA flexibel genoeg zijn om met zeer verschillende omgevingen te werken en om aan de continu veranderende eisen van de klant te voldoen.

Om deze intelligentie of logica te realiseren en toch flexibel te zijn moet een WorkFlow Managment (WFM) systeem geïmplementeerd worden waarin de logica van het systeem wordt gedefinieerd buiten de code van de SMA server om. Hiermee kan dus in theorie overal dezelfde SMA gebruikt worden met specifieke WFM logica per klant. Door deze scheiding is ontwikkelen en onderhouden van de applicatie veel gemakkelijker en sneller uit te voeren.

1.3 Bedrijf

Flexposure is opgestart in 2002 en bestond toen der tijd slechts uit 3 man die vanuit huis werkte. Toen leverde het bedrijf naast security oplossingen ook applicaties voor narrow casting. Narrow casting is het tonen van bepaalde informatie onderaan het scherm bij video beelden in de vorm van een lichtkrant. Dit is vergelijkbaar zoals ook bij het nieuws wordt gedaan alleen werd het hier toegepast om statistieken over het gebouw te tonen of bijvoorbeeld de namen van de personen in de vergaderruimte. Flexposure levert niet langer narrow casting applicaties vanwege een gebrek aan vraag hiernaar.

Vandaag de dag biedt Flexposure totaal oplossingen in de vorm van een SMA voor bedrijven en zelfs binnenkort particulieren. De laatste versie van het product, genaamd FlinQ 4.0, is nog sterk in ontwikkeling en bied een 3D weergave waarin de verschillende gebouwen worden weergegeven en waarin een duidelijk herkenbaar overzicht van de actoren en sensoren te zien is. Hier kan bijvoorbeeld bij het af gaan van een alarm ingezoomd worden op het betreffende gebouw en verdieping waar dan een plattegrond wordt getoond met daarin het device dat het alarm in gang heeft gezet. Hier kan de gebruiker kiezen om bijvoorbeeld het alarm tijdelijk uit te zetten of de camera beelden op te vragen van die verdieping.

Promotie foto van FlinQ 4.0

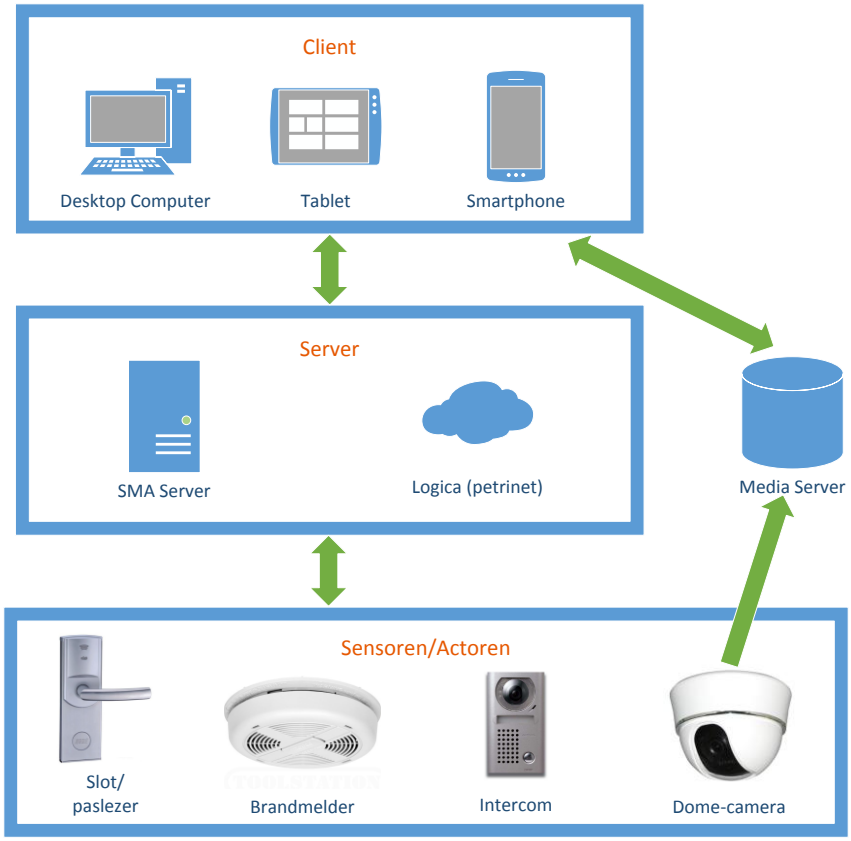
In de afbeelding hieronder is een schema getoond met een voorbeeld configuratie van een SMA gemaakt rondom de FlinQ 4.0 server. De server in het midden van het schema bestaat uit een universele applicatie en een set logica componenten die verantwoordelijk is voor het WFM. De server draait als applicatie op een desktop OS en is platform onafhankelijk opgezet.

De operator bedient de SMA via een client die beschikbaar zal zijn op een desktop computer onafhankelijk van het OS én tablet of smartphone met Apple iOS of Google Android.

Omdat de vele verschillende actoren en sensoren ieder een eigen protocol gebruiken om te communiceren wordt hier tussen een abstractie laag toegevoegd in de vorm van een uniek stuk software welke de vele protocollen vertaald naar de taal die de FlinQ 4.0 server spreekt. Dit is vergelijkbaar met drivers voor de externe hardware van een desktop computer.

Zowel de communicatie van client naar server als die van server naar actoren/sensoren verloopt via zogenaamde Tokens. In deze Tokens staat gespecificeerd welk device er aangesproken moet worden, waar het vandaan komt en wat de actie is die gedaan moet worden of welk event er te melden is.

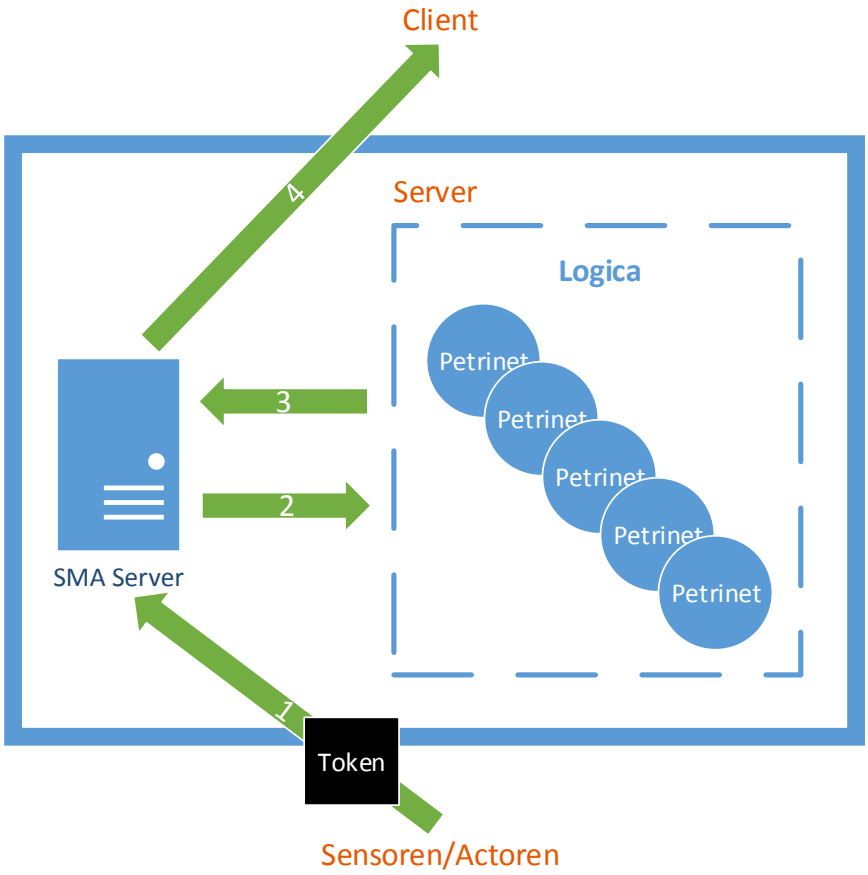
Er is ook nog een communicatie lijn die niet via de FlinQ 4.0 Server verloopt of Tokens gebruikt en dat is tussen camera's en de Media Server en tussen de Media Server en een client.
De Media Server is direct verbonden met de camera's om beelden meteen op te kunnen slaan. Ook kan een client direct met de Media Server communiceren om huidige of oude beelden op te vragen.



Een visuele weergave van de verschillende onderdelen van FlinQ 4.0

Het verwerken van deze Tokens gebeurd in de logica die Petrinets genoemd zijn. In deze logica wordt voor elk event of handeling bepaald welke actie ondernomen moet worden.
Een voorbeeld hiervan is dat er een notificatie wordt getoond op de client in het geval dat er een brandalarm afgaat. Een ander voorbeeld is wanneer er bij een autogarage een auto naar buiten rijdt het bord buiten de garage van “Bezet” in “1 Plaats vrij” veranderd. Voor de structuur van deze events en de acties die daar op volgen is gekozen voor een Petrinet formulering.

Om het voorbeeld van de alarm notificatie nog iets beter uit te leggen is de afbeelding hieronder gebruikt. Bij pijl #1 wordt een token verstuurd van een sensor, in dit geval dus een brandalarm, naar de SMA in de Server. De SMA kiest hierna het juiste Petrinet uit de logica om de Token te verwerken bij pijl #2. Tijdens het verwerken van de Token in het Petrinet wordt opdracht gegeven om een nieuw Token te versturen naar de client, zie pijl #3 en #4.



Verwerking van een Token in een FlinQ 4.0 Server

Wat Petrinet precies is wordt in het volgende hoofdstuk uitgelegd.

1.4 Opdracht

De opdracht zal uitgevoerd worden als onderdeel van de R&D afdeling van Flexposure. Deze afdeling bestaat uit drie programmeurs en één coördinator.

De stageopdracht houdt in dat een visuele editor gemaakt moet worden om deze Petrinets te creëren. Het doel hierachter is om het maken van deze logica toegankelijker te maken voor de werknemers buiten de R&D afdeling. Ook moet de editor automatisch de juiste opties tonen afhankelijk van de mogelijkheden van de FlinQ 4.0 server waarvoor een Petrinet gemaakt wordt.

In het volgende hoofdstuk wordt de opdracht in groter detail uitgelegd.

2. Plan van Aanpak

In dit hoofdstuk is het Plan van Aanpak verwerkt dat vlak na de start van het afstudeer project is geschreven. Hierin wordt de motivatie achter de opdracht en de achtergrond hiervan uitgelegd. Ook is de initiële aanpak en planning beschreven en worden de vereiste competenties en risico factoren vast gelegd.
Er zijn hoofdstukken weg gelaten en kleine wijzigingen toegepast in dit hoofdstuk ten opzichte van het originele Plan van Aanpak om de leesbaarheid van deze scriptie te verbeteren of omdat de situatie is gewijzigd sinds de start van het project.

2.1 De opdracht aanleiding

Hoewel het meeste werk aan de server en client delen van het nieuwe software pakket voltooid is moet er voor elke klant specifieke logica geschreven worden. Voor deze logica is gekozen voor een Petrinet formulering.
Het schrijven van Petrinets is een tijdrovend, complex en fout gevoelig werk dat op dit moment vooral door de R&D afdeling en enkele, meer ervaren, mensen van de service afdeling wordt gedaan. Om dit toegankelijker te maken voor non-programmeurs is het de bedoeling dat er een grafische editor komt om deze Petrinets te creëren.
Hierdoor wordt de werk last van de R&D afdeling verlicht en is de learning curve lager geworden voor andere medewerkers. Er wordt ook aan gedacht om in de toekomst, als de editor helemaal is door ontwikkeld, deze ook aan de klant te leveren zodat deze intern nieuwe Petrinets kan creëren.

Er is gekozen voor de WML Petrinet omdat hiermee op een visuele manier een proces getoond kan worden en deze taal relatief weinig en simpele regels heeft. Ook is Petrinet een open standaard en heeft dus geen licentie kosten.

2.2 Werking Petrinet

Een Petrinet, ook bekend als Place/Transition of P/T net, is oorspronkelijk bedacht in 1939 om een chemisch proces te omschrijven.
Een Petrinet bestaat uit de volgende basis componenten en heeft de volgende regels.

- Place 

  Een Place kan nul of meer Tokens bevatten en representeert passief een system state.
  Grafisch worden deze aangegeven met een cirkel.

- Transition 

  Een Transition representeert een activiteit of een proces en is actief.
  Grafisch worden deze aangegeven met een vierkant.

- Arc 

  Arcs verbinden een Place met een Transition en andersom maar nooit tussen twee Places of twee Transitions.
  Arcs worden grafisch weergegeven met een pijl.

- Token 

  Tokens zijn de objecten die verstuurd worden door een Petrinet. In de context van een chemisch proces stellen dit moleculen voor welke opgesplitst of samengevoegd worden tot nieuwe moleculen tijdens een Transition.
  In de context van WFM is dit een event message die binnen een Petrinet mogelijk nieuwe Tokens genereert om bepaalde taken uit te voeren.
  Grafisch worden deze weergegeven met een zwarte stip.

- Emitter 

  Een Emitter is een speciaal soort Transition die het punt aangeeft waar de Tokens in het Petrinet worden geïntroduceerd.
  Grafisch zijn deze te onderscheiden van een gewone Transition door de letter ‘E’ die hierop is afgebeeld.

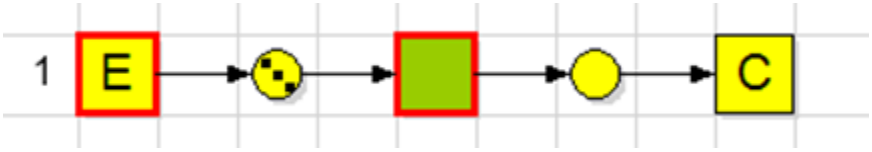
- Collector 

  Een Collector is een speciaal soort Transition die het punt aangeeft waar de Tokens het Petrinet verlaten.
  Grafisch zijn deze te onderscheiden van een gewone Transition door de letter ‘C’ die hierop is afgebeeld.

Het idee is dat een zo geheten Token wordt geïntroduceerd in het Petrinet via de Emitter waarna deze zich verplaatst van naar een Place. Vanaf hier verplaatst de Token zich door een Transition om vervolgens weer bij een Place terecht te komen. Naar welke Place of Transition er wordt verplaatst is afhankelijk van de Arcs die de Transitions en Places verbind. Zodra de Token de Collector heeft bereikt wordt deze verwijderd.
Belangrijk om te vermelden is dat zich meerdere Tokens tegelijkertijd in een Petrinet kunnen bevinden en ook meerdere op één Place. Een Token gaat altijd 'door' een Transition, waar de inhoud van de Token mogelijk aangepast wordt, om vervolgens te 'rusten' bij een Place.

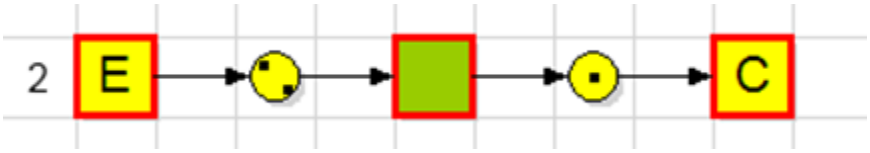
Dit is vergelijkbaar met hoe een chemische reactie verloopt. De Emitter stelt het moment voor dat bepaalde moleculen, dus Tokens, geïntroduceerd worden in bijvoorbeeld een Erlenmeyer beker. Deze moleculen bevinden zich op dit moment in stabiele staat, dus op een Place. Zie afbeelding 1 hieronder.

Opmerking: In de afbeeldingen hieronder zijn de vierkanten die de Transitions voorstellen soms rood omrandt. Dit is binnen de editor waar deze afbeeldingen vandaan komen om aan te geven dat deze Transitions een token kunnen genereren of opnemen.



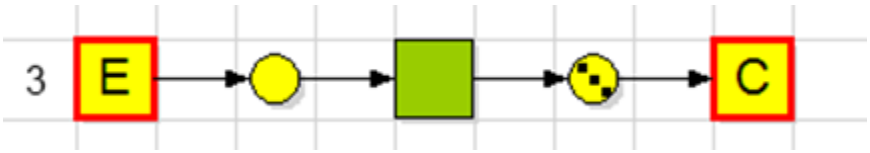
Petrinet afbeelding1

Nu de beker met de moleculen verhit wordt ondergaan deze moleculen een verandering, dit gebeurd in de Transition. Zie afbeelding 2 hieronder.



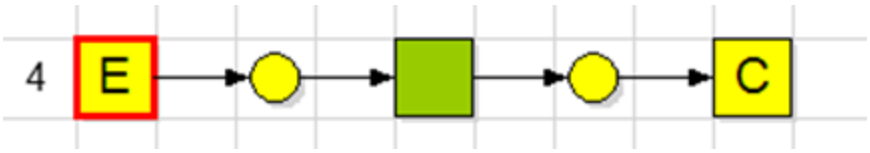
Petrinet afbeelding 2

Eén voor één ondergaan deze moleculen deze verandering en worden omgevormd in andere moleculen die weer in stabiele staat verkeren op de Place na de Transition. Zie hieronder afbeelding 3.



Petrinet afbeelding 3

Nadat de moleculen zijn verwerkt worden deze verzameld door de Collector en verwijderd uit het Petrinet. Zie afbeelding 4 hieronder.

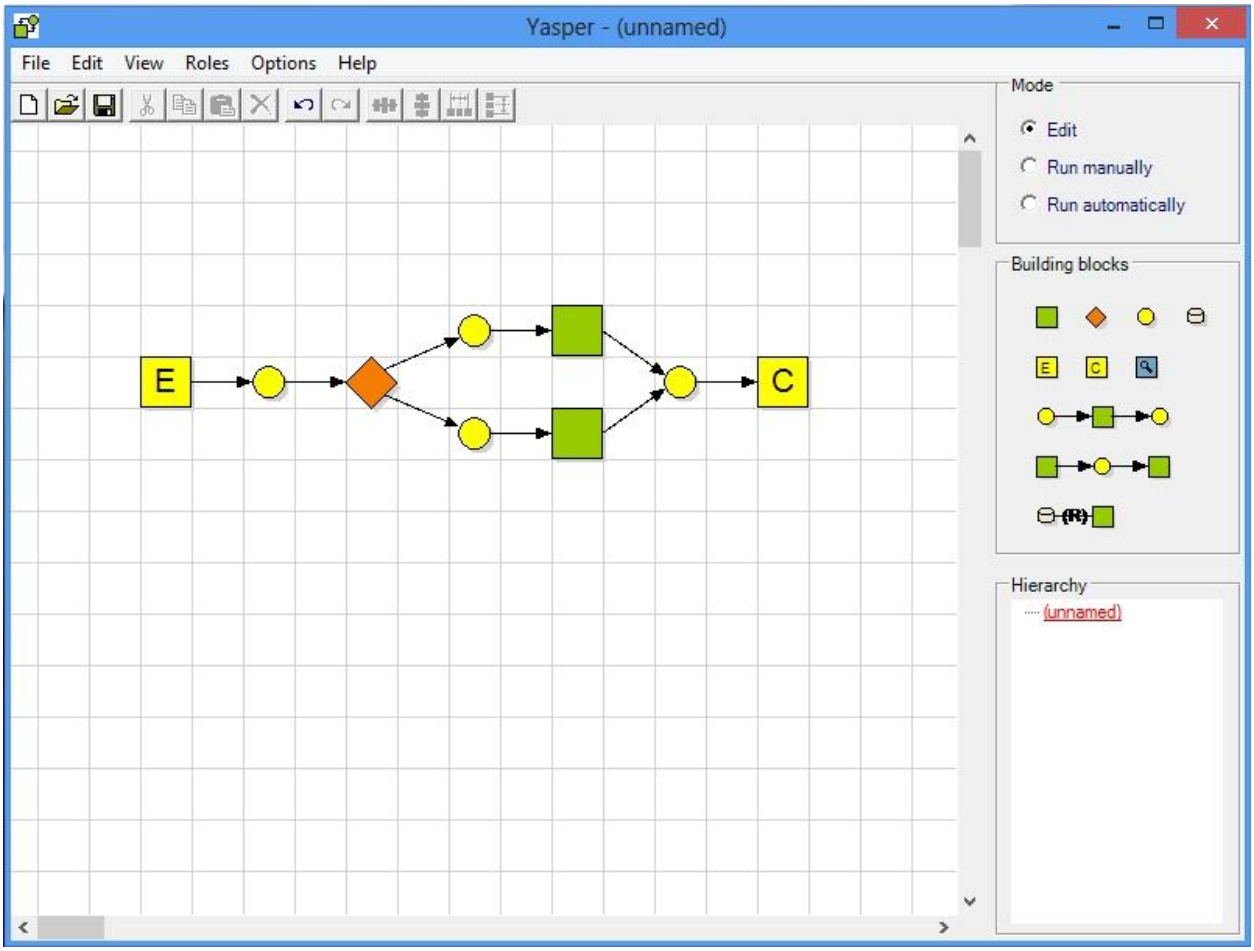


Petrinet afbeelding 4

Net als met een chemisch proces kan een Transition meer dan één Token als input vereisen of meerdere tokens als output hebben.

Een voorbeeld van een visuele Petrinet editor is YASPER, gemaakt door de Technische Universiteit Eindhoven, deze is hieronder afgebeeld.
Naast de standaard iconen van een Petrinet bevat deze editor één extra uitbreiding, de XOR Transition. Deze wordt afgebeeld met een oranje ruit.

In het voorbeeld Petrinet hieronder zal een Token vanuit de Emitter verzonden worden naar de eerste Place. Hier vanuit gaat de Token door de XOR Transition waarna het op een van de twee Places eindigt. Hoe deze keuze wordt bepaald wordt in het volgende hoofdstuk uitgelegd. Nadat de Token door de Transition is gegaan komt deze uit op de laatste Place en wordt deze opgenomen door de Collector.



YASPER een voorbeeld van een visuele Petrinet Editor

2.3 FlinQ 4.0 Petrinet

Boven op de standaard werking heeft een FlinQ 4.0 Petrinet nog wat extra functionaliteit. Een Token heeft bijvoorbeeld informatie in zich over het type event, van welk apparaat het afkomstig is en andere parameters.

Ook heeft elk Petrinet een Criteria element waaraan voldaan moet worden wil het een Token accepteren. In dit Criteria worden de waardes in de Token vergeleken met bepaalde specifieke waardes aan de hand van een XQuery, dit heeft een boolean uitkomst dus 'True' of 'False'.
Dit mechanisme is nodig omdat een Token gegenereerd wordt aan de hand van een event, bijvoorbeeld een alarm dat afgaat en dit Token wordt aan elk Petrinet gepresenteerd tot de juiste deze accepteert (het kan ook voorkomen dat geen enkel Petrinet de Token accepteert).

Een andere belangrijke toevoeging is dat Transitions mogelijk Actions in zich hebben die worden uitgevoerd als de Token daar is aangekomen. Deze Actions zijn net als Criteria in XQuery geschreven maar hebben geen boolean uitkomst.
In het voorbeeld op de volgende pagina zijn er twee acties, één om een commando voor de client te creëren en de tweede actie verstuurd deze.

Naast de standaard Transition, Emitter en Collector is er nog een vierde type Transition; de XOR. Een XOR type Transition heeft geen Actions maar Conditions, met de Tokens als input kan aan de hand van deze Conditions een boolean beslissing gemaakt worden. Een XOR Transition mag maar twee uitgaande Arcs hebben. Eén hiervan moet van het standaard type Arc zijn en de andere van een speciaal type, de Negative Arc. Afhankelijk van de uitkomst van de Conditions wordt de Token verstuurd over de standaard Arc bij een positieve uitkomst en over de Negative Arc bij een negatieve uitkomst.

Ook hebben de Petrinets die gebruikt worden voor FlinQ 4.0 speciale Arcs namelijk de Reset en Inhibit Arcs.
De Reset Arc is een Arc vanaf een Transition naar een Place. Zodra een token door de betreffende Transition gaat worden alle tokens verwijderd bij de Place aan de andere kant van de Arc.
De Inhibit Arc gaat van een Place naar een Transition toe. Zolang er op de Place één of meerdere Tokens bevinden voorkomt deze Arc dat de Transition nieuwe Tokens kan accepteren.

Er is voor gekozen om vrijwel alle mogelijke acties door de server af te handelen. Dit om de client zo licht en simpel mogelijk te houden. Een ander voordeel is dat specifieke formulieren of UI verzoeken die een klant misschien heeft niet verwerkt hoeft te worden in de software maar in de logica gedaan kan worden.

Zowel de Criteria, de XOR Conditions als de Transition Actions zijn geschreven in XQuery, dit is een taal om te communiceren met een XML Database. Initieel is er voor de FlinQ 4.0 server gekozen om de status van apparaten op te slaan. Hiervoor is de keuze gevallen op een Sedna XML database gedeeltelijk omdat de tokens ook in XML format verstuurd worden. Later is besloten dat het opslaan van gegevens optioneel is en een XML database niet langer vereist is. Er is een vervangende optie bedacht voor XQuery/Sedna Actions, hier wordt later meer over uitgelegd.

Hieronder is een afbeelding van hoe een Petrinet op dit moment geformuleerd wordt in XML. Hierin is te zien dat het hoofdelement 'Petrinet' is met als childs Criteria, Transition, Place en Arc elementen. Elk van deze elementen heeft een bepaald type met uitzondering van Places. Ook hebben de Places en Transitions een ID waar naar verwezen wordt bij de Arcs. De gele text in de Criteria, Action en Condition elementen (Condition is niet afgebeeld in dit voorbeeld) is de XQuery.

```
<?xml version="1.0" encoding="utf-8"?>
<PetriNet  xsi:noNamespaceSchemaLocation="network.xsd" xmlns=""
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Criteria type="SednaCriteria">
    <![CDATA[
    declare variable $token := $TOKEN;
    name($token)='test_close_form'
    ]]>
  </Criteria>
  <Transition id="1" type="EMITTER" />
  <Transition id="2" type="DEFAULT">
    <Action type="SednaQueryAction">
     <![CDATA[
       declare variable $token := $TOKEN;

       <ClientCommand type='CloseForm' form_id='{string($token)}'>
         <Client id="0" />
       </ClientCommand>
     ]]>
    </Action>
    <Action type="SendCommandToClientAction" />
  </Transition>
  <Transition id="3" type="COLLECTOR"/>

  <Place id="4" />
  <Place id="5" />

  <Arc fromId="1" toId="4" type="FORWARD" />
  <Arc fromId="4" toId="2" type="FORWARD" />
  <Arc fromId="2" toId="5" type="FORWARD" />
  <Arc fromId="5" toId="3" type="FORWARD" />

</PetriNet>
```

Petrinet in XML

Het bovenstaande Petrinet is voor het afhandelen van het sluiten van een formulier bij de client.
Er wordt bij de Criteria gecontroleerd of het Token van het type 'test_close_form' is. Zodra de Token bij de Transition is aangekomen worden de Actions uitgevoerd. In dit geval houdt dat in dat er een nieuw Token wordt gegenereerd voor de Client en bij de tweede Action wordt deze verstuurd. De Transition die hierna volgt is van het type Collector dat het einde van het Petrinet aanduidt.

2.4 Probleemstelling

Zoals eerder aangegeven in dit document is het probleem dat het creëren van een Petrinet te veel expertise vereist en te fout gevoelig is. Zoals in de voorbeeldafbeelding op de vorige pagina wordt getoond is de notering op dit moment in XML en XQuery.
Het XML deel kan worden geschreven in elke text editor en heeft ook geen functie om te controleren of dit een correct Petrinet is, behalve door dit te proberen in te laden op de server en te controleren of er een foutmelding wordt gegeven.
Het XQuery deel wordt vaak eerst geschreven in de text editor Sublime Text 2 voordat het bij het XML wordt geplakt. Voor Sublime Text 2 is een script geschreven om de code uit te voeren op de XML database. Dus hiervoor is wel een controle mechanisme maar deze is niet geïntegreerd en onhandig.

Een ander probleem dat zich mogelijk in de toekomst kan voor doen is dat er een nieuwe versie van de FlinQ 4.0 server uitkomt met nieuwe syntax voor de Petrinet of nieuwe type Actions.
Het is bijvoorbeeld goed mogelijk dat de huidige XML database wordt vervangen door een interne set objecten die dus een hele andere taal vereist om mee te kunnen communiceren. Wanneer er een nieuw Petrinet wordt geschreven voor een oudere server wordt er niet duidelijk aangegeven welke syntax correct is of welke Actions beschikbaar zijn.

2.5 Doelstellingen

Het doel van dit project is het maken van Petrinets toegankelijker te maken. Door in plaats van handmatig XML text te typen moet er een visuele editor komen. Dit is intuïtiever, voorkomt syntax fouten en toont automatisch de beschikbare functies van de server waarvoor geschreven wordt.

Waar nu nog meerdere tools aan te pas komen moet één editor komen om aan alle behoeftes te voldoen. Bij voorkeur is deze editor in C++ geschreven en platform onafhankelijk, net als de Server en client software.

2.6 Deelopdrachten, functionele eisen en projectgrenzen

Hieronder staan deelopdrachten die opgeleverd moeten worden. Er wordt gepoogd om elk van deze deelopdrachten in een sprint te verwerken en af te ronden.

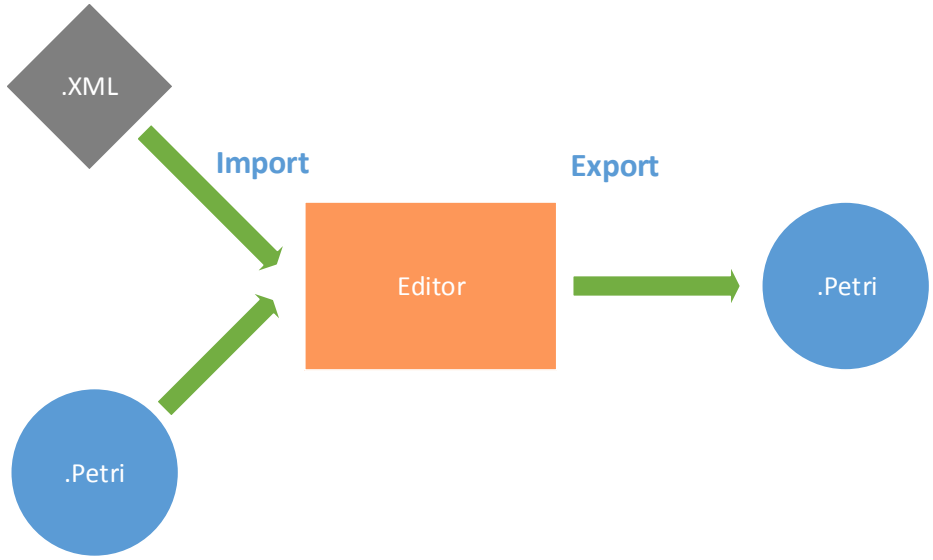
Deelopdrachten/producten:

- Visuele editor op Petrinet (XML) niveau
- Template editor op Action/Conditons/Criteria (XQuery) niveau
- Visuele editor op Action/Conditons/Criteria (XQuery) niveau
- DLL met server specifieke properties die ingeladen kan worden
- Error checker voor Petrinet (XML)
- Error checker voor Action/Conditions/Criteria (XQuery)
- Opslag format voor server properties, functionele en visuele data.
- Nieuwe functie in de server om nieuwe opslag format uit te lezen

Hieronder staan de functionele eisen die verwacht worden van de Petrinet editor.
Hoewel sommige van deze functionele eisen beschouwd kunnen worden als technische eisen zijn er geen puur technische eisen opgegeven. Om die reden wordt hier dus geen aparte lijst met technische eisen getoond.

Functionele eisen:

- Opslaan van functionele en visuele data in XML
- De server moet de output files (.Petri) van de editor in kunnen laden
- De editor moet deze .Petri files kunnen bewerken
- De editor moet ook enkel functionele data (XML) kunnen omzetten naar een .Petri file
- Transitions en Places moeten een naam kunnen krijgen.
- Knippen en plakken moet werken tussen meerdere instanties van de editor (moet dus gebruik maken van Windows clipboard)



In en output files van de editor

Project grenzen:

Hoewel ik wel een aantal Petrinets zal schrijven voor klanten van Flexposure om ervaring op te doen met de XQuery taal en Petrinets zelf zal ik niet vast ingezet worden om deze te schrijven.
Ook zal ik niet ingezet worden om andere delen van de FlinQ 4.0 server aan te passen die niet gerelateerd zijn aan het maken van de editor.

Wel zal het waarschijnlijk voor komen dat ik delen van de server zal moeten wijzigen of toevoegen om ondersteuning van de nieuwe .Petri files mogelijk te maken. En omdat heel FlinQ 4.0 nog in ontwikkeling is zal ik vaak mijn code moeten aanpassen omdat bestaande libraries gewijzigd worden.

2.7 Op te leveren producten

Hieronder staat een lijst met producten die ik aan het einde van mijn stage periode zal opleveren.

Voor het bedrijf:

- De Petrinet editor voor FlinQ 4.0
- Een instructie gids voor de editor
- De software voor de FlinQ 4.0 server om de output van de editor in te lezen

Voor school:

- Plan van aanpak
- Een scriptie over mijn stage opdracht
- Een presentatie over mijn stage opdracht

2.8 Project activiteiten

Hieronder staat een tabel met de activiteiten die ik verwacht te ondernemen en de geschatte periode waarin deze gedaan zullen worden.

Activiteit	**Periode**
Onderzoek naar werking Petrinet en de FlinQ 4.0 specifieke functies	Week 1
Onderzoek naar gebruik van verschillende talen en UI platformen voor de editor	Week 1
Eerste opzet van de visuele editor realiseren (lees: het selecteren, verslepen, knippen, plakken, enz. van Transitions, Places en Arcs.)	Week2 tot 3
Opslagmethode ontwikkelen voor functionele en visuele data	Week 4
Actions/Conditions/Criteria toevoegen en aanpassen in de vorm van een text editor (nog niet visueel)	Week 5
Actions/Conditions/Criteria code verplaatsen naar een DLL en deze inladen met de editor	Week 6 tot 7
Server functionaliteit geven om editor output (een bestand met functionele, visuele data en server properties) in te laden	Week 8
Actions/Conditions/Criteria toevoegen en aanpassen aan de hand van templates (nog niet visueel)	Week 9 tot 10
Error checking op Petrinet niveau	Week 11
Error checking op XQuery niveau	Week 12~13
Mogelijkheid om XQuery te testen op XML database via de editor	Week 14
Onderzoek naar visuele taal voor XQuery editor	Week 15
Visuele editor maken voor Actions/Conditions/Criteria.	Week 16 tot 17
XQuery editor koppelen aan XML database om dynamisch de beschikbare devices en eigenschappen te tonen.	Week 18 tot 19
Uitloop	Week 20

2.9 Technieken en middelen

Hieronder staat een lijst van talen, technieken, tools en boeken die ik gebruik of denk te gaan gebruiken om mijn stageopdracht te voltooien.

-C++

De programmeertaal waarin de huidige software en de software die nog gemaakt moet worden in geschreven is.

-XML

De taal waarin de Petrinets op dit moment worden geschreven.

-XQuery

De taal die gebruikt wordt om te communiceren met de XML database.

-Microsoft Visual Studio Developer

De IDE die gebruikt wordt in het bedrijf voor het schrijven van C++ en C#

-**Sublime Text 2**

De IDE die gebruikt wordt voor het testen van query's op de XML database.

-WxWidgets

De opensource UI library die gebruikt wordt voor het ontwikkelen van de editor.

-GoogleTest

Een toolkit voor het geautomatiseerd testen van C++ code.

-Head First: Design Patterns

Een boek over Design Patterns welke ik op advies van mijn stagebegeleider aan het lezen ben. Hierin staan veel nuttige technieken voor het ontwerpen van software die ik kan toepassen.

2.10 Risico's

Omdat ik nog niet bekend ben met de XQuery is het een risico dat ik deze taal niet snel genoeg onder de knie zal krijgen voor het correct en op tijd voltooien van de editor.
Ook is mij in de eerste paar weken van mijn stage opgevallen dat mijn kennis van C++ niet op het niveau is dat ik het graag zou hebben. Hoewel ik niet verwacht dat deze risico's negatieve gevolgen zullen hebben zijn het wel risico's die aangekaart moeten worden.

3. Gekozen Aanpak

In dit hoofdstuk worden activiteiten aan het begin van het project behandeld, worden de beslissingen die genomen zijn na het maken van het eerste prototype besproken en de beslissingen die daarna genomen zijn.

3.1 Start van het project

In de eerste weken van de afstudeerstage zijn de meeste activiteiten gedaan om meer kennis te vergaren over de producten die er nu zijn en hoe deze met elkaar samenwerken. Zo heb ik een uitleg gekregen over de interne werking van de Server, heb ik meegekeken hoe een Connector werd geschreven voor een nieuw apparaat, er waren een aantal Petrinet oefenopdrachten voor mij bedacht en heb ik een korte cursus over XQuery gekregen van mijn stagebegeleider.

Nadat er een duidelijk beeld was gecreëerd van de producten die er al waren heb ik me verdiept in de verschillende mogelijkheden voor het product dat nog gemaakt moet worden; de editor.
Hiervoor is onderzoek gedaan naar verschillende programmeertalen, zijn diverse UI toolkits uitgeprobeerd en is er gekeken naar diverse, al bestaande, Petrinet editors.

Een UI toolkit is een library waarmee het creëren van een GUI kan worden gerealiseerd.

3.2 Prototype

Het doel van het prototype was om een simpele opzet te maken van de uiteindelijke applicatie maar die nog niet meteen op een correcte of elegante manier geschreven hoeft te worden. Op deze manier kan er snel een werkende applicatie gebouwd worden en zal snel ontdekt worden of er problemen zijn waar met het uiteindelijke ontwerp rekening mee gehouden moet worden.

Voor het maken van het prototype ben ik meteen begonnen met C++. Hoewel bestaande libraries voor FlinQ 4.0 in dezelfde taal zijn geschreven heb ik daar expres nog geen gebruik van gemaakt. Dit zou namelijk mogelijk meer tijd kosten om goed te begrijpen en toe te passen dan zelf een simpel equivalent te schrijven.
Ook heb ik op advies van mijn stagebegeleider gebruik gemaakt van de UI toolkit QT. Dit is een zeer populaire kit voor C++ onder Windows, Mac, Linux en Symbian (Nokia). Hoewel het werken met deze populaire toolkit als erg plezierig werd bevonden zat er wel een groot nadeel aan. De QT source kan namelijk niet los geïnstalleerd of gecompileerd worden.

3.3 Beslissingen

In dit hoofdstuk worden alle beslissingen die gemaakt zijn na het creëren van het prototype. Sommige van deze beslissingen zijn gebaseerd op problemen die zijn ondervonden tijdens het ontwikkelen van dit prototype en andere beslissingen zijn gemaakt na algemeen heroverwegingen in dit project.

3.3.1 Taal : C++

Aan het begin van het project mocht ik zelf beslissen in welke taal de editor geschreven ging worden. Hier voor moest ik een aantal overwegingen maken, namelijk;

- Met welke taal ben ik het meest bedreven
- Met welke taal kost het maken van een Graphic User Interface (GUI) de minste moeite
- Als ik niet voor C++ kies, hoeveel bestaande code kan ik niet hergebruiken en moet ik opnieuw maken

Hoewel ik bij de eerste twee punten sterk naar Java neigde is de keuze toch op C++ gevallen. Wat voor deze keuze ook mee heeft gespeeld is dat de kennis die wordt opgedaan in C++ ook toegepast kan worden voor het aanpassen van de andere software componenten.

3.3.2 UI Toolkit : WxWidgets

Om een applicatie te maken met een GUI zijn er vele verschillende toolkits beschikbaar voor C++ zoals QT die voor het prototype is gebruikt. Zoals eerder genoemd heeft QT één groot nadeel namelijk dat de source code hiervan niet los te verkrijgen is. Dit betekend dat iedereen die aan de FlinQ4.0 editor zou willen werken handmatig QT zou moeten installeren. Dit is niet acceptabel.

Om die reden is er verder onderzoek gedaan en is er gekozen voor WxWidgets. Deze UI Toolkit ondersteund eveneens C++ voor Windows, Mac en Linux maar is volledig open-source.

3.3.3 (Nog) geen visuele editor voor Actions/Conditions/Criteria

Hoewel tijdens de initiële planning is bedacht dat er wel een visuele editor zou komen voor de Actions, Conditions en Criteria is hier later van afgezien. Dit besluit heeft meerdere redenen.

Een van deze redenen was dat het onderzoeken van een geschikte visuele taal voor de representatie van XQuery veel onderzoek zou vereisen en zou een te groot deel van de stage periode op eisen. Dit probleem wordt nog groter door de ondersteuning van andere talen naast XQuery voor Actions.

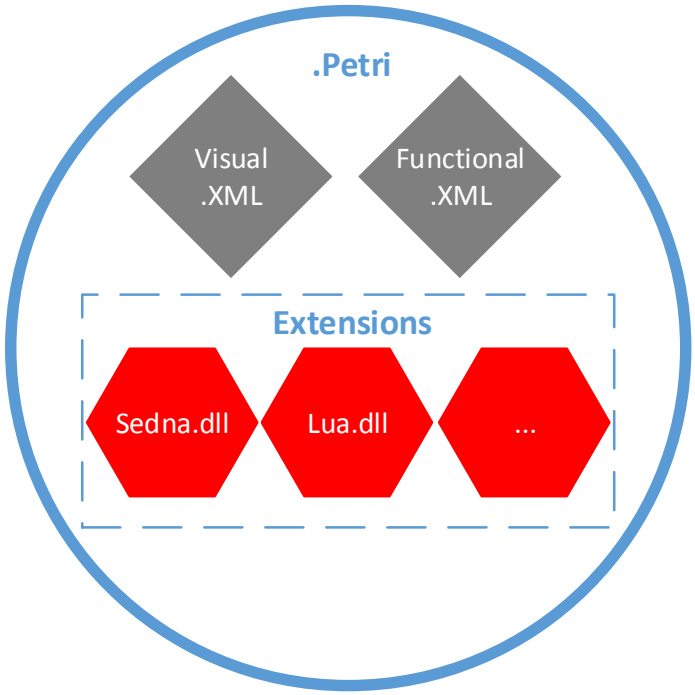
Een ander argument om dit later pas toe te voegen is dat er nu nog geen behoefte aan is. Voorlopig zal de editor enkel intern bij Flexposure gebruikt worden en zijn de werknemers technisch van hoog genoeg niveau dat het schrijven van XQuery waarschijnlijk sneller zal gaan dan het omgaan met een visuele taal.

3.3.4 Output : .Petri files

Om de beslissing van de inhoud van een .Petri file beter te kunnen onderbouwen is wat achtergrond informatie nodig. Zoals eerder gemeld in deze scriptie wordt XQuery in combinatie met een Sedna XML database gebruikt voor het uitvoeren van de Actions en het testen van de Conditions en Criteria. Ook is eerder vermeld dat deze XML database, en dus ook de op XQuery gebaseerde Actions, niet langer een vast onderdeel van de FlinQ 4.0 server zijn. Een andere reden waarom er naar een alternatief is gezocht is vanwege de relatief lage performance van Sedna Query's.
Ter vervanging zijn Lua Actions/Conditions/Criteria gecreëerd. Lua is een veel snellere, lichtgewicht, embedded scripting taal die geen database vereist. Na performance tests blijkt Lua Actions ook tot wel 60 keer sneller te zijn dan Sedna Actions.

In de toekomst is het mogelijk dat er nog meer verschillende type Actions ondersteund moeten worden. Zo kan het de wens zijn van de klant om gegevens in een bestaande SQL database op te slaan in plaats van een XML database. Om zowel de server als de Petrinet editor hier flexibel mee om te laten gaan worden de implementaties van deze specifieke Actions opgeslagen in zo genoemde Extensions. Deze Extensions zijn DLL's die worden opgeslagen in de .Petri file. Extensions worden alleen toegevoegd aan de file als er ook gebruik gemaakt wordt van de corresponderende Actions.

Naast deze Extension DLL's zijn er ook nog twee XML bestanden. Het Functional.XML bestand bevat de informatie van het Petrinet die van belang is voor de server en is hetzelfde als de XML die met de hand werd geschreven voordat met deze Petrinet editor werkt gewerkt.
Het Visual.XML bestand bevat de visuele data die enkel belangrijk is voor de presentatie op de editor. Hierin in zijn de coördinaten en eventuele namen van Places, Transitions, Emitters en Collectors bewaard. Deze worden gekoppeld aan de functionele data aan de hand van een overeenkomend ID.



Overzicht met de inhoudt van een .Petri file

3.4 Testmethoden

De tests die gebruikt worden om de kwaliteit van de editor te kunnen meten kunnen in twee categorieën gesplitst worden.

Het eerste deel zal puur feedback zijn afkomstig van de toekomstige gebruikers van de editor, dit wordt de usability test genoemd. Omdat de editor op het moment van schrijven nog niet toegepast wordt kan hier nog niets over gerapporteerd worden.

Het tweede deel richt zich meer op het correct functioneren van de applicatie. Aan de hand van geautomatiseerde tests zullen bepaalde functies getest worden. Het voordeel van deze testmethode is dat de tests op de achtergrond gedraaid worden terwijl de code aangepast wordt. Dus als er per ongeluk een bug wordt geïntroduceerd tijdens het refactoren van de code wordt je hier meteen op geattendeerd.

Voor de Petrinet editor zijn unit tests gecreëerd voor deze functies:

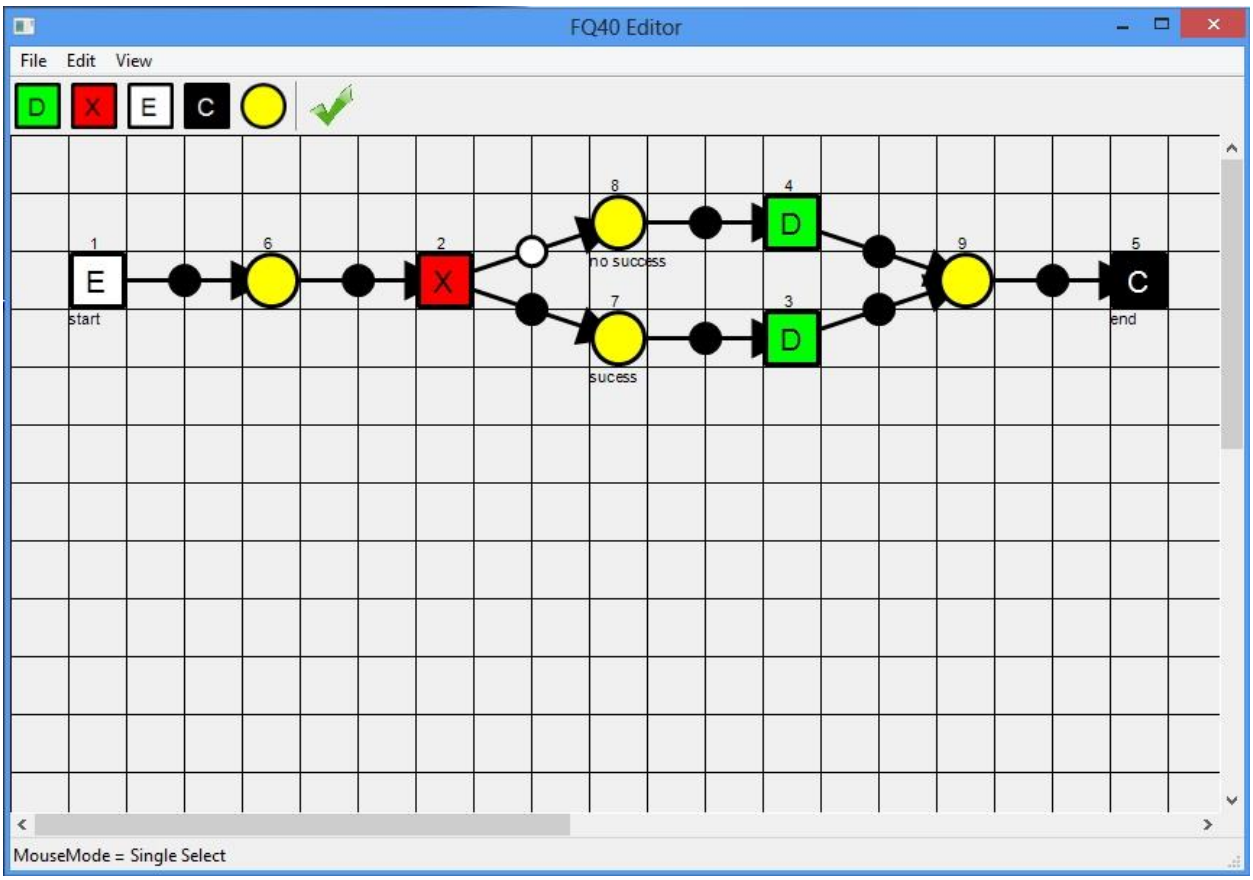
- Serialiseren
- Deserialiseren
- Plakken
- Opslaan file
- Laden vanuit file
- Inladen van DLL

4. De Petrinet Editor

Aan de hand van dit hoofdstuk wordt het eindresultaat besproken. Eerst zal het uiterlijk, de GUI, van de editor besproken worden. Hierna wordt dieper ingegaan op de code, interne werking en functionaliteit.

4.1 Uiterlijk

In de afbeelding hieronder is de uiteindelijke versie van de FlinQ 4.0 Petrinet editor getoond.



De FlinQ 4.0 Petrinet editor

Aan de hand van de afbeeldingen in de volgende paragraaf worden de verschillende iconen uitgelegd en wordt de relatie met de eerder besproken functionaliteit van een FlinQ 4.0 Petrinet gemaakt.

4.1.1 Petrinet objecten

Net als bij YASPER, de Petrinet editor die eerder als voorbeeld is gebruikt, is de Place een gele cirkel. Ook is hier gekozen om de Transition als groen vierkant af te beelden. Voor de duidelijkheid wordt hier wel over een Default Transition gesproken en wordt de letter ‘D’ afgebeeld binnen het vierkant.

- Place 
- Default Transition 

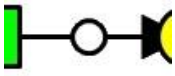
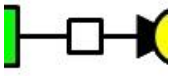
Voor de XOR Transition, Emitter en Collector is wel afgeweken van het kleuren model van YASPER. Dit omdat er besloten is dat elk icoon op kleur te onderscheiden moet zijn en we ons hebben beperkt tot de primaire kleuren. Ook hier wordt de eerste letter van de naam van het object in het icoon getoond.

- XOR Transition 

- Emitter 

- Collector 

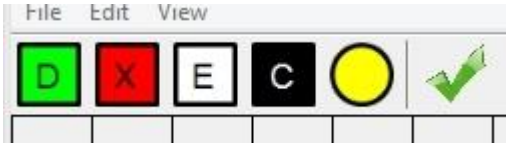
Omdat er in tegenstelling tot de YAPSER editor meerdere type Arcs zijn moeten deze ook visueel te onderscheiden zijn. Hiervoor is bewust niet gekozen om elk van de Arcs een unieke kleur te geven die verschilt van de kleuren van de Places en Transitions. Dit zou resulteren in een overdaad aan verschillende kleuren op het scherm waardoor het gemakkelijk herkennen van iconen verloren raakt. Ook letters in of naast de Arc maken deze niet in een oog opslag te herkennen.
Daarom is hier dus gekozen om het verschil aan te duiden met lege en gevulde cirkels en vierkanten in het midden van de Arc.

- Default Arc 
- Negative Arc 
- Block Arc 
- Reset Arc 

4.1.2 UI elementen

Toolbar

In de afbeelding die hieronder wordt getoond is de Toolbar van de editor te zien. Hier worden de meest gebruikte functies getoond. Vanaf links gezien zijn de eerste 5 Toolbar knoppen om nieuwe Petrinet objecten te creëren. Als hier op gedrukt wordt word het overeenkomende object geplaatst op het raster waar nogmaals met de muis geklikt wordt.
Na de scheidingslijn is de knop om de Validator uit te voeren. Deze controleert het Petrinet op fouten.



De toolbar van de editor

Menu opties

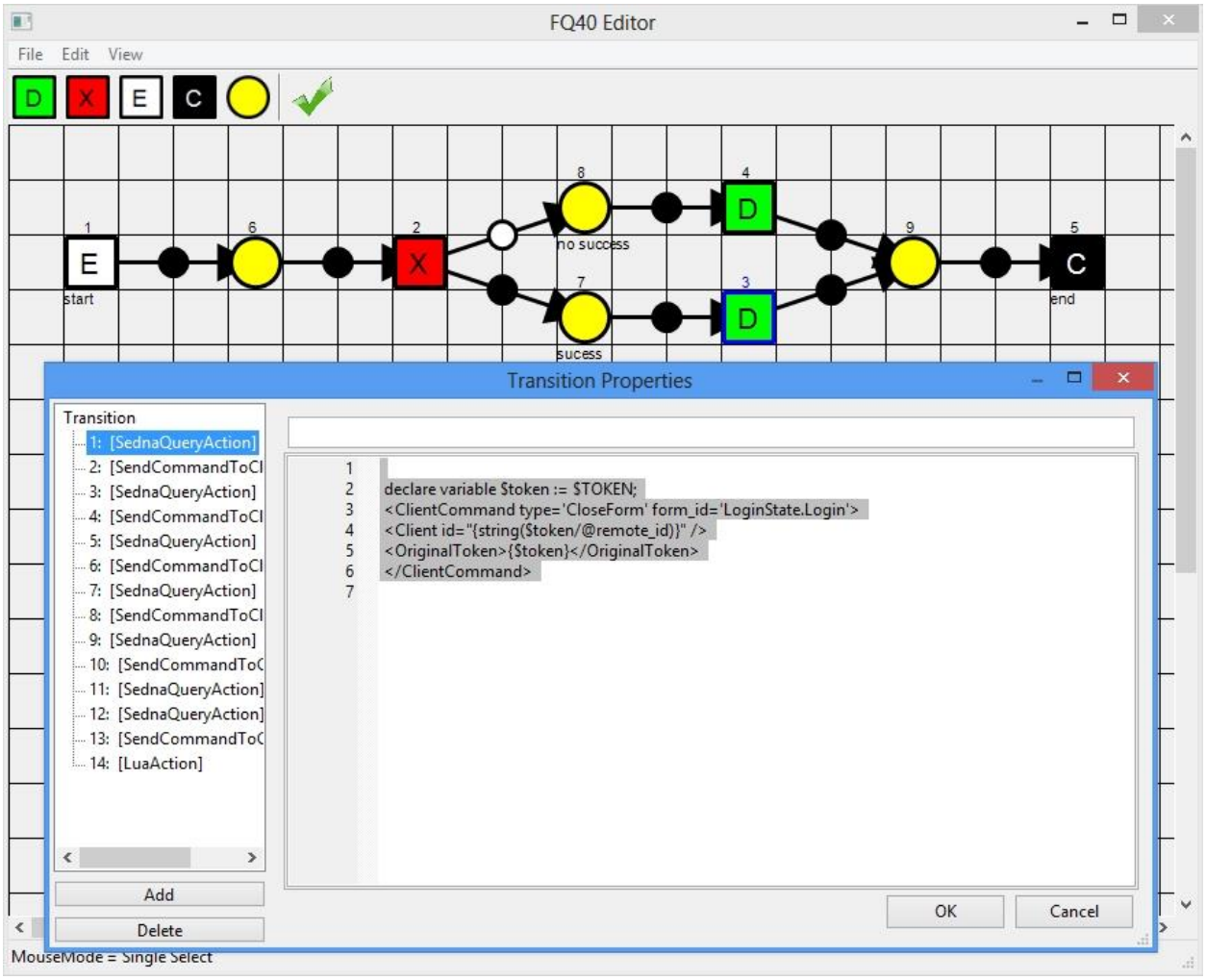
De menu opties spreken redelijk voor zich.
Onder File zijn de opties om een .Petri file of een XML file te openen. Ook schuilt hier de optie om het Petrinet op te slaan.
Bij het Edit menu zit slechts één functie en dat is die voor het tonen van het scherm om de Criteria's aan te passen.
Bij het View menu kunnen de volgende functies aan of uit gevinkt worden:

-Snap to grid -> zorgt dat de Petrinet objecten altijd precies in het raster passen
-Show grid -> toon het raster
-Show name -> toon de naam van de Petrinet objecten onderaan de objecten
-Show ID -> toon het ID van de Petrinet objecten bovenaan de objecten

Editor menu opties

Vensters

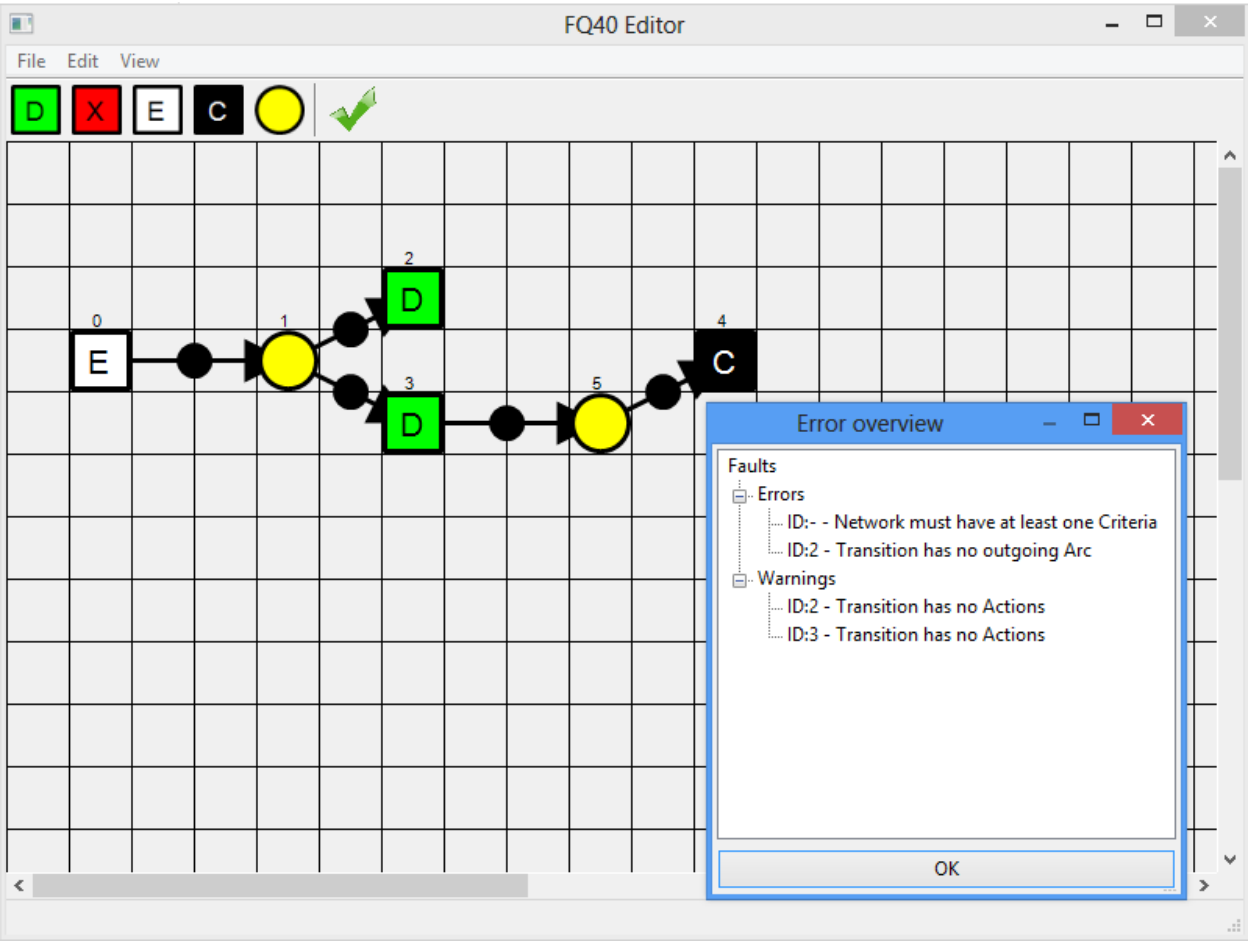
Hieronder is eigenschappen venster getoond van een Default Transition. Deze wordt geopend na het dubbel klikken op de Transition. Hierin wordt een lijst getoond met de Actions die aanwezig zijn. Vanuit hier kunnen Actions toegevoegd of verwijderd worden. Ook hebben sommige Actions de functie of text op te geven, namelijk XQuery of Lua commando's. Indien de geselecteerde Action dit ondersteund wordt er een text veld getoond met de huidige waardes zodat deze aangepast kan worden zoals in de afbeelding te zien is.
Naast het aanpassen van de Actions kan de Transition ook een naam gegeven worden.



Transitions Propeties view waarin Actions toegevoegd, verwijderd of aangepast kunnen worden

Voor het bewerken van de Conditions van een XOR Transition of de Criteria wordt een vergelijkbaar venster gebruikt als bij een Default Transition.
Het eigenschappen venster van een Emitter, Collector of een Place toont enkel de optie om een naam op te geven.

De editor heeft ook een validator functie die het Petrinet controleert of bepaalde componenten wel aanwezig en of bepaalde verbindingen wel correct zijn. Als dit niet het geval is wordt het Error overview getoond zoals in de afbeelding hieronder.



Error overview waarin de fouten en waarschuwingen worden getoond

4.2 Design Patterns en Principles

Voor het ontwerpen van de software zijn een aantal code design principles en patterns toegepast. Hier onder worden deze kort toegelicht.

4.2.1 Model View Controller

Het Model View Controller principle omschrijft het concept van scheiden van data, uiterlijk en verwerking. Dit houdt in dat er aparte objecten zijn voor elk van deze taken en dat deze niet in elkaar overlopen.

Een voorbeeld van deze toepassing is het detailscherm van een Transition. De data wordt bewaard in het VisualTransition object, het uiterlijk van detailscherm wordt aangemaakt in het TransitionPropertiesDialog en het aanroepen van het scherm of aanpassen van de data gebeurt in het controller object. De code van het controller object die de view aanmaakt en de data aanpast staat hieronder.

```
//open the dialog
TransitionDialog dialog((m_owner->GetPosition()+wxPoint(150, 150)),
&m_Network.vTransitions[i], m_Core);

//if OK is pressed the changes are applied to the data object
if(dialog.ShowModal()==wxID_OK)
{
	// Copy the actions back.
	m_Network.vTransitions[i].actions = dialog.actions;
	m_Network.vTransitions[i].SetName(dialog.name);
	Refresh();
}
```

Code snippet van het openen van het de propertiesDialog en aanpassen van de data in de controller

4.2.2 Single Responsibility Principle

Dit coding principle eist dat de programmeur zo veel mogelijk verantwoordelijkheden scheidt in aparte bestanden. Het nut hiervan is dat het gemakkelijker is om code te hergebruiken in verschillende projecten. Het opdelen van verantwoordelijkheden maakt het ook veiliger om één van de objecten aan te passen zonder dat het andere object ongewenst zich ook anders gedraagt.

Een voorbeeld van hoe dit is toegepast bij het ontwikkelen van de Petrinet editor is het scheiden van de functionele en visuele data van een Petrinet. Deze worden niet alleen als twee aparte XML bestanden opgeslagen, ook de verantwoordelijkheid van het serialiseren en deserialiseren is opgesplitst in twee objecten. Omdat dit is gedaan kan het deserialisatie object gemakkelijker gebruikt worden bij het inladen van enkel een functioneel XML bestand en kan deze eventueel ook gebruikt worden voor het deserialiseren bij de FlinQ 4.0 server.

4.2.3 Factory Pattern

Een Factory Pattern wordt toegepast wanneer een object verantwoordelijk is voor het creëren van andere objecten. Eén van de mogelijke redenaties om deze code design pattern toe te passen is vanwege het hiervoor besproken Single Responsibility Principle. Het creëren van deze objecten vereist namelijk bepaalde toegang of andere 'expertise' die de klasse die het object aanvraagt niet heeft.

Voor de toepassing bij de editor is hiervoor gekozen omdat de klasse die het object aanvraagt initieel niet weet welke objecten er beschikbaar zijn om te creëren. Dit gebeurt bij het aanmaken van een Sedna- of Lua- Action/Condition/Criteria. Hier bepaald de DLL die ingeladen bijvoorbeeld welke Actions er beschikbaar zijn. Hierdoor kan dit dus ook niet van te voren bekend zijn in de editor zelf.
Hieronder is de portie van de code van de Sedna.dll. Een verschil tussen dit voorbeeld en een traditioneel Factory Pattern implementatie is dat hier niet het object zelf opgevraagd kan worden maar het serialisatie object. Dit object is dan wel weer in staat om een Action object te creëren.

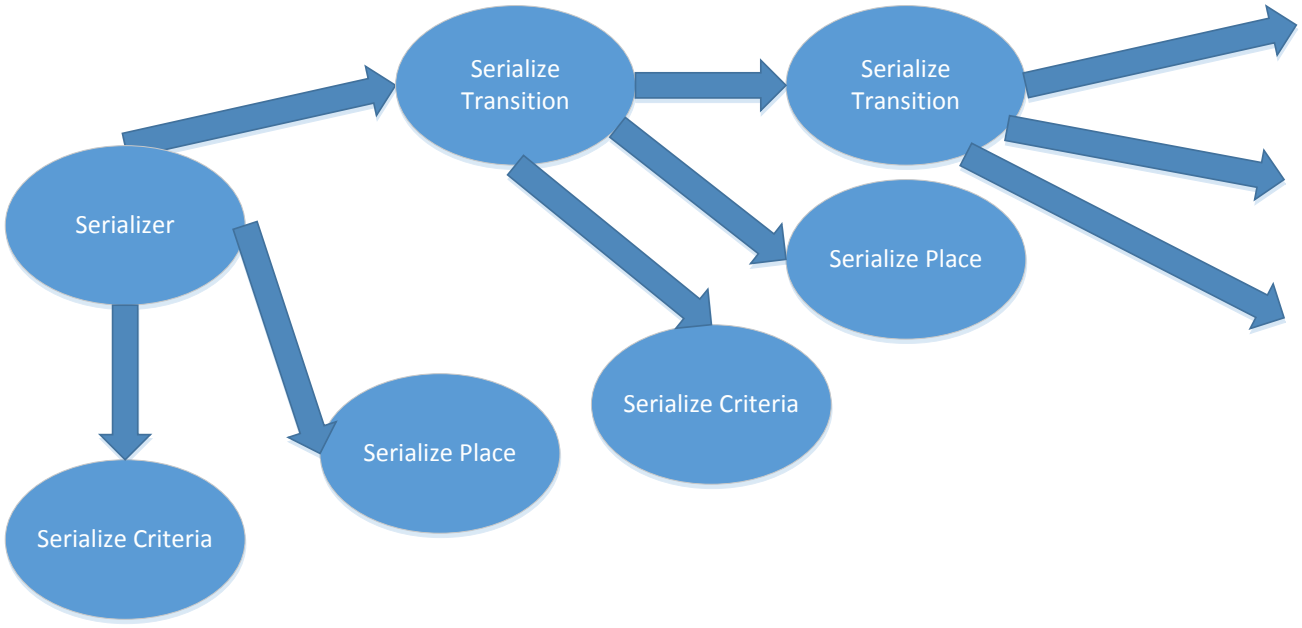
```
// Get a list of all available Action names
virtual std::vector<std::string> GetActions()
{
	std::vector<std::string> actions;
	actions.push_back(TERM_VALUE_SEDNA_QUERY_ACTION);
	actions.push_back(TERM_VALUE_SEDNA_ECHO_ACTION);
	actions.push_back(TERM_VALUE_SEDNA_EVENT_CMD_ACTION);
	actions.push_back(TERM_VALUE_SEDNA_EVENT_INDEX_ACTION);
	actions.push_back(TERM_VALUE_SEDNA_RELOAD_SITE_DATA_ACTION);
	actions.push_back(TERM_VALUE_SEDNA_SEND_CMD_TO_CLIENT_ACTION);
	actions.push_back(TERM_VALUE_SEDNA_SEND_CMD_TO_CONNECTOR_ACTION);
	actions.push_back(TERM_VALUE_SEDNA_STRING_TO_XML_ACTION);
	return actions;
}

// Get the apropriate serializer for the given Action name
virtual Petri::IActionSerializer* CreateActionSerializer( std::string name )
{
	Petri::IActionSerializer* serializer = nullptr;
	if(name == TERM_VALUE_SEDNA_QUERY_ACTION)
	{
		serializer = CreateSednaAction();
		serializer->SetCore(m_Core);
	}
	else if(name == TERM_VALUE_SEDNA_ECHO_ACTION)
	{
		serializer = CreateSednaEchoAction();
		serializer->SetCore(m_Core);
	}
// [....]
	else if(name == TERM_VALUE_SEDNA_STRING_TO_XML_ACTION)
	{
		serializer = CreateSednaStringToXmlAction();
		serializer->SetCore(m_Core);
	}
	return serializer;
}
```

Code snippet voor het ophalen van beschikbare Actions en de objecten om deze te (de)serialiseren

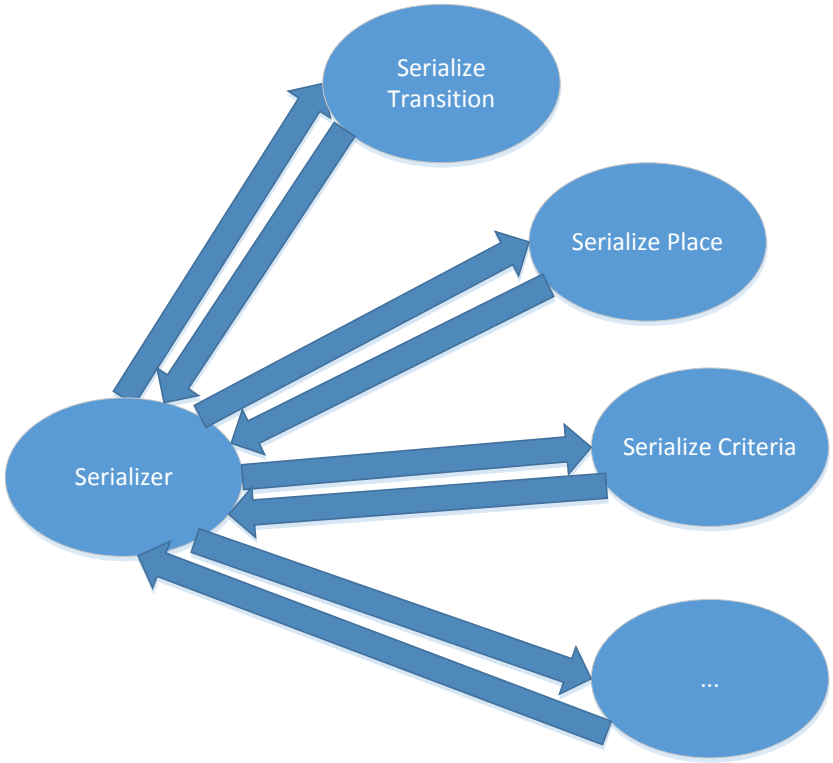
4.2.4 Visitor Pattern

Het Visitor Pattern is een van de minder vaak voor komende Design Patterns maar is zeer geschikt voor het (de)serialiseren van Petrinets.
Gezien de Petrinet data in XML wordt opgeslagen en hierin geen vaste volgorde van elementen is vastgesteld is maakt dit het (de)serialisatie proces niet-lineair. Daar boven op kan elk XML element weer child elementen hebben. Deze situatie is hieronder afgebeeld.



Elke XML element heeft de mogelijkheid om elk ander type element als child te hebben. Serialisatie is op deze methode onmogelijk.

Door het Visitor Pattern toe te passen hoef je hier geen rekening te houden. Dit is in principe een recursief algoritme dat voor elk XML child element weer terug keert naar de basis om daar opnieuw een pad te kiezen. Zie de afbeelding hieronder.



Voor elk XML child element wordt terug gekeerd naar de basis om opieuw een (de)serialisatie functie in te gaan

4.3 Flexibiliteit

De kracht van de FlinQ 4.0 Petrinet editor is de flexibiliteit. Zoals eerder behandeld is het goed mogelijk er nieuwe type Actions/Conditions/Criteria beschikbaar komen voor de FlinQ 4.0 server. Omdat we deze functionaliteit naar een DLL hebben verschoven kunnen deze ingeladen worden en hoeft de editor dus niet aangepast te worden.
Dit brengt wel een probleem met zich mee. Omdat de Editor niet weet welke Actions er bestaan weet deze dus ook niet welke wel of geen een aanpasbare text heeft. Om dit probleem op te lossen is een uitzondering gemaakt op de hiervoor besproken Design Patterns. Een Action is namelijk zelf verantwoordelijk voor het aanmaken van het text veld. Dit is in strijdt met het Single Responsibility Principle en het Model View Controller design pattern.

4.4 Functionaliteit

In dit hoofdstuk wordt de functionaliteit behandeld die niet eerder in dit verslag aan bod is gekomen, namelijk knippen en plakken en kopiëren.

Een van de eisen waaraan de editor moet voldoen is dat Petrinet objecten geselecteerd kunnen worden waarna deze objecten gekopieerd worden. Dit gebeurt aan de hand van de bekende knop combinatie CTRL+C. Uiteraard kan er ook geknipt worden met CTRL+X en geplakt met CTRL+V.
Bij het plakken moet ook de relatieve coördinaten van die de gekopieerde objecten bewaard blijven. Om dit te realiseren word zowel de functionele als de visuele data van de objecten geserialiseerd naar XML text.
Deze text wordt daarna verplaats naar het Clipboard of Klemboard van Windows. Dit maakt het ook mogelijk om de XML data te plakken in een text bestand of in een andere instantie van de editor.
Als de XML data geplakt wordt in de editor zal de data gedeserialiseerd worden naar Petrinet objecten. Hier moet wel opgemerkt worden dat het ID van de objecten wel wordt aangepast zodat het niet voorkomt dat er meerdere objecten hetzelfde ID hebben. Een belangrijk punt is dat de relatieve afstand van de objecten die gekopieerd bewaard blijft maar dat gehele selectie verplaatst wordt naar de locatie van de muis indien deze zich boven het raster bevindt.

5. Conclusies en aanbevelingen

Terug kijkend op het project kan ik best trots zijn op wat er is gemaakt. De R&D afdeling is tevreden met het eindresultaat en het voldoet aan alle eisen die op dit moment gesteld zijn. Op sommige punten in de ontwikkeling van deze applicatie leek het af en toe wel alsof er voor een onnodig complexe is gekozen maar ik denk dat flexibiliteit zeer gewaardeerd gaat worden als er meer uitbreidingen komen voor FlinQ 4.0 en de editor niet aangepast hoeft te worden.

Het is nu afwachten wat de service afdeling er van vindt zodra zij daar mee aan de slag gaan. Natuurlijk zal hier een langere tijd overheen gaan omdat niet alleen de editor nieuw is maar ook het hele concept van Petrinet.
Delen van deze scriptie zullen gebruikt worden als handleiding hiervoor.

De editor zal echter voorlopig niet verder ontwikkeld worden, hooguit zullen er kleine wijzigingen en bug fixes gedaan worden. Uiteindelijk is het wel de bedoeling dat de editor uitgebreid wordt zodat deze ook door derde partijen gebruikt kan worden. Maar daar is nog geen concreet plan of tijdstip voor aangeduid.

Wanneer er wel tijd wordt vrijgemaakt om de editor verder te ontwikkelen heb ik de volgende aanbevelingen.
Gezien C++ en WxWidgets van zichzelf platform onafhankelijk zijn is het wellicht slechts een kleine moeite om de editor ook te testen voor OS X en Linux. Dit lijkt een logische stap omdat de FlinQ 4.0 server en client ook platform onafhankelijk zijn.
Een ander idee waar we tijdens de ontwikkeling aan hebben gedacht maar niet hebben geïmplementeerd is templates voor Actions, Conditions of Criteria. Wat hiermee wordt bedoeld is dat in plaats van een zelf in te vullen text veld een template getoond wordt waar in slecht enkele attributen gewijzigd kunnen worden.

6. Evaluatie

Hieronder worden mijn persoonlijke ervaringen tijdens de stage besproken. Ook heb zijn er aanbevelingen gegeven aan het stagebedrijf en de school aan de hand van deze ervaringen.

6.1 Persoonlijke evaluatie

Terugkijkend op mijn stage kan ik zeggen dat ik een zeer leerzame en fijne ervaring heb gehad. In tegenstelling tot mijn vorige stage heb ik dit keer bewust gekozen voor een veel kleiner bedrijf en ik ben blij dat ik hiervoor heb gekozen. Omdat de R&D afdeling slechts bestaat uit 4 medewerkers heb ik relatief snel mijn plaats gevonden. Naast dat de collega's erg fijn in de omgang zijn werd ik veel sterker betrokken bij beslissingen en voelde ik me een waardig onderdeel van het team.

De initiële training en kennismaking met de huidige software en producten was erg uitgebreid en geduldig aan mij voor gedragen. En ook tijdens de loop van het project kon ik naast mijn stage begeleider ook bij andere collega's terecht met vragen.
Er moet wel gezegd worden dat in enkele gevallen er te veel of ongewenst hulp werd aangeboden. Wanneer bijvoorbeeld een simpele ja/nee vraag wordt gesteld is er al snel de behoefte om de achtergrond hiervan uit te leggen. Dit kan soms tijdrovend zijn maar toegegeven zie ik deze eigenschap ook bij mijzelf terug.

Juist omdat het een klein team is en mijn stagebegeleider vrijwel altijd aanwezig wel om advies aan te vragen werd er ook erg flexibel plannen gemaakt. Hoewel ik me wel kon vinden in deze werkstijl had ik wel een gemis. Ik was namelijk vanuit school uit gewend dat van mij verwacht wordt om UML diagrammen te maken aan het begin van een project. Omdat mijn stagebegeleider hier verlangen naar had heb ik dit ook niet gedaan. Achteraf had ik dit liever wel gedaan omdat ik duidelijker was wat mijn visie was op het ontwerp en hoe deze verschilde van de visie van mijn begeleider.

Een vaardigheid die ik heb geleerd en zeker nog veel aan zal hebben is het correct toepassen van gedeelde libraries in grote projecten om code duplication te voorkomen.

Uiteindelijk kan ik zeggen dat de ervaringen die ik op heb gedaan een goede aanvulling zijn op mij opleiding en dat het werk dat ik gedaan heb heel goed aansluit op waar mijn interesse en ambities liggen.
Daarom ben ik ook heel blij dat Flexposure mij een baan heeft aangeboden en ik me hier verder mag ontwikkelen. Ik kijk uit naar de komende tijd als vast onderdeel van het team.

6.2 Aanbevelingen voor Flexposure

Hoewel ik zeer zeker veel heb geleerd tijdens mijn stage van mijn collega's denk ik ook dat er verbeterpunten zijn en er waardevolle technieken zijn die ik mijn collega's kan leren.
Zo is het erg duidelijk te merken dat de R&D afdeling gewend is om in een zeer klein team te werken en om die reden worden vaker informele afspraken gemaakt dan formele.

Ik heb aangegeven aan Flexposure dat het team een persoon nodig heeft die sterk is in UML diagrammen en het creëren van uitgebreide planningen. Hier zijn zij akkoord mee want zij vinden dit zelf ook een groot gemis.

Ik zou graag mijn kennis delen en wat meer structuur en documentatie brengen naar Flexposure. Ik ben van mening dat dit de communicatie tussen zowel de verschillende afdelingen als het uitbesteden van werk ten goede komt.

6.3 Aanbevelingen voor Hogeschool Utrecht

Hoewel ik heb gemerkt dat mijn kennis van C++ niet op het niveau was dat ik het graag zou willen heb ik hier geen commentaar over richting de opleiding. Dit is omdat ik besef dat het dieper ingaan op één taal ten koste zal gaan van de tijd die beschikbaar is om de kennis te verbreden en andere talen te leren.

Wat ik wel graag zou zien is dat er meer aandacht wordt besteed aan code design patterns en algoritmes. Deze technieken zijn op meerdere talen toepasbaar en hebben sterke meerwaarde bij het ontwerpen van een complex programma.
Hiervoor refereer ik graag naar het boek Head First: Design Patterns dat ik zelf ook heb gebruikt als referentie tijdens mijn stage. Niet alleen is dit boek zeer informatief maar ook fijn om te lezen.

7. Literatuurlijst

-Scriptie Zhen Chen – Universiteit Twente - Workflow Managment Solution For A Security Managment Application

-Head First: Design patterns

8. Bedrijfs- en Persoonsgegevens

Bedrijf

FlexPosure BV
Koelenhofstraat 39
4004 JR, Tiel
flexposure.nl

Bedrijfsleider

Dieter Lentjes
+31(0) 344 634022
d.lentjes@flexposure.nl

Stagebegeleider

Wouter Lindehof
06-5147 5901
wouter.lindenhof@flexposure.nl

Student

Max Aalderink
06-17544 122
maxaalderink@gmail.com

Docentbegeleider

Joop Kaldeway
06-108 47 619
joop.kaldeway@hu.nl