Ontwikkeling van een Interactieve Model-gedreven E-formulier Generator

Namik Aktekin
n.aktekin@student.utwente.nl
eMAXX B.V., Hengelo, 23 november 2007

Afstudeerdatum: 26 november 2007

Begeleiders:
Dr. ir. Bedir Tekinerdogan (Universiteit Twente)
Prof. dr. ir. Mehmet Aksit (Universiteit Twente)
Ir. Anton Boerma (eMAXX B.V.)
Ir. Richard Scholten (eMAXX B.V.)

Samenvatting

In deze tijd waar Internet erg veel wordt gebruikt, neemt de waarde van e-formulieren toe. De webpagina's van de e-formulieren worden echter veelal handmatig geïmplementeerd en dat kost veel tijd. Om de benodigde tijd voor het opstellen en onderhouden van de e-formulieren te verminderen is onderzocht of de e-formulieren automatisch opgesteld kunnen worden aan de hand van datamodellen. In dit afstudeerverslag wordt onderzocht of modellen gebruikt kunnen worden om e-formulieren op te stellen en zo tijd te besparen met het opstellen en onderhouden van e-formulieren.

In dit onderzoek wordt de ontwikkeltechniek MDE gebruikt om de generator te ontwikkelen. Het datamodel voor de generator moet door een modelleringstechniek worden gemodelleerd. Het invullen van een e-formulier kan van eindgebruiker tot eindgebruiker verschillen. In dit onderzoek is de Feature Modeling modelleringstechniek gekozen omdat deze variaties kan modelleren. Vooral Cardinality-Based Feature Modeling is geschikt om datamodellen te modelleren die de data van de e-formulieren representeren.

Gebruikmakend van deze twee technieken zijn er drie generatoren ontwikkeld die van elkaar verschillen in complexiteit. De eerste generator genereert één webpagina vanuit het datamodel waarin alle benodigde gegevens aan de eindgebruiker worden gevraagd. De eindgebruiker vult deze webpagina in en verzend de ingevulde gegevens naar de webserver. De generator vult automatisch het datamodel met deze verzonden gegevens. Vervolgens wordt een rapport van het datamodel gegenereerd en dit rapport wordt verzonden naar de eMAXX Mid Office die voor de verdere afhandeling zorgt.

De tweede generator is een uitbreiding op de eerste generator. Hier wordt als eerst aan de eindgebruiker zijn identiteitsnummer gevraagd om met één van de functies van de eMAXX Mid Office opgeslagen gegevens van deze gebruiker op te halen, bijvoorbeeld uit een back office systeem. Met de functie *getPersonDetails* kunnen bijvoorbeeld de persoons- en adresgegevens van een persoon opgehaald worden wanneer het BSN nummer van deze persoon bekend is. Het resultaat van deze functie wordt naar het datamodel weggeschreven. Vervolgens genereert de generator een webpagina waarin alleen om die gegevens gevraagd wordt die nog geen waarde hebben. De rest van het proces is hetzelfde als de eerste generator.

Terwijl de eerste en de tweede generator de benodigde data op één webpagina aan de eindgebruiker vragen, worden er door de derde generator (indien nodig) meerdere webpagina's gegenereerd om de benodigde data in porties te vragen aan de eindgebruiker. Deze generator kan ook meerdere keren een functie aanroepen om opgeslagen gegevens op te halen. Een lijst van functies bevat functies die aangeroepen kunnen worden voor het desbetreffende datamodel. De generator beslist aan de hand van het datamodel en de lijst met functies wanneer er een functie wordt aangeroepen en wanneer er een webpagina wordt gegenereerd.

Aan het eind van dit onderzoek zijn de effectiviteit en de efficiëntie van de huidige ontwikkeltechniek, waarbij de e-formulieren handmatig worden opgesteld, geëvalueerd/getest. De effectiviteit en de efficientie van de nieuwe ontwikkeltechniek, waarbij er datamodellen worden opgesteld waarmee er vervolgens door de generator automatisch e-formulieren worden opgesteld, is ook geëvalueerd/getest. Vervolgens zijn de effectiviteit en de efficientie van de beide ontwikkeltechnieken met elkaar vergeleken om te kijken welke ontwikkeltechniek beter is om e-formulieren op te stellen.

De nieuwe ontwikkeltechniek is nog niet helemaal effectief. Er zitten nog fouten in de generator die uitgehaald moeten worden. Dit is meer een implementatie probleem dan de opgestelde nieuwe ontwikkeltechniek. Daar en tegen is de huidige ontwikkeltechniek wel effectief. Wel is de nieuwe ontwikkeltechniek efficiënter dan de huidige ontwikkeltechniek. Dit komt doordat er een deel van het werk (voor het opstellen van e-formulieren) geautomatiseerd is in de nieuwe ontwikkeltechniek.

Dit onderzoek toont aan dat aan de hand van datamodellen automatisch e-formulieren opgesteld kunnen worden. Werken met deze nieuwe ontwikkeltechniek kan ervoor zorgen dat er geld en tijd worden gespaard bij het opstellen en onderhouden van e-formulieren.

Verklarende woordenlijst en afkortingen

A-nummer	Administratienummer is het administratienummer als bedoeld in artikel 50 van de Wet gemeentelijke basisadministratie persoonsgegevens [Dig07].
BSN	Burgerservicenummer is een uniek persoonsnummer dat als nummer gelijk is aan het sofinummer (**so**ciaal-**fi**scaal nummer). Het heeft echter een ander wettelijk kader waardoor een breder gebruik mogelijk is [Wik07a].
CBFM	Cardinality-Based Feature Modeling is een uitbreiding op de oorspronkelijke Feature Oriented Domain Analysis (FODA) [Kan90].
CSS	Cascading Style Sheets is een techniek voor de stijl (vormgeving) van webpagina's. De informatie over de vormgeving wordt toegevoegd aan de HTML-code van het document [Wik07p].
DigiD	Digitale Identiteit. Met DigiD (spreek uit: die-gie-dee) kunnen burgers en bedrijven met één inlogcode bij elektronische diensten van steeds meer overheidsinstellingen terecht [Dig06].
E-formulier	Elektronische formulier is een reeks webpagina’s waarop velden staan die ingevuld kunnen worden.
FTP	File Transfer Protocol is een protocol dat uitwisseling van bestanden tussen computers vergemakkelijkt. Het standaardiseert een aantal handelingen die tussen besturingssystemen vaak verschillen.
FODA	Feature Oriented Domain Analysis [Kan90]
GBA	Gemeentelijke Basisadministratie Persoonsgegevens is in Nederland de benaming voor de boekhouding van bepaalde gegevens die iedere gemeente bijhoudt omtrent alle personen die in de gemeente gevestigd zijn of waren zoals: naam, geboortedatum, adres etc. [Wik07h].
HTML	HyperText Markup Language is een taal (geen programmeertaal) voor de opmaak van documenten. HTML wordt vooral gebruikt op het internet, om webpagina's te tonen [Wik07l].
HTTP	Hypertext Transfer Protocol is het protocol voor de communicatie tussen een webclient (meestal een webbrowser) en een webserver. Dit protocol wordt niet alleen veel op het World Wide Web gebruikt, maar ook op lokale netwerken (we spreken dan van een intranet) [Wik07c].
HTTPS	HTTP Secure is een uitbreiding op het HTTP-protocol met als doel een veilige uitwisseling van gegevens. Bij gebruik van HTTPS wordt de data versleuteld, waardoor het voor een buitenstaander, bijvoorbeeld iemand die afluistert, onmogelijk zou moeten zijn om te weten welke gegevens verstuurd worden [Wik07e].
JSP	Java Server Pages is een manier om dynamisch HTML, XML of andere inhoud te genereren op basis van statische en dynamische elementen. Dit wordt gedaan door Java code en bepaalde voorgedefinieerde acties op te nemen in de statische inhoud [Wik07g].
MDA	Model Driven Architecture is een OMG standaard die een transformatie uitgevoerd van PIM (Platform Independant Model) naar PSM (Platform Specific Model).

MDE	Model Driven Engineering is een model gedreven ontwikkeltechniek waarbij het model centraal staat. Deze ontwikkeltechniek probeert een framework neer te zetten om (1) duidelijke methodologiëen te definiëren, (2) systeem te ontwikkelen van elke level of abstractie en om (3) tests en validatie's te organiseren en te automatiseren [Fon04].
OMG	Object Management Group is een open ledenconsortium zonder winstbejag. Het consortium produceert en onderhoudt computerspecificaties voor interoperabele toepassingen ten behoeve van ondernemingen [Wik07k].
MOF	Meta-Object Facility is een OMG standaard voor MDE [Wik07i].
SMTP	Simple Mail Transfer Protocol is de de facto-standaard voor het versturen van e-mail over het Internet [Wik07d].
SOA	Service-Oriented Architecture is een softwarearchitectuurmodel. Een volgens SOA opgebouwd systeem bestaat meestal uit twee soorten componenten. Enerzijds de componenten die een dienst aanbieden, de "services", anderzijds de componenten die de uitwisseling van informatie tussen services regelen, de "Enterprise Service Bus" [Wik07m].
SOAP	Simple Object Access Protocol is een (computer)protocol dat wordt gebruikt voor communicatie tussen verschillende componenten. SOAP wordt gesteund door een groot aantal bedrijven waaronder Sun, IBM, Microsoft, BEA, Oracle, Apache. SOAP is een protocol dat XML-berichten stuurt, meestal over HTTP, maar ook over SMTP, HTTPS of FTP [Wik06c].
Sofinummer	Het sofinummer is in Nederland een door de rijksbelastingdienst aan een natuurlijke persoon toegekend identificerend nummer [Wik07b].
SPL	Software Product Line verwijst naar techniekmethodes, hulpmiddelen en technieken om een collectie van gelijksoortige software systemen te creëren [Wik07n].
XML	eXtensible Markup Language is een standaard voor het definiëren van formele markup-talen voor de representatie van gestructureerde gegevens in de vorm van platte tekst [Wik06a].
XSD	XML Schema Definition is een aanbeveling van het Consortium van World Wide Web (W3C), die specificeert hoe de elementen in een XML document formeel beschreven moet worden [SEA06].
XSL	Extensible Stylesheet Language is een formele taal waarin beschreven kan worden, hoe XML-documenten geformatteerd moeten worden [Wik07o].
XSLT	Extensible Stylesheet Language Transformations is een standaard voor het omzetten van de informatie in een XML document naar een ander formaat, of een anders gestructureerde XML document [Wik06b].

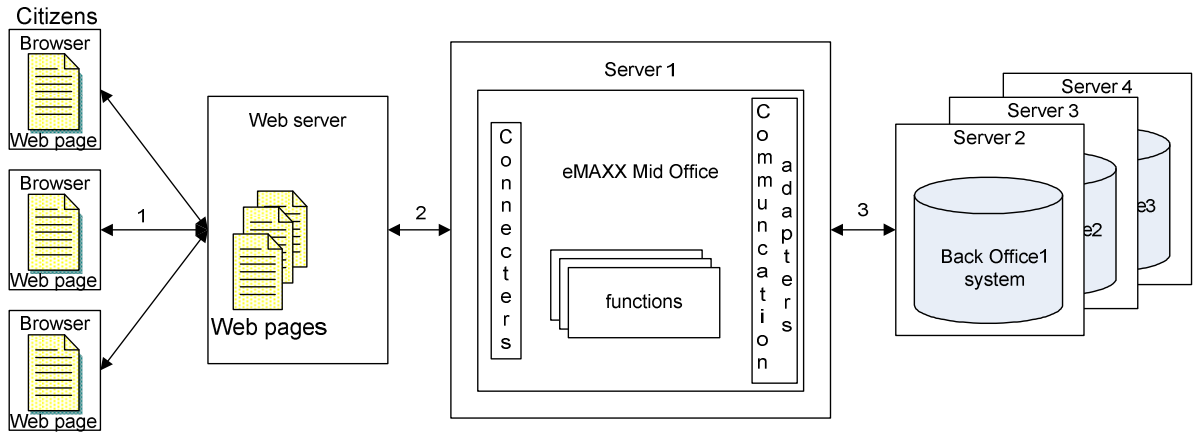
1 Inleiding

Het onderzoek dat in dit verslag is beschreven is uitgevoerd bij eMAXX B.V. eMAXX is een snel groeiend middelgroot ICT-bedrijf, gevestigd in Enschede dat zich richt op het aandragen van oplossingen voor de elektronische overheid en de agro-sector. Met ‘eMAXX Mid Office’ is eMAXX een groeiende marktleider op het gebied van transactieverwerking binnen digitale loketten. Deze software wordt aangetroffen bij veel grote gemeenten.

De eMAXX Mid Office biedt als technische oplossing mogelijkheden om als 'enabler' te fungeren voor het verbeteren van dienstverlening en interne prestaties. eMAXX gaat hierbij uit van een opsplitsing van het werkproces in een front- en back office proces. De Mid Office vormt de basis voor een op diensten gebaseerde architectuur (Service-Oriented Architecture), ook wel SOA genoemd. Daarbij gaat het om het realiseren van een zgn. 'service bus' waarin alle verschillende applicaties toegevoegd kunnen worden. De eMAXX Mid Office ondersteunt deze concepten en beoogt hiermee een schaalbare en toekomstgerichte architectuur. eMAXX maakt veel gebruik van software componenten om formulieren en werkprocessen te automatiseren.

1.1 eMAXX Mid office

Elke gemeente heeft zijn eigen eMAXX Mid Office die op een server van de desbetreffende gemeente draait. Verder heeft elke gemeente één of meerdere webservers en één of meerdere back office systemen, die op servers draaien. Een e-formulier is een reeks samenhangende webpagina’s waarop velden staan die ingevuld kunnen worden. In de huidige werkwijze staan de webpagina’s, die handmatig geïmplementeerd zijn, op een webserver van de gemeente (zie Figuur 1.1). Via de communicatielijn 1 in Figuur 1.1 kunnen de burgers met behulp van een web browser bij deze webpagina’s komen en zo hun aanvragen bij de desbetreffende gemeente doen.



Figuur 1.1: huidige weergave van de servers van een gemeente

De huidige webpagina’s van eMAXX maken gebruik van de eMAXX Mid Office functies daar waar het nodig is. Deze functies halen bijvoorbeeld gegevens uit het GBA (Gemeentelijke Basisadministratie Persoonsgegevens) op. GBA is in Nederland de benaming voor de boekhouding van bepaalde gegevens die iedere gemeente bijhoudt omtrent alle personen die in de gemeente gevestigd zijn of waren zoals: naam, geboortedatum, adres etc. [Wik07h].

De eMAXX Mid Office maakt intern (en ook extern als interface) gebruik van XML (eXtensible Markup Language) berichten. XML is een standaard voor het definiëren van formele markup-talen voor de representatie van gestructureerde gegevens in de vorm van platte tekst [Wik06a]. De communicatie van de back office systemen met de eMAXX Mid Office wordt met behulp van communicatie adapters via de communicatielijn 3 in Figuur 1.1 bewerkstelligd. Deze communicatie adapters zorgen ervoor dat de informatie die verzonden wordt naar de eMAXX Mid Office wordt omgezet in een XML bericht. De informatie die de eMAXX Mid Office verstuurt naar back office systemen wordt omgezet naar het formaat dat de betreffende systemen gebruiken.

De communicatie van webpagina's met de eMAXX Mid Office gebeurd met behulp van connectors via de communicatielijn 2 in Figuur 1.1. Er zijn twee soorten connectoren die eMAXX Mid Office gebruikt namelijk de HTTP (Hypertext Transfer Protocol) en de SOAP (Simple Object Access Protocol) connectoren. HTTP is het protocol voor de communicatie tussen een webclient (meestal een webbrowser) en een webserver. Dit protocol wordt niet alleen veel op het World Wide Web gebruikt, maar ook op lokale netwerken (we spreken dan van een intranet) [Wik07c]. SOAP is een computerprotocol dat wordt gebruikt voor communicatie tussen verschillende componenten. SOAP wordt gesteund door een groot aantal bedrijven waaronder Sun, IBM, Microsoft, BEA, Oracle, Apache. SOAP is een protocol dat XML-berichten stuurt, meestal over HTTP, maar bijvoorbeeld ook over SMTP [Wik07d], HTTPS [Wik07e] of FTP [Wik07f] [Wik06c].

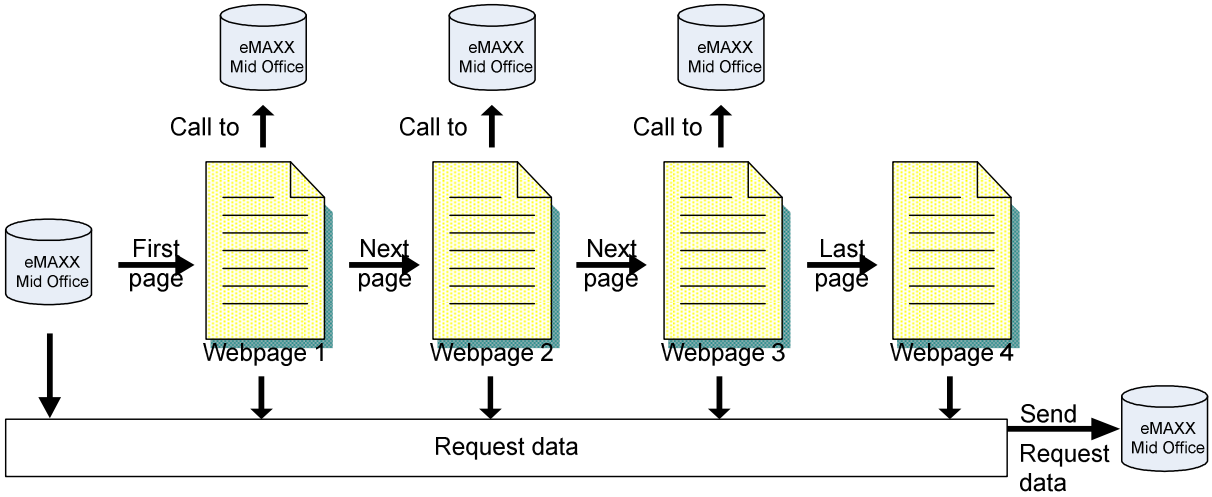
De webpagina's van eMAXX zijn JSP (JavaServer Pages) pagina's. JSP is een manier om dynamisch HTML [Wik07l], XML of andere inhoud te genereren op basis van statische en dynamische elementen. Dit wordt gedaan door Java code en bepaalde voorgedefinieerde acties op te nemen in de statische inhoud [Wik07g]. De informatie van een JSP pagina wordt eerst verstuurd naar een Java class, die deze informatie in een XML bericht zet en van daaruit wordt, eventueel met behulp van andere Java classes, dit XML bericht naar de eMAXX Mid Office verstuurd.

Een groot voordeel van het gebruik van de eMAXX Mid Office is dat de webpagina's en back office systemen zijn gescheiden. Er kunnen verschillende soorten back office systemen zijn die anders werken en verschillende interfaces hebben. Voor al deze verschillende back office systemen hoeven geen aparte webpagina's worden geïmplementeerd. De webpagina's communiceren alleen maar met de eMAXX Mid Office en hoeven zo geen rekening te houden met de interfaces van de back office systemen.

1.2 Probleemstelling

eMAXX richt zich op het aandragen van oplossingen voor de elektronische overheid. Een deel van deze oplossingen zijn gemeentelijke e-formulieren. Met deze e-formulieren kunnen de burgers digitaal hun aanvragen bij de gemeenten doen. Voorbeelden van aanvragen zijn: afspraak maken, een verhuizing doorgeven of een bouwvergunning aanvragen. De webpagina's van de e-formulieren voor deze aanvragen worden nu door eMAXX handmatig geïmplementeerd.

De huidige werkwijze van e-formulieren voor het aanvragen van producten/diensten is in Figuur 1.2 (zie volgende pagina) abstract weergegeven.



Figuur 1.2: abstracte weergave van werking huidige e-formulieren van eMAXX

Allereerst worden via de eMAXX Mid Office de gegevens van de burger die de aanvraag doet opgehaald uit het GBA (Gemeentelijke Basis Administratie). Vervolgens worden de resterende gegevens via webpagina's aan de burger gevraagd. In een webpagina kan een aanroep naar een eMAXX Mid Office functie gedaan worden zoals *getPersonDetails* die persoonsgegeven ophaalt uit GBA. Al deze gegevens worden in de sessie van de burger opgeslagen op de webserver als aanvraaggegevens (zie Figuur 1.2). De aanvraaggegevens bestaan uit alle gegegevens die gebruikt gaan worden om de aanvraag uit te voeren. Nadat de burger op de laatste webpagina de gegevens heeft bevestigd worden deze verstuurd naar de eMAXX Mid Office. Deze zorgt vervolgens voor de verdere afhandeling van de aanvraag.

Het e-formulier voor een aanvraag kan van gemeente tot gemeente verschillen. Het verschil zit met name in de gevraagde gegevens voor de aanvraag. De ene gemeente vraagt om meer/minder gegevens dan een ander gemeente. Een gemeente kan bijvoorbeeld om een identificatienummer van de burger vragen terwijl een ander gemeente dit niet vraagt. Ook worden er wijzigingen aangebracht in deze e-formulieren tijdens onderhoud, zoals het weghalen/toevoegen van een veld of het veranderen van de volgorde van de velden.

De problemen van handmatig implementeren van de webpagina's van deze e-formulieren zijn:

- ***Realiseren van e-formulieren kost veel tijd***

  Bij het realiseren van een e-formulier wordt de plaats van velden in een webpagina en de volgorde van webpagina's handmatig geïmplementeerd wat veel tijd kost.

- ***Onderhoud van webpagina's kost veel tijd***

  Bij het onderhouden van een webpagina worden kleine verschillen aangebracht maar toch kost het onderhoud relatief veel tijd omdat ook deze verschillen handmatig worden aangebracht.

- ***Relatie van e-formulieren met back office systemen***

  Er wordt geen rekening gehouden met variaties in volledigheid of aanwezigheid van brongegevens waardoor een aanvraag niet ingediend kan worden.

Om deze problemen op te lossen, wil eMAXX een ontwerp en een prototype van een nieuwe e-formulieren engine ontwikkelen. Deze engine moet, aan de hand van gevraagde gegevens (het datamodel) en gebruik makend van eMAXX Mid Office functies, e-formulieren genereren waardoor tijd wordt gespaard bij het ontwikkelen en onderhouden van e-formulieren.

1.3 Benadering

Het doel van dit onderzoek is:

> *Onderzoeken of het mogelijk is om aan de hand van datamodellen automatisch webpagina's voor e-formulieren te genereren met als doel tijd en geld te besparen bij het ontwikkelen en onderhouden van deze webpagina's.*

Om deze engine te realiseren moet eerst een modelleringtechniek voor datamodellen worden gekozen om de aanvragen in te modelleren. De gekozen modelleringtechniek moet aan de volgende eisen voldoen om de doelstelling te kunnen halen:

- ***De modelleringtechniek moet simpel zijn***

  De modelleringtechniek moet simpel zijn om tijd te besparen. Als de modelleringtechniek te complex is zal het opstellen en onderhouden van de datamodellen veel tijd kosten en zal het originele probleem niet verholpen zijn. Dit zal er alleen maar voor zorgen dat de plaats van het probleem verschoven wordt van het implementeren en onderhouden van webpagina's naar het opstellen en onderhouden van datamodellen.

- ***De modelleringtechniek moet herbruikbare deelmodellen ondersteunen***

  De modelleringtechniek moeten herbruikbare deelmodellen ondersteunen die vaker voorkomen. Door gebruik te maken van herbruikbare deelmodellen, kan er tijd worden bespaard tijdens het opstellen en het onderhouden van de datamodellen.
- ***De modelleringtechniek moet expressief genoeg zijn om eigenschappen/beperkingen te modelleren***

  De modelleringtechniek moet expressief genoeg zijn om de volgende eigenschappen/beperkingen van het datamodel te kunnen beschrijven die samen met de begeleiders zijn opgesteld:

  1. **Het datamodel moet een structuur hebben**

     Het datamodel moet een structuur hebben omdat er elementen zijn die meer dan één keer voorkomen. Zonder structuur zal het niet duidelijk zijn waar bepaalde elementen voor dienen. Zo kunnen bijvoorbeeld de gegevens van de aanvrager en de medeverhuizers in de war raken als het datamodel geen structuur heeft.

2. **De elementen van het datamodel moeten getypeerd kunnen worden**

   De elementen moeten getypeerd zijn om niet in conflict te komen met de systemen die later deze elementen gaan gebruiken. Ook is dit handig om controle op de elementen uit te voeren.

3. **De elementen van het datamodel moeten verplicht of optioneel kunnen zijn**

   In het datamodel moeten verplichte/optionele elementen weergegeven kunnen worden zodat het duidelijk is voor de burgers welke elementen verplicht/optioneel zijn. Ook kunnen er controles worden uitgevoerd op verplichte elementen om fouten bij de systemen waarnaar de elementen worden opgestuurd te voorkomen. Bijvoorbeeld als een burger geen medeverhuizers heeft, dan hoeft hij of zij geen gegevens van de medeverhuizers op te geven. Daarentegen moet de burger wel altijd zijn eigen sofinummer opgeven.

In dit onderzoek is er voor CBFM (Cardinality-Based Feature Modeling) gekozen als modelleringtechniek. In sectie 3.2 wordt deze modelleringtechniek uitvoerig besproken. In hoofdstuk 3 zal de motivatie van deze keuze worden besproken.

Verder zal een ontwikkelmethodiek gebruikt worden om de generator te ontwikkelen. De gekozen ontwikkeltechniek moet met modellen kunnen werken en deze kunnen transformeren. De generator zal immers webpagina's genereren aan de hand van datamodellen.

In dit onderzoek is er voor MDE (Model Driven Engineering) gekozen om de generator te ontwikkelen. In sectie 3.1 wordt deze ontwikkeltechniek uitvoerig besproken. In hoofdstuk 3 zal de motivatie van deze keuze worden weergegeven

1.4 Eigen bijdrage

De eigen bijdragen van dit onderzoek zijn:

1 Architectuur ontworden voor een generator

In dit onderzoek is een architectuur ontworpen voor een generator die aan de hand van een datamodel webpagina's genereert voor het invullen van een e-formulier. Het datamodel is een model met regels waaraan de gegevens moeten voldoen. Er zijn twee eigenschappen die deze generator uniek maken ten opzichte van andere e-formulierengenerators.

De eerste eigenschap is dat deze generator de webpagina's op een webserver genereert waardoor deze meteen gebruikt kunnen worden. De andere generators genereren webpagina's offline die eventueel handmatig bijgewerkt worden voordat deze online gebruikt kunnen worden.

De tweede eigenschap van de ontwikkelde generator is dat deze interactie heeft met de eindgebruiker. Het aanvraagproces is het doorlopen van de webpagina's om een e-formulier in te vullen. De generator kan aan de hand van de ingevulde gegevens van de eindgebruiker samen met de constraints van het datamodel beslissen wat er in de volgende stap moet gebeuren. De volgende stap kan een aanroep zijn waarmee gegevens van een extern systeem worden opgehaald (of weggeschreven) of het kan de generatie zijn van de volgende webpagina van het e-formulier. De andere generators voeren interactie uit met een ontwerper i.p.v. met een eindgebruiker. Daarbij dient de interactie meestal voor het expliciet instellen van het uiterlijk en de eigenschappen van de velden van een webpagina, i.p.v. dat dit impliciet door de generator gebeurt.

2 Opstellen van het proces voor het automatisch genereren van webpagina's voor het invullen van e-formulieren

Er is een stappenplan opgesteld voor het ontwerpen van de uiteindelijke generator. In elke stap wordt een ander probleem bij het automatisch genereren van webpagina's beschreven en opgelost:

1.1 De model gedreven e-formulier generator zonder interactie

Er is een proces opgesteld voor deze generator, waarbij deze generator vanuit een datamodel één webpagina genereert.

1.2 De model gedreven e-formulier generator met ondersteuning van de eMAXX Mid Office

Er is een proces opgesteld voor deze generator, die een uitbreiding is op de vorige generator, waarbij deze generator interactie met de eMAXX Mid Office heeft en daarmee opgeslagen gegevens bijvoorbeeld uit een back office systeem ophaalt.

1.3 De model gedreven e-formulier generator met workflow

Er is een proces opgesteld voor deze generator, die een uitbreiding is op de vorige generator, waarbij deze generator meerdere webpagina's genereert en eventueel vaker (indien nodig) een interactie met de eMAXX Mid Office heeft.

2 Ontwikkelen van de generatoren

De opgestelde processen van de bovenstaande generatoren zijn ook geïmplementeerd. De generatoren zijn met behulp van het Eclipse platform gebouwd. Eclipse is een open-source framework voor software-ontwikkelomgevingen. Daarnaast is Eclipse ook zelfstandig te gebruiken als krachtige Java-ontwikkelomgeving. Eclipse heeft een open structuur waardoor het mogelijk is de functionaliteit uit te breiden door middel van plug-ins [Wik07j]. Verder is er gebruik gemaakt van JSP pagina's en van Java classes. Voor de transformaties is er gebruik gemaakt van XSLT (Extensible Stylesheet Language Transformations). XSLT is een standaard voor het omzetten van een XML document naar een ander formaat, of een anders gestructureerd XML document [Wik06b].

3 Opstellen van datamodellen voor e-formulieren

Er zijn datamodellen voor e-formulieren opgesteld. De opgestelde datamodellen zijn uiteindelijk in XML documenten opgeslagen. Deze datamodellen voldoen aan de opgestelde eisen die in sectie 1.3 zijn weergegeven.

4 Evaluatie middels een case studie

De benodigde tijden voor het opstellen van e-formulieren met beide ontwikkeltechnieken zijn met elkaar vergeleken om te evalueren/testen of de nieuwe ontwikkeltechniek tijd bespaart bij het opstellen van e-formulieren. Ook is de werking van de generator geëvalueerd/getest door het invullen van de e-formulieren in combinatie met de ontwikkelde generator.

1.5 Leeswijzer

Allereerst zullen in hoofdstuk 2 de probleemstelling en de substellingen worden besproken. In hoofdstuk 3 zal de theorie van MDE en CBFM worden besproken. Ook zal de terminologie van deze technieken in dit hoofdstuk worden besproken. In hoofdstuk 4 zal de model gedreven e-formulier generator zonder interactie worden besproken. Dit is het proces waarin een datamodel wordt getransformeerd naar één webpagina. In hoofdstuk 5 zal de model gedreven e-formulier generator met ondersteuning van de eMAXX Mid office worden besproken. Dit is een uitbreiding op de generator van hoofdstuk 4 waarbij er eerst een aanroep naar de eMAXX Mid Office wordt gedaan om opgeslagen gegevens op te halen voordat een webpagina wordt gegenereerd. In hoofdstuk 6 zal de model gedreven e-formulier generator met workflow worden besproken. Dit is een uitbreiding van de generator van hoofdstuk 5 waarbij er eMAXX Mid office functies kunnen worden aangeroepen en meerdere webpagina's kunnen worden gegenereerd. In hoofdstuk 7 zal de evalueren/testen van de model gedreven e-formulier generator worden besproken. Ten slotte zullen in hoofdstuk 8 de conclusies en eventuele aanbevelingen worden gegeven.

2 Probleem stelling

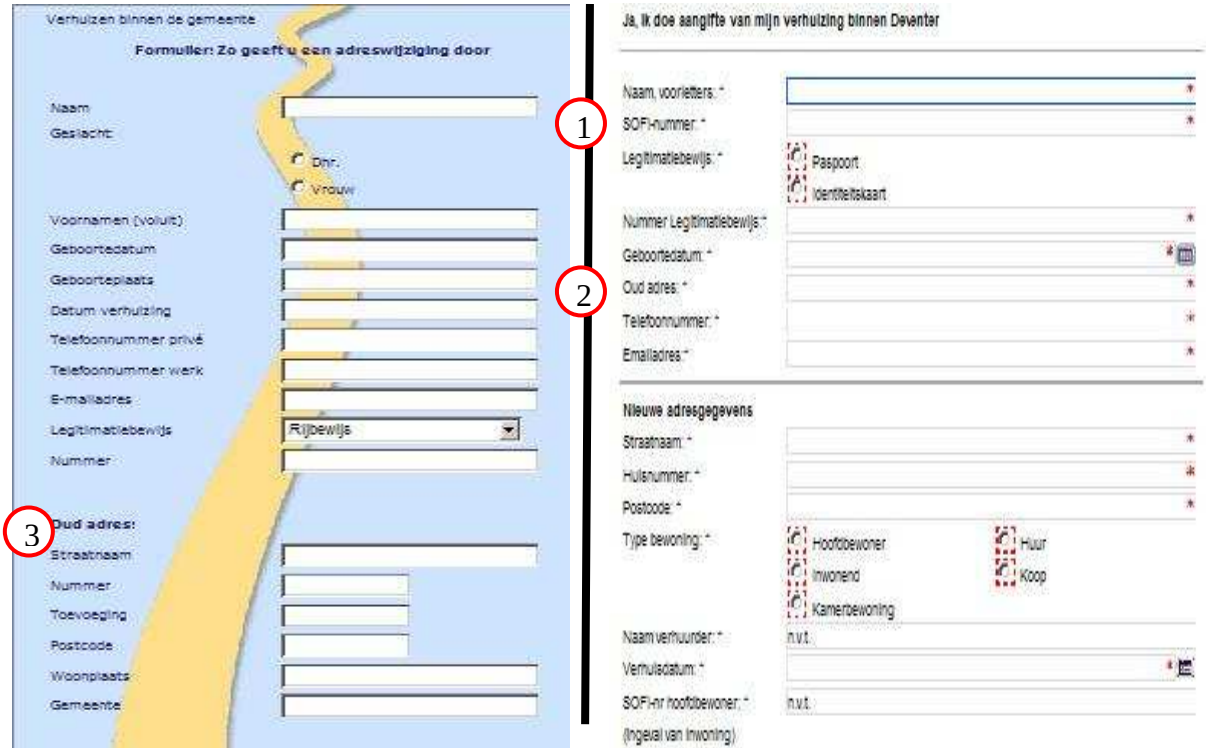
Zoals eerder aangegeven implementeert eMAXX handmatig de webpagina's van de e-formulieren voor de gemeenten. Elke gemeente heeft zo zijn eigen wensen en eisen met betrekking tot deze e-formulieren voor dezelfde aanvraag, waardoor deze van elkaar kunnen verschillen. In sectie 2.1 zullen de problemen van het handmatig implementeren van de webpagina's van e-formulieren worden besproken.

Binnen eMAXX is er eerder een onderzoek gedaan hoe gemeentelijke e-formulieren verbeterd kunnen worden [Kui06]. In dat onderzoek werd een aantal problemen benoemd die de burgers ondervonden bij het invullen van gemeentelijke e-formulieren. Bij het ontwikkelen van de generator is het handig om rekening te houden met deze problemen zodat deze (deels) opgelost kunnen worden. In sectie 2.2 zullen deze gebruikersproblemen verder worden besproken.

In sectie 2.3 worden de doelstellingen van dit onderzoek besproken. Ten slotte zal in sectie 2.4 het stappenplan voor het realiseren van de generator worden besproken.

2.1 Problemen van handmatig implementeren van webpagina's van e-formulieren

In sectie 1.2 zijn de verschillende wensen en eisen van de gemeenten met betrekking tot gemeentelijke e-formulieren voor dezelfde aanvraag beschreven. In Figuur 2.1 is een voorbeeld gegeven van twee verschillende gemeenten met verschillende wensen en eisen voor dezelfde aanvraag namelijk: *doorgeven van verhuizing binnen gemeente*. In dit figuur zijn de webpagina's van gemeenten Hoogeveen (links) en Deventer (rechts) weergegeven.



Figuur 2.1: webpagina's van gemeentelijke e-formulieren voor het doorgeven van verhuizing van de gemeente Hoogeveen [Gho07] (links) en van de gemeente Deventer [Gde07] (rechts)

Er zijn duidelijke verschillen tussen deze webpagina's. Een verschil is bijvoorbeeld dat er bij gemeente Hoogeveen het sofinummer niet wordt gevraagd terwijl bij gemeente Deventer het verplicht is om een sofinummer op te geven (1). Een ander verschil is de manier waarop gegevens gevraagd worden. De webpagina van Deventer heeft voor het invullen van het oude adres maar één veld (2) terwijl de webpagina van Hoogeveen daar meerdere velden (3) voor heeft. Zo kunnen er meerdere van dit soort kleine verschillen tussen de webpagina's van gemeentelijke e-formulieren zijn.

2.1.1 Realiseren en onderhoud van webpagina's van e-formulieren

Het hoofdprobleem is dat het realiseren en het onderhouden van e-formulieren erg veel tijd kost. Dit komt doordat de wijzigingen in alle lagen van de applicatie aanpassingen vergen. Om deze wijzigingen door te voeren moet men goed weten waar de aanpassingen invloed hebben en hoe de applicatie in elkaar zit. Doordat er op meerdere plekken wijzigingen doorgevoerd moeten worden, is de applicatie foutgevoeliger. In elke laag moeten de volgende wijzigingen zorgvuldig worden uitgevoerd.

Gegevens:

- ***aanpassen van data objecten***

De gegevenslaag is de laag aan de server kant waarin met ingevoerde gegevens wordt gewerkt. In deze laag moeten de data objecten aangepast worden om de wijzigingen door te voeren. Dat wil zeggen dat er objecten aangemaakt moeten worden of objecten verwijderd moeten worden om de wijzigingen door te voeren.

Weergave:

- ***aanpassen van de webpagina's van het e-formulier***
- ***aanpassen van het overzichtsscherm***

De weergavelaag is de laag die de burger op de webpagina ziet. De wijzigingen moeten doorgevoerd worden op de webpagina's. Denk hierbij aan het weghalen van velden of toevoegen van velden die gerelateerd zijn aan de te wijzigen gegevens. Daarnaast zal ook het overzichtsscherm aangepast moeten worden. Het overzichtsscherm is het scherm waarop de aanvraaggegevens getoond worden aan de burger zodat de burger deze kan controleren.

Controller:

- ***aanpassen van volgorde van de webpagina's***
- ***toevoegen nieuwe webpagina's***
- ***aanpassen van validatie***
- ***aanpassen van het vullen van XML berichten***

De controllerlaag is de laag waarin de volgorde van de webpagina's wordt bepaald en deze moet dan ook worden aangepast. De wijzigingen kunnen invloed hebben op de volgorde van de webpagina's. Het kan zijn dat de wijzigingen op een nieuwe webpagina moet komen, dan moet een nieuwe webpagina toegevoegd worden. Ook moet de volgorde van de webpagina's aangepast worden om de nieuwe webpagina aan de rest te koppelen. De validatie van de gewijzigde gegevens moet ook aangepast worden. Wanneer de gewijzigde gegevens ook nog eens opgeslagen worden met behulp van de eMAXX Mid Office, dan moeten deze wijzigingen ook worden doorgevoerd voor het invullen van de XML berichten waarvan de eMAXX Mid Office intern gebruik maakt.

2.1.2 Relatie tot back office systemen

Een ander probleem is dat de huidige manier van implementeren van webpagina's van e-formulieren geen rekening houdt met variaties in volledigheid of aanwezigheid van brongegevens. Momenteel wordt er mogelijk een foutmelding gegeven indien een aanroep niet uitgevoerd kan worden. Dan kan de burger niet verder met aanvraag terwijl dit wel mogelijk kan zijn door de niet ophaalbare gegevens alsnog aan de burger te vragen. Ook wordt er geen rekening gehouden met opgehaalde gegevens die incompleet zijn. Als deze ontbrekende gegevens verplichte gegevens zijn, zal het systeem in de huidige situatie een foutmelding geven bij het valideren van de aanvraaggegevens en kan de burger de aanvraag niet afronden.

2.2 Problemen die de burgers ondervinden bij het invullen van gemeentelijke e-formulieren

eMAXX heeft een onderzoek laten verrichten met betrekking tot het gemeentelijke e-formulieren. Daarbij is gekeken naar hoe de gemeentelijke e-formulieren verbeterd kunnen worden. In het onderzoek [Kui06] zijn een aantal problemen van burgers voor het invullen van e-formulieren benoemd. Er wordt met twee van deze problemen rekening gehouden in dit onderzoek; het *oriëntatie-* en het *selectieprobleem*. Deze problemen zullen in respectievelijk sectie 2.2.1 en 2.2.2 worden besproken.

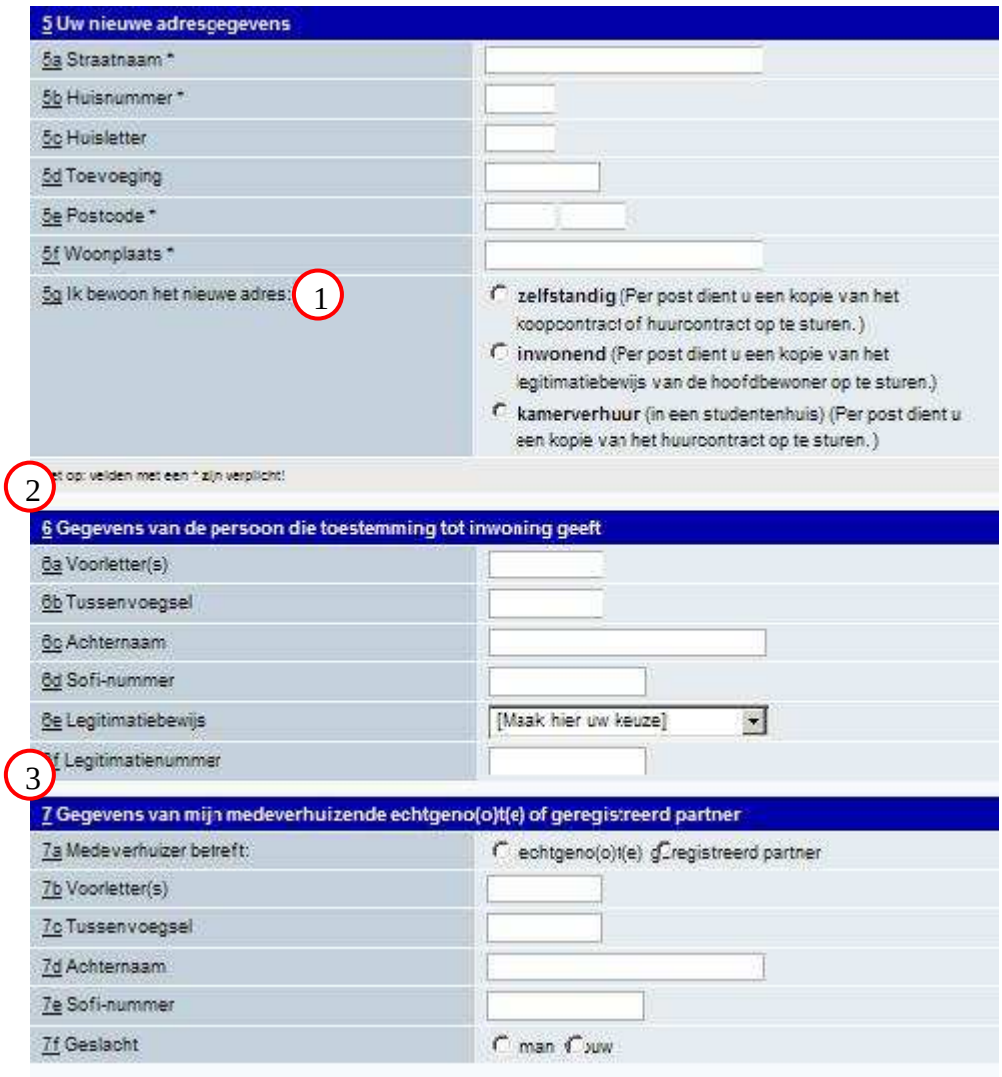
2.2.1 Oriëntatieprobleem

Wanneer alle velden zoals in Figuur 2.1 (zie pagina 15) op één webpagina worden weergegeven, is er geen goed overzicht. Het zou beter zijn als de gevraagde gegevens over meerdere webpagina's worden verdeeld. Daarbij is het erg belangrijk dat de samenhangende gegevens op dezelfde webpagina worden weergegeven zodat de burger niet in de war raakt. Wanneer bijvoorbeeld op een webpagina de naam en de achternaam van de aanvrager worden gevraagd en op de volgende webpagina de naam en achternaam van de meeverhuizers en de geboortedatum van de aanvrager, dan kan de burger in de war raken. eMAXX lost dit probleem op door de samenhangende velden handmatig op één webpagina te groeperen. Met het oriëntatieprobleem wordt rekening gehouden om zodoende bruikbare e-formulieren te genereren.

2.2.2 Selectieprobleem

Op een webpagina kunnen gegevens worden gevraagd aan de burger die niet van toepassing zijn op hem of haar. Ook kunnen er gegevens zijn opgeslagen in een back office systeem die opgehaald kunnen worden. Het is dan overbodig om deze gegevens alsnog aan de burger te vragen. Deze selectieproblemen zorgen ervoor dat de burgers meer tijd kwijt kan zijn met het invullen van een e-formulier. eMAXX maakt ook gebruik van opgeslagen gegevens, maar houdt geen rekening met incomplete opgehaalde gegevens en/of het uitvallen van back office systeem, waarin de gegevens zijn opgeslagen, waardoor er in een dergelijke situatie geen aanvraag ingediend kan worden.

In Figuur 2.2 is een voorbeeld gegeven van een deel van een e-formulier voor het doorgeven van een verhuizing. De burger moet de velden invullen die relevant zijn voor hem of haar maar het is niet altijd duidelijk welke dat zijn. Zo kan het zijn dat de burger zelfstandig gaat inwonen (1), maar dat hij of zij alsnog de velden te zien krijgt waarbij de persoonsgegevens van de persoon, die toestemming geeft tot inwonen zou moeten geven, worden gevraagd (2). Dit geld ook voor velden die bestemd zijn voor medeverhuizers (3) terwijl de burger misschien geen medeverhuizers heeft. Dit probleem kan opgelost worden door alleen die velden die relevant zijn voor de burger te tonen.



Figuur 2.2: statische gemeentelijk e-formulier voor het doorgeven van verhuizingen[Ema06a]

eMAXX lost dit probleem op door gebruik te maken van (Java)scripts in webpagina's om deze dynamisch te maken waarbij alleen de velden worden getoond die relevant zijn voor de burger. In Figuur 2.3 is een voorbeeld gegeven van een dynamische webpagina voor het doorgeven van een verhuizing waarbij de wijze van bewoning wordt gevraagd. Wanneer de burger zelfstandig gaat wonen (1), dan gaat hij of zij verder op de volgende webpagina met invullen.

Figuur 2.3: e-formulier voor het doorgeven van verhuizing, webpagina 2: wijze van bewoning [Ema06b]

Echter, als de burger voor *Inwonend* kiest zoals in Figuur 2.4 (1), dan komen er extra velden bij (2). Door deze keuze worden er geen overbodige velden getoond op de webpagina's van e-formulieren en zodoende is het selectie probleem opgelost.

Figuur 2.4: e-formulier voor het doorgeven van verhuizing, webpagina 2: wijze van bewoning (inwonend) [Ema06b]

2.3 Doelstelling

De doelstelling van dit onderzoek is al in sectie 1.3 beschreven. Het gaat in dit onderzoek om hoe de generator aan de hand van een datamodel webpagina(s) voor een e-formulier gaat genereren. Hierbij wordt niet op de lay-out van de webpagina gelet; dit valt buiten het kader van dit onderzoek. Dit onderzoek zal proberen oplossingen te bieden voor de problemen die in Tabel 2.1 zijn weergegeven.

	Problemen waarvoor een oplossing moet komen
Problemen voor eMAXX	- Implementatie en onderhoudt kost veel tijd - Relatie tot back office systemen
Problemen voor de burgers	- Selectieprobleem - Oriëntatieprobleem

Tabel 2.1: problemen waarvoor een oplossing moet komen.

Het uitgangspunt van de te ontwerpen generator moet een datamodel van een e-formulier zijn waarin de gevraagde gegevens en hun relaties staan gemodelleerd. De generator gaat aan de hand van dit datamodel, en gebruikmakend van de eMAXX Mid Office functies, webpagina's genereren. Als er opgeslagen gegevens zijn, zullen deze gegevens met eMAXX Mid Office functies opgehaald worden. Waneer de gegevens niet opgehaald kunnen worden, worden deze naar webpagina's getransformeerd om zodoende deze ontbrekende gegevens aan de burger te vragen.

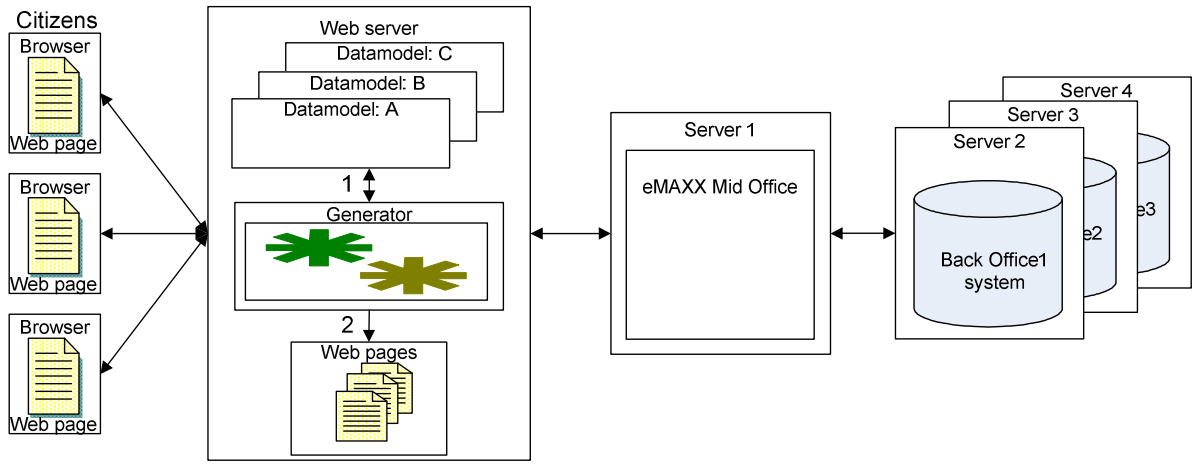
Om een goed werkende generator te ontwikkelen, die ook nog eens rekening houdt met de problemen die de burgers kunnen ondervinden, zullen de volgende vragen beantwoord moeten worden.

- ***Hoe wordt er aan de hand van een datamodel webpagina(s) van een e-formulier gegenereerd?***
    - **Wat is het datamodel?**
    - **Hoe wordt het datamodel gerepresenteerd?**
    - **Hoe wordt het datamodel getransformeerd naar webpagina's?**
- ***Hoe wordt er gebruik gemaakt van eMAXX Mid Office functies?***
    - **Welke eMAXX Mid Office functionaliteiten zijn er te gebruiken?**
    - **Wanneer wordt er een functie gebruikt?**
    - **Hoe wordt de volgorde van de functies bepaald?**
- ***Hoe wordt ervoor gezorgd dat de samenhangende gegevens op een webpagina staan?***
    - **Hoe wordt bepaald welke gegevens met elkaar samenhangen?**
    - **Hoe worden deze samenhangende gegevens op dezelfde webpagina gezet?**

2.4 Stappenplan voor het realiseren van de generator

In dit onderzoek moet de generator aan de hand van een datamodel webpagina's genereren. Het datamodel en het webpagina zijn beide modellen en dus is er hier sprake van model naar model transformatie. MDE (Model Driven Engineering) is een software ontwikkel techniek waarin het model centraal staat en waarin transformatie van model naar model plaats vindt. Deze techniek zal in dit onderzoek gebruikt worden om de generator te ontwerpen.

In Figuur 2.5 is een abstracte weergave van de relaties tussen de browser, generator, eMAXX Mid Office, back office systemen en datamodellen weergegeven.

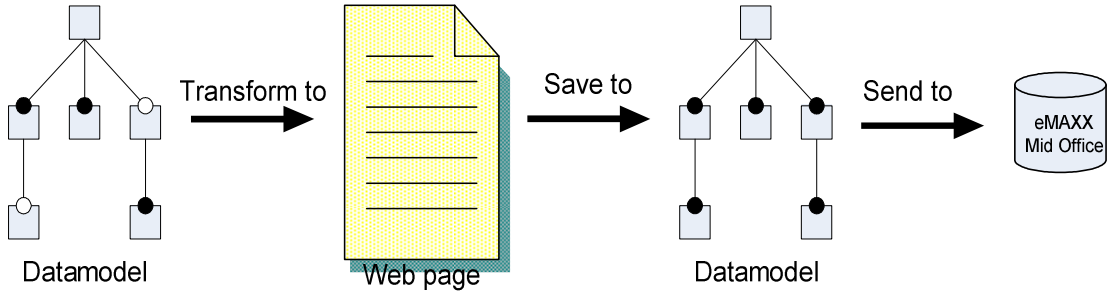


Figuur 2.5: abstracte weergave van de koppelingen tussen de browser, generator, eMAXX Mid Office, back office systemen en datamodellen

Er zullen drie verschillende generatoren ontworpen worden met oplopende complexiteit. Dit wordt gedaan om de problemen in delen op te lossen.

1. Generator zonder interactie

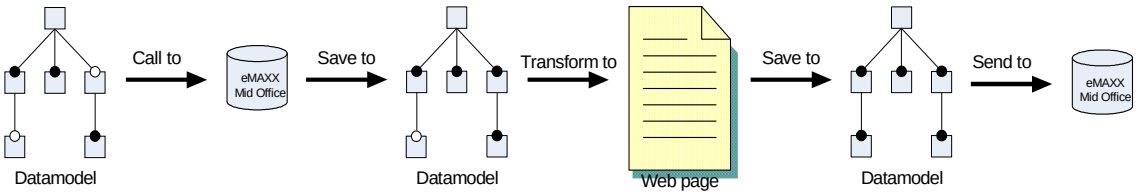
Allereerst zal een generator zonder interactie worden gebouwd die een datamodel transformeert naar één webpagina (zie Figuur 2.6). Met de communicatielijn *1* in Figuur 2.5 wordt het gekozen datamodel opgehaald en wordt deze vervolgens getransformeerd naar een webpagina (zie de communicatielijn *2* in Figuur 2.5). Deze webpagina wordt vervolgens opgestuurd naar de browser van de burger. De ingevulde gegevens op deze webpagina worden naar de generator opgestuurd. De generator slaat deze gegevens op in het (opgehaalde) datamodel. Ten slotte zal de generator deze opgeslagen gegevens opsturen naar de eMAXX Mid Office die verder voor het afhandelen van de aanvraag gaat zorgen. Deze generator zal in hoofdstuk 4 uitvoerig worden besproken.



Figuur 2.6: illustratie van globale acties van generator zonder interactie

2. Generator met ondersteuning van eMAXX Mid Office

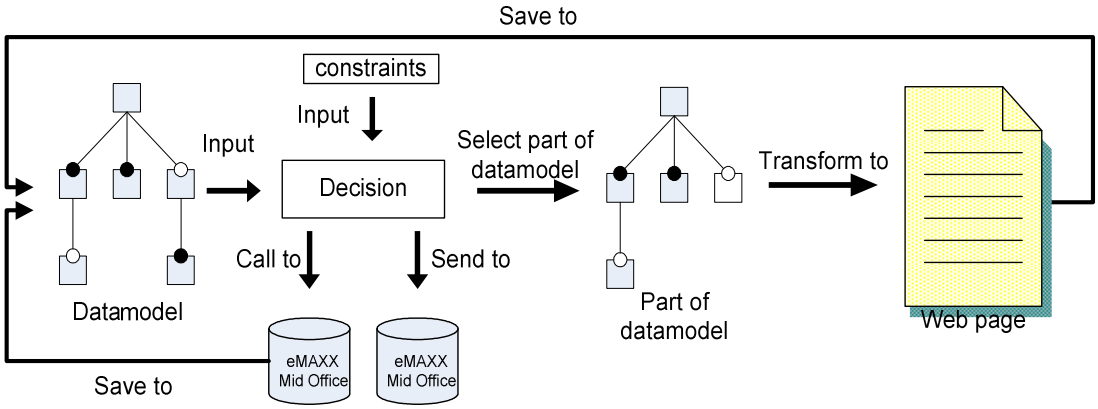
Deze generator is een uitbreiding van de vorige generator met ondersteuning van de eMAXX Mid Office functies (zie Figuur 2.7). Met deze functies kunnen o.a. de opgeslagen gegevens worden opgehaald uit back office systemen. De opgehaalde gegevens zullen in het datamodel worden opgeslagen. De ontbrekende gegevens in het datamodel worden dan getransformeerd naar één webpagina zodat de burger deze kan invullen. De ingevulde waarden worden in het datamodel opgeslagen. Ten slotte zal de generator de gegevens die opgeslagen zijn in het datamodel opsturen naar de eMAXX Mid Office die de aanvraag verder gaat afhandelen. Deze generator zal in hoofdstuk 5 uitvoerig worden besproken.



Figuur 2.7: illustratie van globale acties van generator met ondersteuning van eMAXX Mid Office functie

3. Generator met workflow

Deze generator is een uitbreiding van de vorige generator met workflow. Deze keer wordt een algoritme ingebouwd die aan de hand van een datamodel en de bijbehorende constraints keuze maakt (zie Figuur 2.8). Deze constraints geven extra informatie over het datamodel zoals welke functies van de eMAXX Mid Office aangeroepen kunnen worden voor dit datamodel. Dit algoritme kan kiezen om een webpagina te genereren of om gebruik te maken van eMAXX Mid Office functies. Na elke uitgevoerde keuze worden de ontvangen gegevens opgeslagen in het datamodel en begint het proces opnieuw totdat alle gegevens in het datamodel bekend zijn. Ten slotte zal de generator de gegevens die opgeslagen zijn in het datamodel opsturen naar de eMAXX Mid Office die de aanvraag verder gaat afhandelen. Deze generator zal in hoofdstuk 6 uitvoerig worden besproken.



Figuur 2.8: illustratie van globale acties van generator met workflow

3 Model gedreven software ontwikkeling en variatie modellering

In dit hoofdstuk zullen de technieken worden besproken die in dit onderzoek zijn gebruikt voor het ontwerpen van de generator en het modelleren van het datamodel. In principe zijn er twee technieken waar dit onderzoek op berust, namelijk: MDE en feature modellering.

De generator, die in dit onderzoek wordt ontworpen, moet aan de hand van datamodellen webpagina's genereren. Het doel hiervan is dat er tijd wordt bespaard bij het realiseren en onderhouden van e-formulieren. MDE belooft dat de inspanning voor het ontwikkelen en onderhouden van systemen gereduceerd kan worden door te werken met modellen in plaats van code [Deu07]. Doordat de inspanningen gereduceerd kunnen worden, is het de verwachting dat het ook minder tijd kost om deze systemen te ontwikkelen en te onderhouden. Dit is precies wat dit onderzoek wil bereiken en daarom is voor deze ontwikkeltechniek gekozen.

Het invullen van een e-formulier voor een aanvraag kan van burger tot burger verschillen. Het op te stellen datamodel moet hiermee rekening kunnen houden. Feature modeling kan gebruikt worden om de gemeenschappelijke en variabele delen van het datamodel weer te geven. De feature modeling techniek voldoet verder aan de eisen die in sectie 1.3 zijn opgesteld. In overleg met eMAXX is er voor deze modelleringtechniek gekozen.

In sectie 3.1 zal de model gedreven ontwikkeltechniek worden besproken. Aan de hand van deze techniek wordt de generator ontwikkeld. In sectie 3.2 zal de feature modeling techniek worden besproken.

3.1 Model gedreven ontwikkelingtechniek (MDE)

In deze sectie zal de model gedreven ontwikkeltechniek worden besproken. Allereerst zal in sectie 3.1.1 een korte inleiding gegeven worden wat MDE (Model Driven Engineering) is. In sectie 3.1.2 zal een korte uitleg gegeven worden wat een model en een metamodel is, deze spelen een grote rol bij de transformaties. Ten slotte zal in sectie 3.1.3 transformatie besproken worden, de belangrijkste operatie van MDE.

3.1.1 MDE in het algemeen

MDE is een model gedreven ontwikkeltechniek waarbij het model centraal staat. Deze ontwikkeltechniek probeert een framework neer te zetten om (1) duidelijke methodologie te definiëren, (2) systeem te ontwikkelen van elke abstractie niveau en om (3) tests en validatie's te organiseren en te automatiseren [Fon04]. Het heeft de potentie om de ontwikkelproductiviteit en de kwaliteit te verhogen door de belangrijke aspecten van de oplossing te beschrijven met mensenvriendelijke abstractie en door genereren van gezamenlijke applicatiefragmenten met templates [Sen03].

MDE moet niet verward worden met MDA (Model Driven Architecture). MDA is een OMG (Object Management Group) standaard die een transformatie uitvoert van PIM (Platform Independant Model) naar PSM (Platform Specific Model). OMG is een open ledenconsortium zonder winstbejag. Het consortium produceert en onderhoudt computerspecificaties voor interoperabele toepassingen ten behoeve van ondernemingen [Wik07k]. MDE heeft een bredere scope dan MDA, waarbij MDA een onderdeel van MDE is. In MDE kan een transformatie uitgevoerd worden van een willekeurig model naar een ander model terwijl MDA zich beperkt tot de transformatie van PIM naar PSM.

3.1.2 Model en Metamodel

Er is nog geen standaard definitie voor *model*. Wel zijn er een aantal pogingen gedaan om deze term te definiëren. De meest gebruikte hiervan zijn:

- Een *model* van een systeem is een beschrijving of een specificatie van dat systeem en zijn omgeving voor een zeker doel. Een model is meestal te representeren als combinatie van tekeningen en tekst. De tekst kan een modellering taal of een natuurlijke taal zijn.[Mda03]

- Een *model* is een simplificatie van een systeem dat gemaakt is met een bepaald doel in gedachten. Het model moet antwoorden kunnen geven in plaats van het echte systeem. [Bez01]

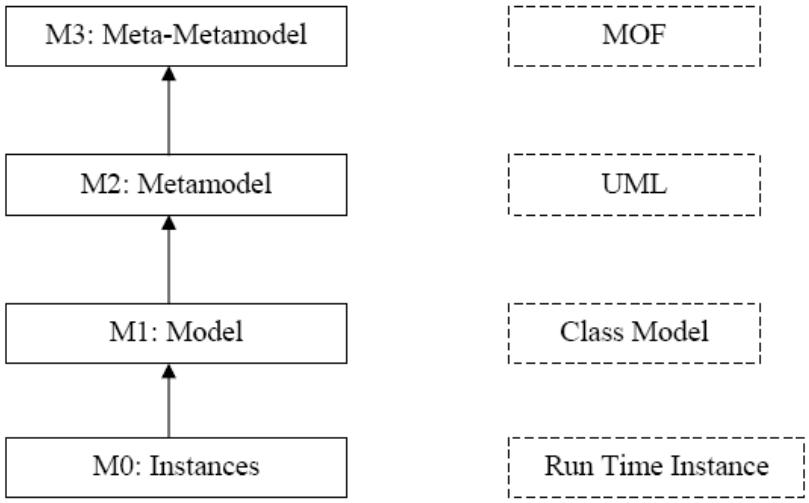
- Een *model* is een beschrijving van een (deel) systeem geschreven in een goed gedefinieerde taal. Een goed gedefinieerde taal met goed gedefinieerde vorm (syntactisch), en betekenis (semantiek), wat toegankelijk is voor automatische interpretatie door een computer. [Kle03]

Waar het globaal op neerkomt, is dat *model* een abstracte representatie van iets is met een bepaald doel.

In de meest brede zin is een *metamodel* een *model* van een modelleringstaal. De term “meta“ betekent overtreffend of hierboven, het feit benadrukkend dat een *metamodel* een modelleringstaal beschrijft op een hoger niveau van abstractie dan de modelleringstaal zelf. Om te begrijpen wat een *metamodel* is, is het nuttig om het verschil te begrijpen tussen een metamodel en een model. Terwijl een *metamodel* ook een *model* is, heeft een *metamodel* twee belangrijke onderscheidende kenmerken:

- Een *metamodel* moet de essentiële eigenschappen en de eigenschappen van de taal, die wordt gemodelleerd, kunnen omvatten. Aldus, zou een metamodel de concrete syntaxis, abstracte syntaxis en de semantieken van een taal moeten kunnen beschrijven.

- Een *metamodel* moet deel uitmaken van een metamodelarchitectuur zoals de architectuurlaag van MOF (Meta-Object Facility) van de OMG zoals in Figuur 3.1 (zie volgende pagina) is weergegeven. MOF is gebouwd als vier-laags architectuur. Het meta-metamodel op de bovenste laag, genaamd de M3 laag, is de taal die door MOF wordt gebruikt om metamodellen (genaamd M2-models) op te stellen [Wik07i]. Metamodellen kunnen gebruikt worden voor het beschrijven van een geldig model of programma’s die toegestaan zijn door de taal, die zichzelf laat beschrijven door een ander metamodel. Dit laat toe om alle metamodellen te beschrijven door één metamodel. Dit éne metamodel, die bekend staat als meta-metamodel, is de sleutel van metamodellering waardoor het toelaat om alle modelleringstalen op een uniforme manier te beschrijven [Cla04].



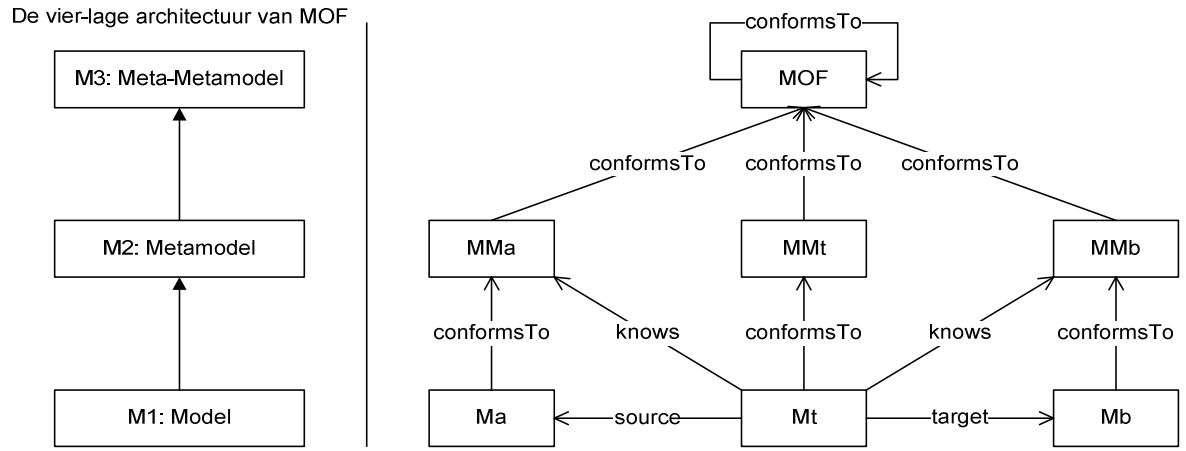
Figuur 3.1 De vier-lage architectuur van MOF in OMG [Wan05]

3.1.3 Transformatie

De *transformatie* is de meest belangrijkste operatie in model engineering [Béz06]. Er zijn twee soorten *transformaties* namelijk *horizontale* en *verticale*. *Horizontale transformatie* transformeert een model naar een ander model van hetzelfde abstractie niveau. *Verticale transformatie* transformeert een abstract model naar een concreter model [Ros06].

In Figuur 3.2 is het algemene plaatje van een metamodel gebaseerde transformatie weergegeven. Een transformatie operatie Mt neemt een model Ma als invoer model en produceert een model Mb. De modellen Ma, Mb en Mt moeten voldoen aan de metamodellen MMa, MMb en MMt. Meestal heeft de transformatie Mt kennis over metamodellen MMa en MMb. Verder moeten de metamodellen MMa, MMb en MMt voldoen aan een meta-metamodel. In dit geval is het meta-metamodel het OMG's MOF wat aan zichzelf moet voldoen.

De metamodel gebaseerde transformatie maakt gebruik van de vier-laags architectuur van MOF. In hetzelfde figuur is links de vier-laags architectuur van MOF weergegeven (die eerder in Figuur 3.1 is weergegeven). Daarin is te zien dat het meta-metamodel MOF van de laag M3 is. Verder is te zien dat metamodellen MMa, MMt en MMb van de M2 laag zijn. De modellen Ma, Mt en Mb zijn van de M1 laag.



Figuur 3.2: metamodel gebaseerde transformatie van een model Ma naar een model Mb met behulp van transformatie Mt [Béz06]

3.2 Feature Modeling

In deze sectie zal de Feature Modeling techniek worden besproken. In sectie 3.2.1 zal de Cardinality-Based Feature Modeling modelleringtechniek worden besproken. In sectie 3.2.2 zullen de programma's die Feature Modeling techniek ondersteunen worden besproken.

3.2.1 Cardinality-Based Feature Modeling

Feature Modeling is een domein modeling techniek. Deze techniek heeft veel interesse opgewekt bij de SPL (Software Product Line) gemeenschap [Cza06]. Feature Modeling kan gebruikt worden om de gemeenschappelijke en variabele systeemdelen van een productfamilie weer te geven. In dit onderzoek bevatten datamodellen verschillende variaties.

Zoals eerder in sectie 3.1.2 is aangegeven, is een metamodel een model van een modelleringstaal. Feature Modeling is zo'n model die een modelleringstaal beschrijft. De Feature Modeling zit daarom in de M2 laag van de vier-laags architectuur van MOF (die in Figuur 3.1 is weergegeven). De modellen die opgesteld worden door deze Feature Modeling techniek zitten op de M1 laag van de vier-laags architectuur van MOF.

De generator gaat webpagina's genereren vanuit datamodellen. Deze datamodellen zijn de feature modellen die met behulp van de feature modelingtechniek worden opgesteld. Voor het genereren van webpagina's wordt er gebruik gemaakt van transformaties (gebruikmakend van de metamodel gebaseerde transformatie overzicht die in Figuur 3.2 is weergegeven). Het opgestelde feature model is dus het model Ma en Feature Modeling is het metamodel MMa die in Figuur 3.2 zijn weergegeven. De gegenereerde webpagina is het model Mb en de syntax waaraan een webpagina moet voldoen is het metamodel MMb.

In deze sectie wordt Cardinality-Based Feature Modeling (CBFM) [Cza04] besproken. Deze modellering is een uitbreiding op de oorspronkelijke Feature Oriented Domain Analysis (FODA) [Kan90]. Deze is vooral gekozen omdat met CBFM relaties beter in kaart gebracht kunnen worden. Bijvoorbeeld een [0..*] relatie van een feature met een andere feature kan niet worden weergegeven in FODA, maar wel in CBFM.

Een feature model bestaat uit feature diagrammen en extra informatie zoals feature beschrijving, binding tijd, prioriteit enz. [Cza04]. Een feature is een systeemeigenschap die enige relevantie heeft voor sommige gebruikers. Features worden gebruikt om de overeenkomsten of de verschillen tussen de systemen weer te geven. De features zijn georganiseerd in feature diagrammen. Een feature diagram is een boom met als root de naam van het feature diagram. De onderliggende knopen zijn features. In Figuur 3.3 (zie volgende pagina) zijn de symbolen die gebruikt worden in deze boomstructuur weergegeven.

Symbol	Explanation
	Solitary feature with cardinality [1..1], i.e., *mandatory* feature
	Solitary feature with cardinality [0..1], i.e., *optional* feature
	Solitary feature with cardinality $[0..m]$, $m > 1$, i.e., *optional clonable* feature
	Solitary feature with cardinality $[n..m]$, $n > 0 \wedge m > 1$, i.e., *mandatory clonable* feature
	Grouped feature with cardinality [0..1]
	Grouped feature with cardinality [1..1]
	Feature F with attribute of type T and value of *value*
	Feature model reference F
	Feature group with cardinality $\langle 1-1 \rangle$, i.e. *xor*-group
	Feature group with cardinality $\langle 1-k \rangle$, where k is the group size, i.e. *or*-group
	Feature group with cardinality $\langle i-j \rangle$

Figuur 3.3: symbolen van Cardinality-Based Feature modeling [Cza04]

Features (*Solitary Feature*) kunnen geannoteerd worden met cardinaliteiten zoals [1..*] of [3..3]. Cardinaliteit [m..n] geeft aan dat een feature minimaal m en maximaal n keer gekloond mag worden. Wanneer er een cardinaliteit [m..*] staat, dan geeft dat aan dat de feature minimaal m en maximaal ongelimiteerd keer gekloond mag worden. Verplichte en optionele features kunnen gezien worden als speciale gevallen met cardinaliteiten [1..1] en [0..1]. Een feature bevat de attributen '*type*' en '*value*'. '*Type*' geeft het type en '*value*' geeft de waarde van de feature weer.

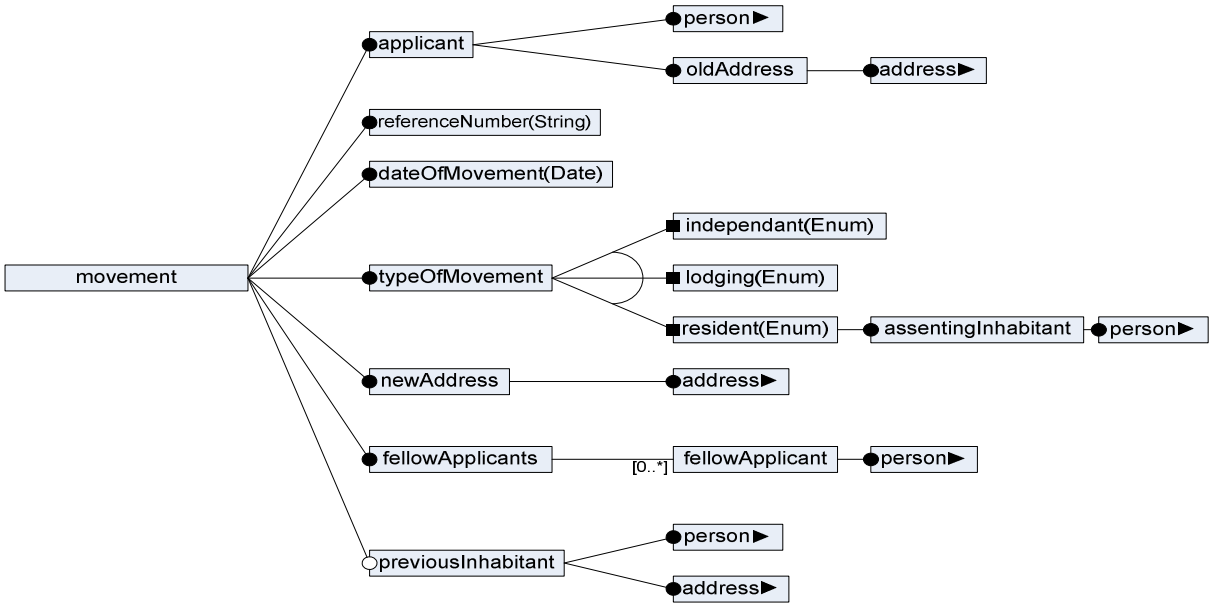
Een feature groep (*Feature Group*) geeft een keuze over gegroepeerde features (*Grouped Feature*) weer. De keuze wordt beperkt door de groep cardinaliteit <m-n>, die aangeeft dat minimaal m en maximaal n gegroepeerde features gekozen mogen worden. Hierbij geldt wel de beperking $0 \leq m \leq n \leq k$, waarbij $k \geq 0$ en k het aantal gegroepeerde features is binnen de feature groep. Gegroepeerde features hebben zelf verder geen cardinaliteit, omdat deze niet meer als *Solitary Feature* worden gezien. Dit voorkomt redundanties.

Een referentie feature (*Feature model reference*) refereert naar een root feature (het begin van het feature diagram) van een ander feature model. Dit is erg handig als er features zijn in het feature model die overeenkomen met een feature model dat eerder is gespecificeerd. Door referentie features kunnen eerder opgestelde feature modellen worden hergebruikt. Ook zorgen de referentie features voor een beter overzicht van het feature model door veel voorkomende takken als referentie features te modelleren.

In Figuur 3.4 (zie volgende pagina) is een voorbeeld van een feature diagram van het product: *doorgeven van een verhuizing* weergegeven. Daarin is de RootFeature *movement* weergegeven. In dit feature diagram is er één "*optional*" feature namelijk *previousInhabitant*. De cardinaliteit van deze feature is [0..1]. Deze feature kan maar één keer voorkomen, maar kan ook achterwege gelaten worden. Daarentegen komt de feature *fellowApplicant* voor met cardinaliteit [0..*] wat aangeeft dat de feature achterwege gelaten kan worden maar ook dat de feature oneindig vaak gekloond kan worden.

Verder zijn er twee referentie features namelijk *person* en *address*. De takken *person* en *address* komen meerdere keren voor in het feature diagram waardoor ze als referentie features worden gebruikt. De feature diagrammen *person* en *address* zijn te vinden in bijlage A. Ook is er één feature groep met drie gegroepeerde features in het feature diagram. De cardinaliteit van de feature groep is [1..1]. Hier mag er dus maar één feature gekozen worden van de drie gegroepeerde features *independent*, *lodging* en *resident*.

De resterende features zijn Solitary Features met als cardinaliteit [1..1]. Dat wil zeggen dat deze features exact één keer moeten voorkomen.



Figuur 3.4: voorbeeld feature diagram van het product: doorgeven van verhuizing

De in dit onderzoek opgestelde feature modellen bevatten alleen feature diagrammen. Alle wijzigingen of operaties die uitgevoerd worden op een feature diagram zijn in wezen dus ook wijzigingen of operaties op het feature model. Om verwarring te voorkomen wordt in dit verslag in het vervolg de term feature model gebruikt voor het feature diagram en het feature model.

3.2.2 Programma's die feature modeling ondersteunen

Er zijn talloze programma's om de feature modellen mee op te stellen. In dit onderzoek is er naar drie programma's gekeken, namelijk: *Pure Variant*, *FeaturePlugin* en *XFeature*. Niet alle programma's zijn getest, dus het kan zijn dat er een ander Feature Modeling programma is die simpeler is en meer functionaliteit bevat dan de geteste programma's. De nadruk van dit onderzoek ligt echter niet op het beste programma om feature modellen te maken maar om van feature modellen webpagina's te genereren.

- ***Pure Variant***

  Pure Variant [Pur06] is een commercieel programma. Hiermee kunnen snel feature modellen worden gemaakt en er kunnen ook constraints worden opgegeven. In dit programma kan helaas de cardinaliteit [0..*] niet worden weergegeven. In het feature model verhuizing wordt deze cardinaliteit wel gebruikt voor *fellowApplicants* feature. Dit programma valt dus af.

- ***FeaturePlugin***

  FeaturePlugin [Fea04] is een plugin in *Eclipse*. Met het *FeaturePlugin* programma is het erg simpel een feature model te maken. Het nadeel van dit programma is dat wanneer er een XML export wordt gedaan niet alle waarden worden weggeschreven, zoals "*annotation*" en "*description"*. Deze waarden worden gebruikt bij de transformatie van een feature model naar een webpagina. In de broncode van de opgestelde feature model staat deze waarden wel, maar deze bron bevat te veel onnodige informatie waardoor het veel ruimte in beslag neemt.

- ***XFeature***

  Een ander programma is *XFeature* [Xfe05] welke ook een plugin in *Eclipse* is. Dit is ook een erg simpel programma die de "*cardinaliteiten"*, de "*annotations"* en de "*descriptions"* kan weergeven. Verder is er duidelijk verschil in features zoals SolitaryFeature, GroupFeature en ReferenceFeature. Het feature model wordt opgeslagen in een XML document. In Code 3.1 is een deel van de XML document weergegeven. Het hele document waarin het feature model van het product doorgeven verhuizing is te vinden in bijlage B. Dit XML document bevat geen overbodige informatie en is goed te gebruiken voor transformaties, waardoor er voor dit programma gekozen is om de feature modellen op te stellen.

```
1.    <fm:SolitaryFeature fm:value="applicant">
2.       <fm:Annotation fm:value="Annotation">
3.          <fm:Description fm:value="Aanvrager"/>
4.       </fm:Annotation>
5.       <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
6.       <fm:SolitaryFeature fm:value="oldAddress">
7.          <fm:Annotation fm:value="Annotation">
8.             <fm:Description fm:value="Huidige adres"/>
9.          </fm:Annotation>
10.         <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
11.         <fm:SolitaryReference fm:value="refAddress">
12.            <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
13.         </fm:SolitaryReference>
14.      </fm:SolitaryFeature>
15.      <fm:SolitaryReference fm:value="refPerson">
16.         <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
17.      </fm:SolitaryReference>
18.   </fm:SolitaryFeature>
```

Code 3.1: weergave van SolitaryFeature *applicant* in XML formaat

4 Model gedreven e-formulier generator zonder interactie

In dit hoofdstuk zal de generator zonder interactie worden ontworpen waarbij de problemen, die een rol spelen bij het realiseren en onderhouden van webpagina's van e-formulieren, deels worden opgelost. Deze problemen zijn eerder in sectie 2.1.1 weergegeven.

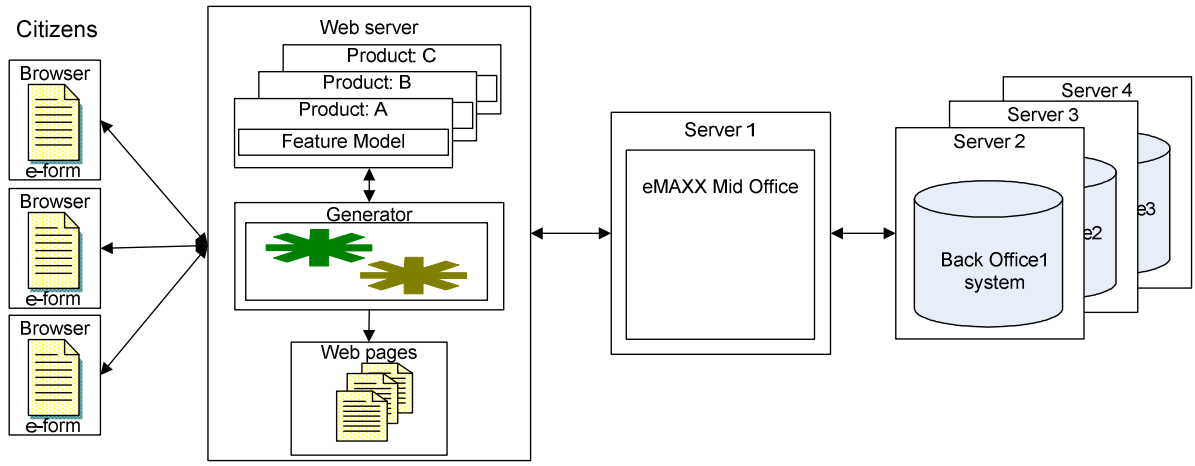
In sectie 4.1 wordt het generatie proces besproken. In de daarop volgende sectie´s worden de processtappen voor het indienen van een aanvraag met behulp van de model gedreven e-formulier generator zonder interactie uitvoerig beschreven. In sectie 4.2 wordt de processtap: *select product* besproken. In sectie 4.3 wordt de processtap: *get feature model of selected product* besproken. In sectie 4.4 wordt de processtap: *generate web page of selected feature model* besproken. In sectie 4.5 wordt de toestand: *generated web page of selected feature model* besproken. In sectie 4.6 wordt de processtap: *specialize selected feature model by means of input values* besproken. In sectie 4.7 wordt de processtap: *generate report of selected product* besproken. Als laatst wordt in sectie 4.8 de processtap: *send report of selected product to eMAXX Mid Office* besproken.

4.1 Overzicht generatie proces

In deze sectie wordt het generatie proces met de model gedreven e-formulier generator zonder interactie weergegeven. Door gebruik te maken van datamodellen in combinatie met deze generator zullen een deel van de problemen die in sectie 2.1.1 staan deels worden opgelost. Zoals eerder aangegeven zal de feature modeling techniek worden gebruikt voor het opstellen van datamodellen. In dit hoofdstuk wordt een generator zonder interactie ontworpen die aan de hand van een feature model één webpagina genereert.

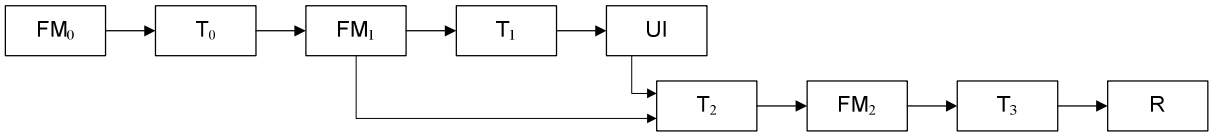
Wanneer er een aanpassing in de applicatie moet worden aangebracht, dan wordt in de nieuwe ontwikkeltechniek alleen het feature model gewijzigd. Dit zou minder tijd moeten kosten dan de aanpassingen handmatig in elke laag (die in sectie 2.1.1 zijn aangegeven) door te voeren. Ook zal de complexiteit van de aanpassingen afnemen doordat alleen het feature model wordt aangepast en er verder geen rekening gehouden hoeft te worden met verschillende lagen zoals bij het handmatig implementeren van e-formulieren wel het geval is.

De generator die ontworpen wordt, moet bruikbaar zijn voor elke gemeente. De feature modellen en de eMAXX Mid Office zullen per gemeente verschillend zijn. In Figuur 4.1 (zie volgende pagina) is een abstracte weergave van de communicatie tussen de verschillende onderdelen van de generator voor een willekeurige gemeente weergegeven.



Figuur 4.1: abstracte weergave van de communicatie van verschillende onderdelen met de generator van een gemeente

Zoals eerder aangegeven zal MDE gebruikt worden om de generator te ontwikkelen. De generator in dit hoofdstuk voert vier transformaties uit die in Figuur 4.2 zijn weergegeven. Voor de transformaties wordt het transformatie overzicht gebruikt die in sectie 3.1.3 is weergegeven.



Figuur 4.2: transformaties van de model gedreven e-formulier generator zonder interactie

FM_0	=	basis feature model
T_0	=	transformatie voor het importeren van referentie feature modellen
FM_1	=	feature model met geïmporteerde referenties
T_1	=	transformatie voor het genereren van de UI
UI	=	User Interface
T_2	=	transformatie voor het specialiseren van het feature model
FM_2	=	feature model die gespecialiseerd is
T_3	=	transformatie voor het genereren van een rapport
R	=	rapport dat naar de eMAXX Mid Office wordt verzonden

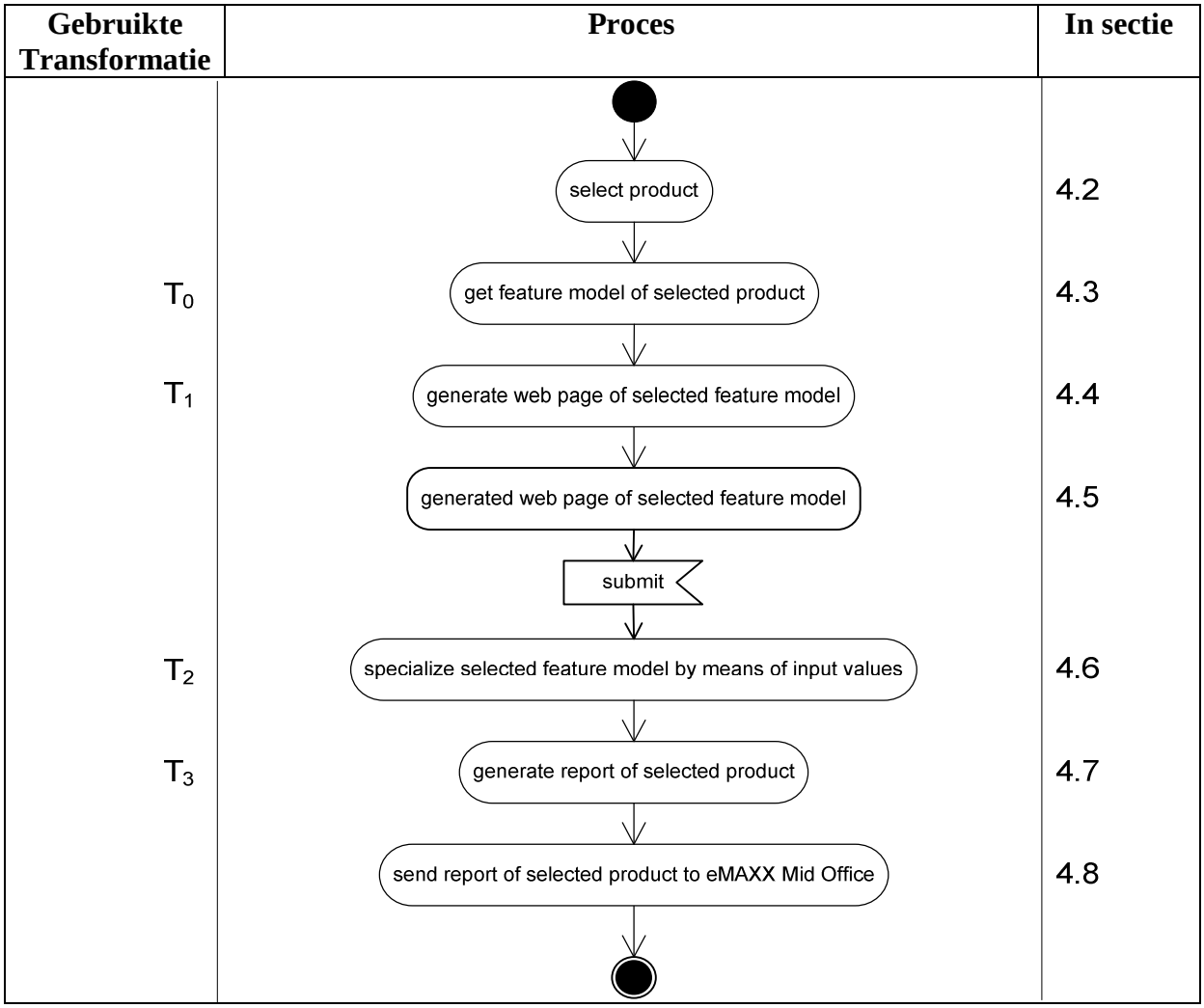
De eerste transformatie T_0 is een transformatie van een basis feature model FM_0 naar het feature model FM_1 waarbij de referentie feature modellen in het geselecteerde feature model zijn geïmporteerd.

De tweede transformatie T_1 is een transformatie van het feature model FM_1 naar UI (User Interface) waarbij een webpagina wordt gegenereerd. FM_1 is het geselecteerde feature model waarin de referentie feature modellen zijn geïmporteerd en waarin nog geen gegevens staan. De UI is het model van de webpagina.

De derde transformatie T_2 transformeert het feature model FM_1 naar het feature model FM_2 met behulp van de waarden die ingevuld zijn op de webpagina. Wanneer de features van het feature model een waarde krijgen of de keuzes beperkter of geselecteerd worden heet dit *specialiseren* van het feature model. FM_2 is hetzelfde feature model als feature model FM_1 maar dit keer is het feature model gespecialiseerd. De UI is niet meteen een invoer voor de transformatie T_2 maar wordt wel gebruikt bij het uitvoeren van de transformatieregels. Dit zal nader uitgelegd worden in sectie 4.6.

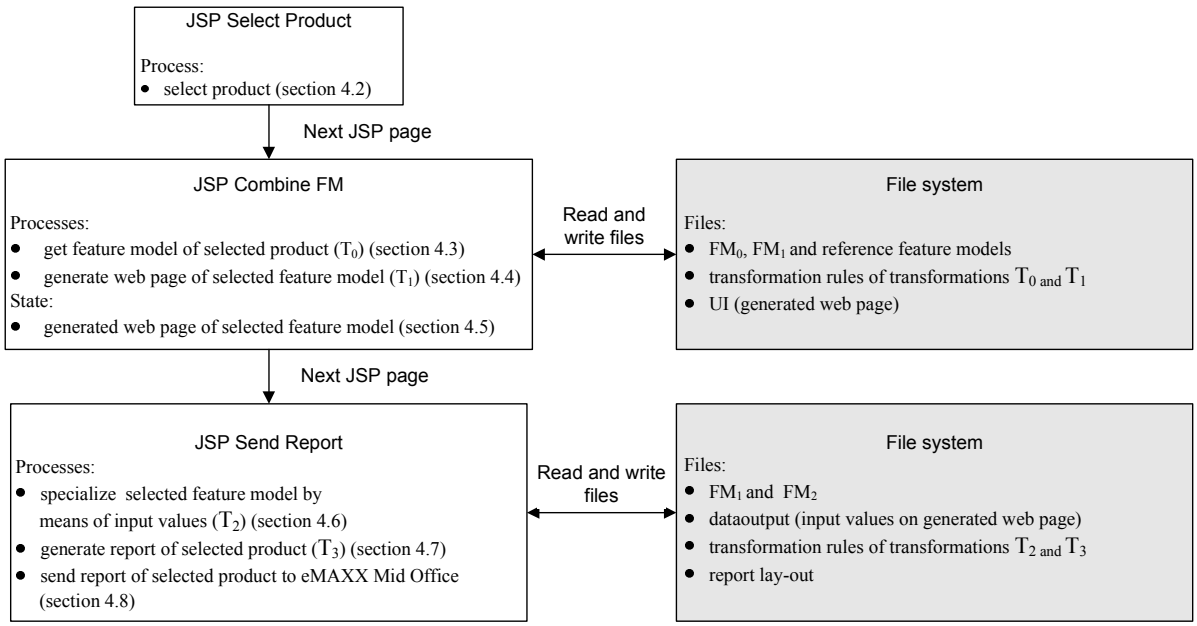
De vierde en het laatste transformatie T_3 transformeert het feature model FM_2 naar het rapport R. Dit rapport is het XML bestand dat naar de eMAXX Mid Office wordt verzonden. De eMAXX Mid Office, die de aanvraaggegevens gaat gebruiken voor het verder afhandelen van de aanvraag, heeft een andere interface dan het feature model waardoor deze transformatie noodzakelijk is.

Zoals in sectie 2.4 is beschreven, zal in dit hoofdstuk een e-formulier generator met interactie worden ontworpen. In Figuur 4.3 worden de processtappen voor het indienen van een aanvraag met behulp van deze generator in een UML activiteitdiagram (een OMG specificatie [OMG06]) weergegeven.



Figuur 4.3: UML activiteitdiagram van de processtappen voor het invullen van een e-formulier met behulp van de model gedreven e-formulier generator zonder interactie

De model gedreven e-formulier generator zonder interactie is opgebouwd uit drie JSP pagina's (zie Figuur 4.4). In de eerste JSP pagina (*JSP Select Product*) wordt de processtap: *select product* uitgevoerd. In de tweede JSP pagina (*JSP Combine FM*) worden de processtappen: *get feature model of selected product* en *generate web page of selected feature model* uitgevoerd. Deze JSP pagina haalt de benodigde bestanden van de webserver op. De resultaten van de transformaties worden weggeschreven naar dezelfde webserver. De gegenereerde web pagina wordt in deze JSP pagina ingeladen waardoor het aanvraagproces in de toestand: *generated web page of selected feature model* terecht komt. In de laatste JSP pagina (*JSP Send Report*) worden de processtappen: *specialize selected feature model by means of input values, generate report of selected product* en *send report of selected product to eMAXX Mid Office* uitgevoerd. Deze JSP haalt ook de benodigde bestanden van de webserver op. De resultaten worden ook hier naar dezelfde webserver weggeschreven.



Figuur 4.4: overzicht JSP pagina's van de model gedreven e-formulier generator zonder interactie

Het proces (indienen van een aanvraag) start met het selecteren van een product door een burger. Dit wordt gedaan door het drukken op een product link op de webpagina. Dit is nodig zodat de generator weet om welk product en dus om welke feature model het gaat.

Nadat dit bekend is, wordt het juiste feature model van de webserver gehaald. Dit feature model wordt gedurende het hele proces gebruikt om de gegevens die door de burger worden ingevuld in dit feature model op te slaan. Ook wordt het feature model gebruikt om te bepalen wat er op een webpagina moet komen. Vervolgens worden de features van het feature model waarvan de waarden ontbreken, getransformeerd naar een webpagina om de ontbrekende waarden door de burger in te laten vullen. De gegenereerde webpagina wordt naar de browser van de desbetreffende burger gestuurd.

Nadat de burger de ontbrekende waarden heeft ingevuld en op de verzendknop heeft gedrukt, worden deze weggeschreven naar het feature model. Dit model wordt vervolgens getransformeerd naar het rapport. Vervolgens wordt dit rapport verzonden naar de eMAXX Mid Office die voor de verdere afhandeling van de aanvraag zorgt. Dit is tevens het einde van het proces.

4.2 processtap: select product

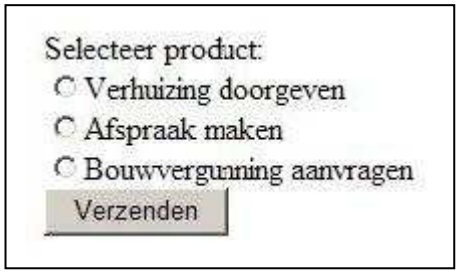
Elke gemeenten heeft een eigen digitaal loket waar ze hun producten in aanbieden. Een digitaal loket is gewoon een webpagina waar de burgers naartoe kunnen surfen om de producten van de gemeente af te nemen. Hieronder in Figuur 4.5 is het digitale loket van de gemeente Enschede als voorbeeld weergegeven.

Figuur 4.5: digitaal loket van de gemeente Enschede[Dle07]

In dit onderzoek zijn feature modellen opgesteld voor de volgende producten:

- ***Afspraak maken***
- ***Verhuizing doorgeven***
- ***Bouwvergunning aanvragen***

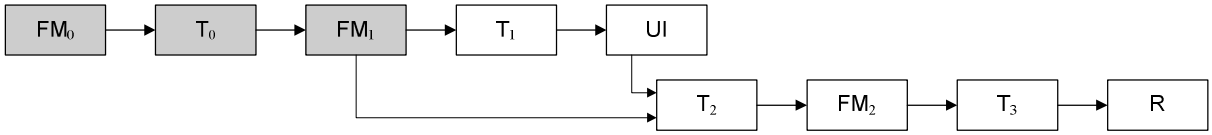
Deze producten zijn gekozen, omdat ze in complexiteit van elkaar verschillen. Zo is het product "*afspraak maken*" het meest simpele en het product "*bouwvergunning aanvragen*" het meest complexe. Het digitale loket, dat in dit onderzoek wordt gebruikt, is handmatig geïmplementeerd. Op dit loket selecteert de burger een product en drukt vervolgens op de verzendknop (zie Figuur 4.6). Hierdoor weet de generator welke producteigenschappen opgehaald moeten worden. In dit document zal het product "*verhuizing doorgeven*" als voorbeeld gebruikt worden. Nadat het product is gekozen, gaat het proces verder met de processtap: *get feature model of selected product* die in sectie 4.3 wordt besproken.



Figuur 4.6: webpagina om een product te selecteren

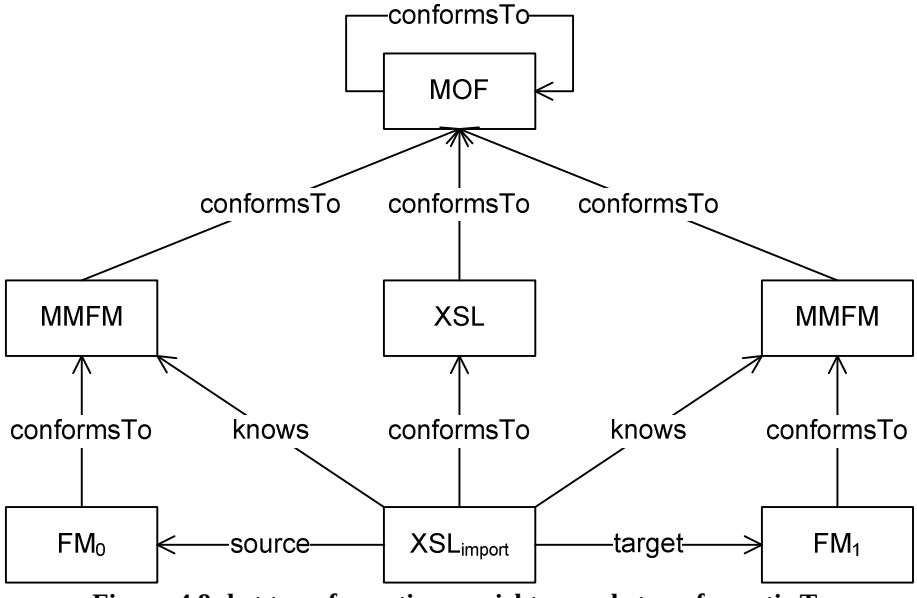
4.3 Processtap: get feature model of selected product

Nadat de burger een product heeft geselecteerd en op verzend heeft gedrukt, wordt de geselecteerde productnaam als parameter verzonden naar de volgende JSP pagina (*JSP Combine FM*). In deze JSP pagina wordt deze parameter gebruikt om het product (het feature model) van de webserver op te halen. De JSP pagina's staan op de webserver van de desbetreffende gemeente. Het ophalen van het feature model gaat met behulp van een link die naar de desbetreffende feature model verwijst die ook op de webserver van de gemeente staat. De parameter is een deel van deze link zodat het juiste feature model wordt opgehaald. Bij het ophalen van het feature model worden de referentie features in het feature model geïmporteerd met behulp van transformatie T_0. Deze transformatie is in Figuur 4.7 terug te vinden waarbij de componenten die hier een rol spelen een donkere achtergrondkleur hebben.



Figuur 4.7: de transformatie T_0 voor het importeren van referentie features in het geselecteerde feature model

Zoals eerder aangegeven wordt het transformatie overzicht, die in sectie 3.1.3 is weergegeven, gebruikt voor de transformaties. In Figuur 4.8 is het transformatie overzicht weergegeven voor transformatie T_0 die een basis feature model FM_0 naar het feature model FM_1 transformeert. Om een transformatie uit te voeren moeten transformatieregels opgesteld worden. De transformatieregels zijn in XSL (Extensible Stylesheet Language) geschreven. XSL is een formele taal waarin beschreven kan worden hoe XML-documenten geformatteerd moeten worden [Wik07o]. De transformatieregels voor deze transformatie T_0 staan in het XSL_{import} bestand.

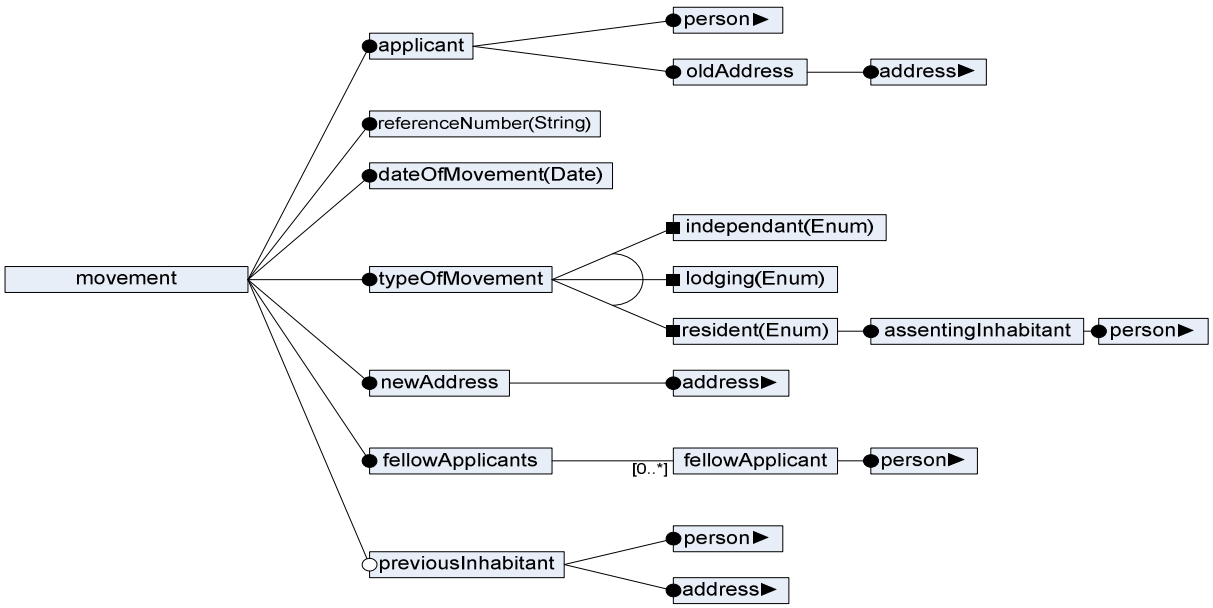


Figuur 4.8: het transformatie overzicht voor de transformatie T_0

Het importeren van referentie features gaat met behulp van XSLT (Extensible Stylesheet Language Transformations). De feature modellen, die opgesteld zijn met behulp van het programma XFeature zijn in een XML document opgeslagen. In dit onderzoek zijn de transformaties van XML naar XML of van XML naar HTML (HyperText Markup Language) taal. HTML is een taal (geen programmeertaal) voor de opmaak van documenten. HTML wordt vooral gebruikt op het Internet, om webpagina's te tonen [Wik07l]. Aangezien XSLT met deze beide talen kan werken, kan dezelfde processor en de taal worden gebruikt voor alle transformaties die in dit onderzoek voorkomen waardoor XSLT gekozen is voor transformaties. De referentie features worden in het feature model geïmporteerd omdat de XSLT processor als invoer één bestand accepteert. De transformatieregels zorgen ervoor dat de feature modellen van de referentie features worden geïmporteerd.

In sectie 4.2 is al aangegeven dat het product: *verhuizing doorgeven* als voorbeeld wordt gebruikt. In Figuur 4.9 is het feature model van dit product weergegeven. Vervolgens gaat het proces verder met de processtap: *generate web page of selected feature model* die in sectie 4.4 wordt besproken.

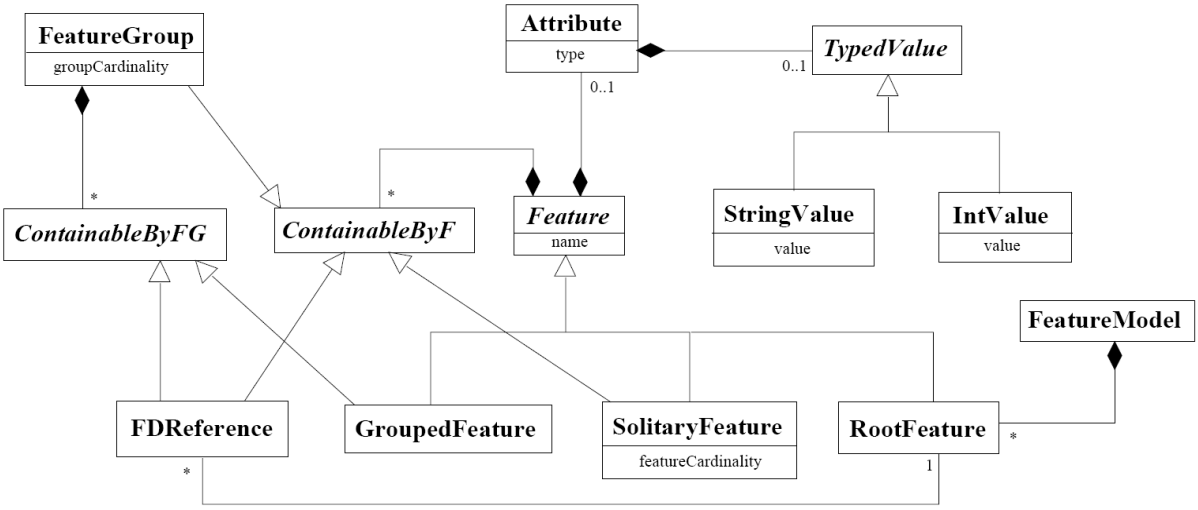
In sectie 4.3.1 wordt het metamodel van het feature model weergegeven. In sectie 4.3.2 worden de scenario's van het product: *verhuizing doorgeven* weergegeven. In sectie 4.3.3 wordt uitgelegd hoe het feature model, aan de hand van het XML schema, is opgesteld. In sectie 4.3.4 wordt de implementatie van de processtap besproken.



Figuur 4.9: het feature model van het product: doorgeven van verhuizing

4.3.1 Metamodel Feature Model

Het gebruik van metamodel is in hoofdstuk 3 uitgelegd. In deze sectie zal nu het UML (Unified Modeling Language) metamodel van cardinality-based feature models beschreven worden die in het artikel van Czarnecki [Cza04] is behandeld. In Figuur 4.10 is het UML metamodel van een cardinality-based feature model weergegeven.



Figuur 4.10: UML metamodel van cardinality-based feature models [Cza04]

De gebruikte entiteiten in dit onderzoek zijn: *FeatureModel, RootFeature, SolitaryFeature, GroupedFeature, FeatureGroup, FDReference, Feature, Attribute, Typedvalue, StringValue* en *Intvalue*. Het *FeatureModel* geeft aan om welke feature model het gaat. De *RootFeature* geeft de feature diagrammen aan. Wanneer er referentie features zijn en deze geïmporteerd worden in het feature model, wordt vanaf de *RootFeature* geïmporteerd. *SolitaryFeature*, *GroupedFeature* en *FeatureGroup* geeft aan om welk soort feature het gaat. *FDReference* refereert naar een feature model. Zo kunnen er stukken van een feature model die vaak voorkomen als referentie worden gebruikt. Bij het onderhoud bespaar je veel tijd door alleen in het gerefereerde feature model wijzigingen aan te brengen en niet bij elke plek waar ernaar verwezen wordt. Aan een *Feature* kan een *Attribute* gekoppeld worden. Hier is *TypedValue* gekoppeld die aangeeft welke voorgedefinieerd type de *Feature* is. In dit metamodel zijn *StringValue* en *IntValue* de voorgedefinieerde typen omdat deze het meest voorkomende typen zijn. Uiteraard kan aan het metamodel meerdere typen toegevoegd worden. Hieronder zullen de entiteiten van het metamodel worden uitgelegd.

Een *FeatureModel* bestaat uit één of meerdere *RootFeatures*. Deze *RootFeatures* zijn de roots van verschillende feature diagrammen in het feature model. Naast *RootFeatures* zijn er nog 2 andere soorten features de *GroupedFeature* en *SolitaryFeature*. De eerste is een feature dat alleen voor kan komen in een *FeatureGroup*.

Een *Feature* kan optioneel een *Attribute* van een bepaald type bevatten. Deze attributen kunnen optioneel ook een bepaalde *TypedValue* hebben. In dit metamodel is voor de eenvoud alleen de *StringValue* en *IntValue* weergegeven.

FDReference staat voor feature model referentie. *FDReference* kan alleen refereren naar één *RootFeature.* Elke *RootFeature* zelf kan door meerdere *FDReferences* worden gerefereerd.

De abstracte klassen *ContainableByFG* en *ContainableByF* staat voor dat soort objecten dat respectievelijk *FeatureGroup* en *Feature* kan bevatten. *FeatureGroup* kan alleen *GroupedFeature* of *FDReference* bevatten, terwijl *Feature* alleen *SolitaryFeatures*, *FeatureGroups* en *FDReferences* kan bevatten.

Een *featureCardinality* is een attribuut van een *SolitaryFeature*. Dit geeft de cardinaliteit van de kinderen van het *SolitaryFeature* weer. Een *featureCardinality* I is een reeks van intervallen van de vorm $[n_1..n'_1]\ldots[\ n_l..n'_l]$ waarin de volgende invariantie worden aangenomen:

$$\forall i \in \{1,...,l-1\} : n_i, n'_i \in \mathrm{N} \qquad n_l \in \mathrm{N} \qquad n'_l \in \mathrm{N} \cup \{*\}$$
$$\forall n \in \mathrm{N} : n < * \qquad 0 \le n_1$$
$$\forall i \in \{1,...,l\} : n_i \le n'_i \qquad \forall i \in \{1,...,l-1\} : n'_i < n_{i+1}$$

Een lege reeks van intervallen is aangegeven met ε.
Een voorbeeld van een geldige specificatie van een *featureCardinality* is [1..2][5..*], wat aangeeft dat een feature 1, 2, 5, of elke getal groter 5 keer gekloond mag worden. Een lege reeks is equivalent met [0..0] en impliceert dat de SolitaryFeature nooit in een configuratie gekozen kan worden.

Een *FeatureGroup* geeft een keuze over *GroupedFeatures* weer. De keuze wordt beperkt door de *groupCardinality* < n - n' > wat aangeeft dat de gebruiker minimaal n en maximaal n' *GroupedFeature* binnen de *FeatureGroup* kan selecteren. Gegeven een k > 0 het aantal *GroupedFeatures* zijn, dan wordt de volgende invariant over *groupCardinality* aangenomen:
$0 \le n \le n' \le k$.

GroupedFeatures hebben verder geen *featureCardinalities* omdat ze geen relatie hebben met *SolitaryFeature*. Hiermee wordt de redundante representatie van groepen voorkomen.

4.3.2 Scenario's van het product: doorgeven van verhuizing

eMAXX maakt gebruik van de eMAXX Mid Office. Deze maakt gebruik van het XML (Extensible Markup Language) formaat. Als men een verhuizing of een bouwvergunning aanvraag doet, worden de opgegeven gegevens in XML formaat opgeslagen en opgestuurd naar de eMAXX Mid Office. Dit opgestuurde XML bericht moet voldoen aan de eisen van de van te voren opgestelde XSD (XML Schema Definition). Het XML bericht is een instantie van XSD. XSD is een aanbeveling van het Consortium van World Wide Web (W3C), die specificeert hoe de elementen in een XML document formeel beschreven moet worden [SEA06]. W3C verstrekt een middel om de structuur, de inhoud en de semantiek van XML-documenten in details te bepalen [W3C06].

Het XML schema is het datamodel van de aanvraaggegevens van de huidige werkwijze. De generator gaat een ander datamodel gebruiken, namelijk het feature model. Dit feature model is geschikter voor de generator doordat er vooraf gedefinieerde elementen zijn zoals SolitaryFeature, *attribute*, description etc die onafhankelijk zijn van de aanvraag. Bij vaste elementnamen kunnen betere transformatieregels opgesteld worden dan bij verschillende elementnamen voor verschillende aanvragen. Vanuit de eerder opgestelde XML schema's worden de feature modellen opgesteld.

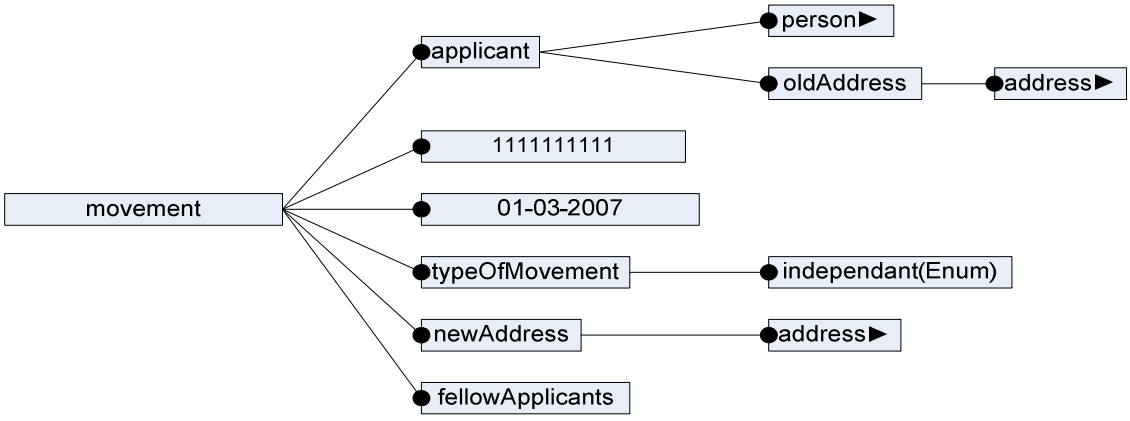
Een feature model geeft een goed beeld over de variatie van gegevens, die van burger tot burger kunnen verschillen. Om enig beeld te krijgen van deze variatie wordt eerst kort ingegaan op aantal mogelijke variaties van de feature modellen.

Het feature model: *persoon* bevat drie keuzes (M, V en F) bij het weergeven van het geslacht. Verder zijn er vijf entiteiten (contactinfo, geboortedatum, geboorteplaats, identiteit, burgerstatus en functie) die door de burger opgegeven of achterwege gelaten kunnen worden. Het aantal variaties van het feature model persoon is dan $3*(2^5) = 96$.
Het feature model: *adres* bevat vijf entiteiten(indicatie, gemeente, stad, district en wijk) die door de burger opgegeven of achterwege gelaten kunnen worden. Het aantal variaties van het feature model *adres* is dan $2^5 = 32$.

Het feature model *verhuizing* bevat twee entiteiten (*persoon* (96 variaties) en *adres* (32 variaties)) die de persoons- en adresgegevens zijn van de verhuizer. Deze entiteiten zijn een referentie naar desbetreffende feature modellen. Verder zijn er drie keuzes (zelfstandig, kamer en inwonen) voor de manier van inwonen. Als bij deze keuze gekozen wordt voor inwonen, dan moeten ook de persoonsgegevens van de eigenaar opgegeven worden. Dit is het feature model *persoon* (96 variaties). Verder is er één entiteit (vorigeInwoner) die door de burger opgegeven of achterwege gelaten kan worden. De entiteiten *persoon* en *adres* van vorigeInwoner zijn afhankelijk van de entiteit vorigeInwoner wel of niet wordt opgegeven door de burger. Wanneer de entiteit vorigeInwoner niet wordt opgegeven kan er ook geen persoon en adres worden opgegeven. De keuze hierbij is dus 1+1*(96*32). Verder is er keuze of er meeverhuizers zijn en hoeveel. Bij elke medeverhuizer moet de persoonsgegevens meegegeven worden. Dit is het feature model *persoon* (96 variaties). Stel dat N personen meeverhuizen, dan is het aantal variaties van deze entiteit 1+(N*96). Het aantal variaties van het feature model *verhuizing* is dan 96*32*(2+(1*96))*(1+1*(96*32))*(1+(N*96)), waarbij N de maximale grens is voor het aantal medeverhuizers. In het gebruikte voorbeeld is de maximale onbegrensd. Het aantal variaties van het feature model *verhuizing* is dan $96*32*(2+(1*96))*(1+1*(96*32))*(1+(\infty*96)) = \infty$.

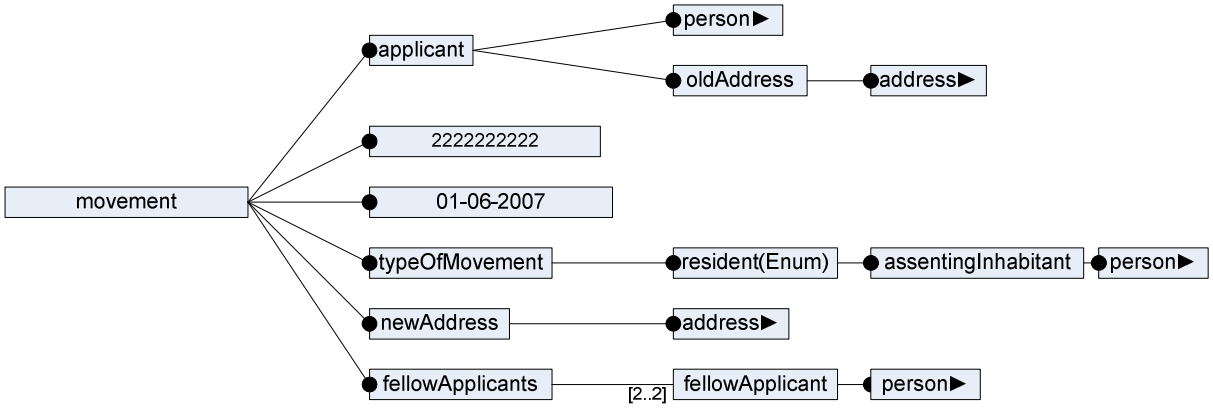
Hieronder is er een aantal scenario's weergegeven van het feature model *verhuizing* die in Figuur 4.9 is weergegeven. Het aantal mogelijke scenario's bij doorgeven van verhuizing is ∞. Dit komt doordat er geen bovengrens is weergegeven bij het aantal meeverhuizers. Als het bovengrens op 4 gezet wordt, dan zal het aantal variaties voor het feature model *verhuizing* 96*32*(2+(1*96))* (1+1*(96*32))*(1+(4*96)) = 356180858880 zijn.

Bij het eerste scenario gaat de burger op 1 maart 2007 verhuizen. De persoons-, adres- en de nieuwe adresgegevens worden meegegeven. Het referentienummer is 1111111111. De burger gaat zelfstandig wonen. Verder zijn er geen meeverhuizers. Dit scenario is weergegeven in Figuur 4.11.



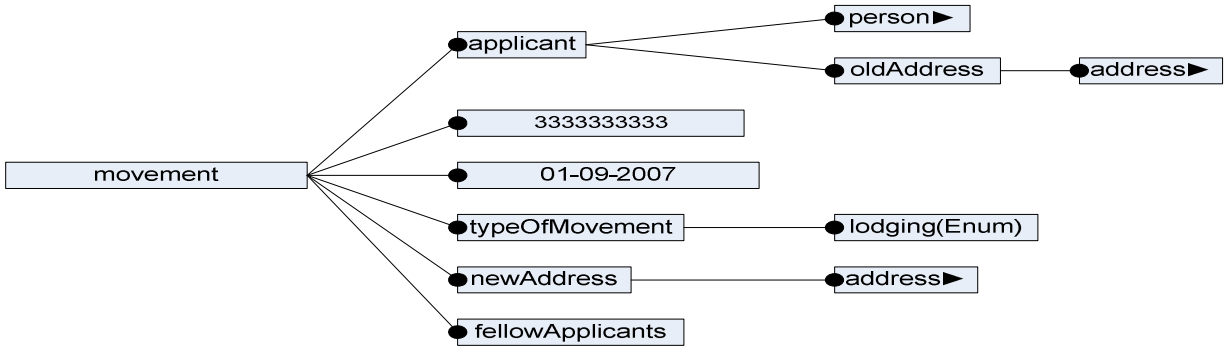
Figuur 4.11: scenario 1 voor het doorgeven van verhuizing

Bij het tweede scenario gaat de burger op 1 juni 2007 verhuizen. De persoons-, adres- en de nieuwe adresgegevens worden meegegeven. Het referentienummer is 2222222222. De burger gaat inwonen. Daarbij wordt de persoonsgegevens van de inwonende meegegeven. Verder zijn er twee meeverhuizers. De persoonsgegevens van deze medeverhuizers worden meegegeven. Dit scenario is weergegeven in Figuur 4.12.



Figuur 4.12: scenario 2 voor het doorgeven van verhuizing

Bij het laatste scenario gaat de burger op 1 september 2007 verhuizen. De persoons-, adres- en de nieuwe adresgegevens worden meegegeven. Het referentienummer is 3333333333. De burger gaat op kamers wonen. Verder zijn er geen meeverhuizers. Dit scenario is weergegeven in Figuur 4.13.



Figuur 4.13: scenario 3 voor het doorgeven van verhuizing

4.3.3 Opstellen van een feature model aan de hand van een XML schema

Voor het opstellen van feature modellen is er gebruik gemaakt van XML schema's die al eerder door eMAXX zijn opgesteld. Deze schema's bevatten veel informatie om feature modellen op te baseren. Hieronder zal beschreven worden hoe feature modellen verkregen kunnen worden aan de hand van eerder opgestelde XML schema's.

De *RootFeature* wordt verkregen door het element in het XML schema die in het *sequence* element van het element (die de zelfde naam heeft als de bestandnaam) staat. In de XML schema's, die door eMAXX zijn opgesteld, staan in het *sequence* element meestal twee elementen. Het ene element is *Applicant* die een feature wordt met als *parent* de *RootFeature*. Het andere element wordt gebruikt voor de *RootFeature*. Uiteraard kon hier het element "*movementRegistrationMessage*" ook als *RootFeature* worden genomen, maar is hier toepasselijker om de productnaam, die het tweede element is van het element "*movementRegistrationMessage*", als *RootFeature* te gebruiken.

In dit onderstaande voorbeeld (zie Code 4.1) heet de bestand "*movementRegistrationMessage*". Dus het element "*movement*" is de *RootFeature* met als *child* het element "*applicant*" welke een *SolitaryFeature* is.

```
1.    <xs:element name="movementRegistrationMessage">
2.       <xs:complexType>
3.          <xs:sequence>
4.             <xs:element name="applicant" type="movement:tPersonAndAddress"/>
5.             <xs:element name="movement" type="movement:tMovement"/>
6.          </xs:sequence>
7.       </xs:complexType>
8.    </xs:element>
```

Code 4.1: voorbeeld voor het verkrijgen van RootFeature

De *SolitaryFeatures* worden verkregen uit elementen die in het XML schema staan die geen *RootFeature* zijn. Als er bij een element een *sequence* element staat, dan is er sprake van nesting. Ook is er nesting wanneer het element niet een standaard type maar een gedefinieerd type is met meer elementen. De elementen in het *sequence* element zijn *SolitaryFeatures* met als *parent* feature het element waarin het *sequence* element staat. Wanneer een element een standaard type is (zoals int, string of float) dan heeft deze feature een attribuut met type die overeenkomt met de standaard type van het element. Zo wordt bepaald welke *SolitaryFeatures* attribuut bevatten.

In het volgende voorbeeld (zie Code 4.2) zijn de elementen "*date*" en "*referenceNumber*" *SolitaryFeatures* met als *parent* de *RootFeature* "*movement*". De *RootFeature* is afkomstig van het vorige voorbeeld. De *SolitaryFeature* "*referenceNumber*" heeft geen *child* omdat het gedefinieerde type hiervan geen element bevat.

```
1.    <xs:complexType name="tMovement">
2.       <xs:sequence>
3.          <xs:element name="referenceNumber"type="movement:tReferenceNumber"/>
4.          <xs:element name="date" type="xs:date" nillable="true"/>
5.       </xs:sequence>
6.    </xs:complexType>
7.    <xs:simpleType name="tReferenceNumber">
8.       <xs:restriction base="xs:string">
9.          <xs:maxLength value="50"/>
10.      </xs:restriction>
11.   </xs:simpleType>
```

Code 4.2: voorbeeld voor het verkrijgen van SolitaryFeature

De *FeatureGroup* kan verkregen worden uit *choice* element in het XML schema. De naam voor de *FeatureGroup* is het element waar in het *choice* element staat. *FeatureGroup* kan ook verkregen worden door het *enumeration* element. De naam van het *FeatureGroup* is dan gelijk aan het element waar de restrictie in staat.

De *GroupedFeatures* worden verkregen door de elementen die in het element staan en hebben als *parent* de *FeatureGroup* van die *choice* element. De attributen van de *GroupedFeatures* zijn de *choice* waarden. De *GroupedFeatures* worden ook verkregen door de *enumeration* elementen. Daarbij is het element waarin het enumeration element staat de *FeatureGroup* en dus de *parent* van de *GroupedFeatures*. De *BaseType* die bij het *enumeration* element behoort, is meteen het type van het attribuut van de *GroupedFeature* met als waarde de waarde van het *enumeration* element.

In dit onderstaande voorbeeld (zie Code 4.3) heeft een gedefinieerde choice type "tM*ovement"*. Dit type is verbonden aan de *RootFeature* van het eerste voorbeeld. Aangezien het element waarin het *choice* element staat gebruikt wordt voor de *RootFeature*, wordt er een ander naam gegeven aan de *FeatureGroup* die door dit *choice* element verkregen wordt. De *GroupedFeatures* van deze *FeatureGroup* zijn de elementen die in het *choice* element staan. De namen van de elementen in het *choice* element zijn ook meteen de attribuutwaarden van de *GroupedFeatures*.

```
1.    <xs:complexType name="tMovement">
2.       <xs:sequence>
3.          <xs:choice>
4.             <xs:element name="independant"/>
5.             <xs:element name="lodging"/>
6.             <xs:element name="resident">
7.          </xs:choice>
8.       <xs:sequence>
9.    </xs:complexType>
```

Code 4.3: voorbeeld voor het verkrijgen van FeatureGroup en GroupedFeature

De cardinaliteiten van *SolitaryFeatures* worden bepaald door de elementeigenschappen *minOccurs* en *maxOccurs*. Elke element kan deze eigenschappen bevatten. Deze eigenschappen hebben een standaard waarde 1. Wanneer de waarde van *minOccurs = m* en de waarde van *maxOccurs = n*, dan is de cardinaliteit [m..n]. Wanneer beide eigenschappen niet zijn weergegeven is de cardinaliteit [1..1]. Het eerste element in de cardinaliteit verwijst naar *minOccurs*. Het tweede element van de cardinaliteit verwijst naar *maxOccurs*. Wanneer de waarde van *maxOccurs = unbounded*, dan is de bovengrens van cardinaliteit onbegrensd.

In het volgende voorbeeld (zie Code 4.4) heeft het "*fellowApplicant*" element een *minOccurs="0" maxOccurs="unbounded"*. Dit betekent dat de cardinaliteit van *SolitaryFeature*, die verkregen wordt door dit element, [0..*] is.

```
1.    <xs:element name="fellowApplicant" type="common:tPerson" minOccurs="0"
10.   maxOccurs="unbounded"/>
```

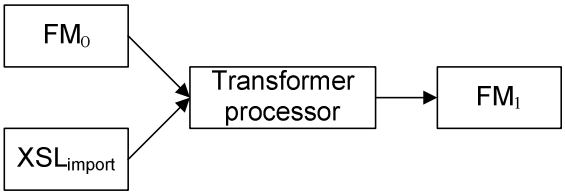
Code 4.4: voorbeeld voor het verkrijgen van cardinaliteit van Solitary Feature

De cardinaliteiten van de *FeatureGroup* is <1..1> wanneer die verkregen is door het *choice* element of door het *enumeration* element . Het *choice* element laat één waarde toe in de instantie. Dit geld ook voor de *enumeration* elementen.

4.3.4 Implementatie van de processtap

In deze sectie zal de implementatie van de processtap worden beschreven. Allereerst zal het concept van de transformatie worden besproken. Vervolgens zal de aanroep van de transformatie worden besproken. Ten slotte zullen de transformatieregels worden besproken.

Zoals eerder aangegeven worden alle transformaties in dit onderzoek uitgevoerd met behulp van een XSLT processor. In Figuur 4.14 zijn de invoer en de uitvoer van de transformer weergegeven. Er wordt een transformer geïnitieerd waarbij het XSL_{import} bestand aan de transformer wordt gekoppeld. In dit XSL bestand zitten de transformatieregels voor de transformatie T_0. Vervolgens wordt een aanroep gedaan om een transformatie uit te voeren. Bij de aanroep wordt een invoer en een uitvoer bestand aangegeven. Het invoer bestand is het FM_0 bestand waarin het basis feature model van het doorgeven van een verhuizing staat. De output is het FM_1 bestand waarin het resultaat van de transformatie staat en dus het feature model waarin de referentie features geïmporteerd zijn.



Figuur 4.14: invoer en uitvoer bestanden van de transformer om referentie features in het feature model te importeren

In Code 4.5 is de implementatie van de aanroep van de transformatie T_0 weergegeven die in de JSP pagina (*JSP Combine FM*) staat. De JSP pagina haalt het feature model van het doorgeven van een verhuizing op. Dit is het invoerbestand voor de transformer. Daarnaast wordt het XSL_{import} bestand opgehaald waarin de transformatieregels voor de transformatie T_0 staan. Vervolgens wordt de transformatie T_0 uitgevoerd waarbij het feature model FM_0 naar het feature model FM_1 wordt getransformeerd.

```
1.    <% //Dit zorgt ervoor dat het feature model en de referentie feature modellen in
2.    één file komen te staan.
3.
4.    // this is the input of the tranformation
5.    File fInput = new File("movementXFeatureModel.xfm");
6.
7.    // this is the file were the transformation rules are written
8.    File fTransformationRules = new File("T0TransformationRules.xsl");
9.
10.   // this is the ouput file of the tranformation
11.   File fResult = new File("combinedFMMovement.xml");
12.
13.   StreamSource FM0= new StreamSource(fInput);
14.   StreamSource XSLimport = new StreamSource(fTransformationRules);
15.   StreamResult FM1= new StreamResult(fResult);
16.
17.   TransformerFactory factory = TransformerFactory.newInstance();
18.   Transformer T0 = factory.newTransformer(XSLimport);
19.   T0.transform(FM0, FM1); %>
```

Code 4.5: implementatie van de aanroep van de transformatie T_0

Hieronder in Tabel 4.1 zijn de gebruikte transformatieregels van de transformatie T_0 weergegeven. In de linker kolom zijn de namen van de transformatieregels weergegeven. In de middelste kolom zijn de condities weergegeven die geldig moeten zijn in het feature model FM_0. In de rechter kolom staan de transformatieacties die uitgevoerd worden wanneer de condities voor deze transformatieregel geldig zijn. Het resultaat van deze transformatieacties worden weggeschreven naar het feature model FM_1.

Transformatieregel	**Condities die geldig moeten zijn in het feature model FM_0 voor de transformatieacties**	**De transformatieacties van transformatie T_0**
TR_1	Element ≠ SolitaryReference	Kopieer element met attributen
TR_2	Als element = SolitaryReference ^ SolitaryReference = docPerson	Kopieer het feature model Person vanaf de root feature
TR_3	Als element = SolitaryReference ^ SolitaryReference = docAddress	Kopieer het feature model Address vanaf de root feature
TR_4	Als element = SolitaryReference ^ SolitaryReference = docOrganization	Kopieer het feature model Organization vanaf de root feature

Tabel 4.1: Transformatieregels van de transformatie T_0

- ***TR_1***

  Voor elke element van het feature model FM_0 wordt gekeken of dit ongelijk is aan SolitaryReference. Indien dit het geval is, dan wordt dit element samen met de bijbehorende attributen gekopieerd naar het feature model FM_1.

- ***TR_2***

  Als *element = SolitaryReference* en *SolitaryReference = docPerson*, dan wordt het feature model Person vanaf de root feature gekopieerd in het feature model FM_1

- ***TR_3***

  Als *element = SolitaryReference* en *SolitaryReference = docAddress*, dan wordt het feature model Address vanaf de root feature gekopieerd in het feature model FM_1

- ***TR_4***

  Als *element = SolitaryReference* en *SolitaryReference = docOrganization*, dan wordt het feature model Person vanaf de root feature gekopieerd in het feature model FM_1

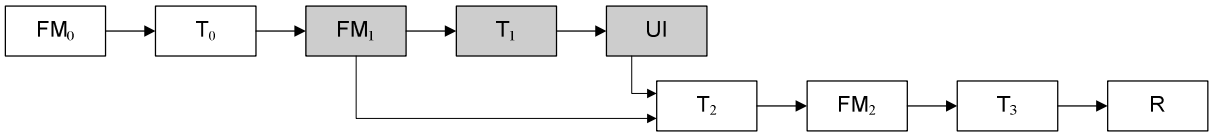
In Code 4.6 is het XSL_{import} bestand weergegeven die voor het importeren van het referentie features zorgt. De conditie voor de transformatieregel TR_1 staat in regel 3 en de transformatieactie van deze transformatieregel staat in regel 2 t/m 6. De condities voor de transformatie regel TR_2 staan in regel 9 en in regel 15 en de transformatieactie van deze transformatieregel staat in regel 16. De condities voor de transformatie regel TR_3 staan in regel 9 en in regel 18 en de transformatieactie van deze transformatieregel staat in regel 19. De condities voor de transformatie regel TR_4 staan in regel 9 en in regel 21 en de transformatieactie van deze transformatieregel staat in regel 22.

```
1.    <xsl:template  match="*">
2.       <xsl:copy>
3.          <xsl:copy-of  select="node[name() != 'fm:SolitaryReference']"/>
4.          <xsl:copy-of select="@*"/>
5.          <xsl:apply-templates />
6.       </xsl:copy>
7.    </xsl:template>
8.
9.    <xsl:template  match="fm:SolitaryReference">
10.      <xsl:variable name="docPerson" select="document('persoonXFeatureModel.xfm')"/>
11.      <xsl:variable name="docAddress" select="document('addressXFeatureModel.xfm')"/>
12.      <xsl:variable name="docOrganization"
13.      select="document('organisatieXFeatureModel.xfm')"/>
14.      <xsl:choose>
15.         <xsl:when test="@fm:value = 'refPerson'">
16.            <xsl:copy-ofselect="$docPerson//fm:RootFeature"/>
17.         </xsl:when>
18.         <xsl:when test="@fm:value = 'refAddress'">
19.            <xsl:copy-of select="$docAddress//fm:RootFeature"/>
20.         </xsl:when>
21.         <xsl:when test="@fm:value = 'refOrganization'">
22.            <xsl:copy-of select="$docOrganization//fm:RootFeature"/>
23.         </xsl:when>
24.         <xsl:otherwise></xsl:otherwise>
25.      </xsl:choose>
26.      <xsl:apply-templates />
27.   </xsl:template>
```

Code 4.6: implementatie van transformatieregels van de transformatie T_0

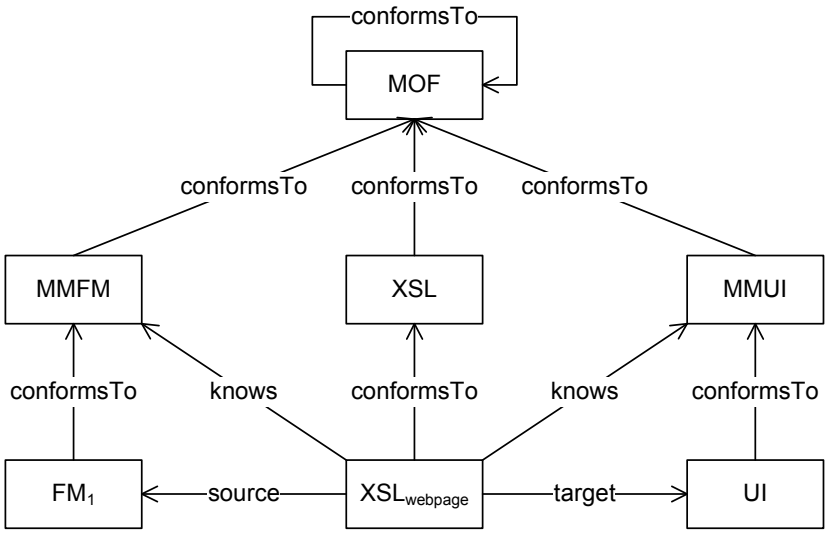
4.4 Processtap: generate web page of selected feature model

Nadat de referentie features in het feature model FM_1 zijn geïmporteerd, wordt dit feature model in dezelfde JSP pagina (*JSP Combine FM*) naar een UI getransformeerd waarbij een webpagina wordt gegenereerd. Bij het genereren van een webpagina worden de features van het feature model FM_1 (die nog geen waarden hebben) getransformeerd naar een UI met behulp van transformatie T_1. Deze transformatie is in Figuur 4.15 terug te vinden waarbij de componenten die hier een rol spelen een donkere achtergrondkleur hebben.



Figuur 4.15: de transformatie T_1 voor het genereren van een webpagina van features die nog geen waarde hebben in het feature model FM_1

In Figuur 4.16 (zie volgende pagina) is het transformatie overzicht weergegeven voor transformatie T_1 die een feature model FM_1 naar het model UI transformeert. Het model UI geeft het model aan van de webpagina die wordt gegenereerd. De transformatieregels voor deze transformatie T_1 staan in $XSL_{webpage}$ bestand.



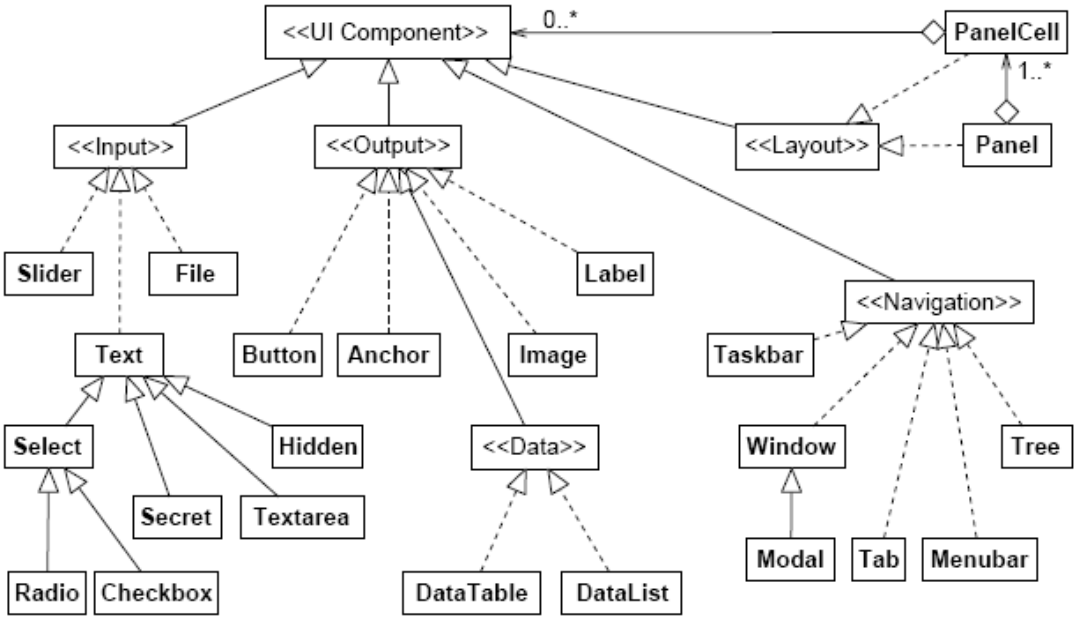
Figuur 4.16: het transformatie overzicht voor de transformatie T_1

Nadat de webpagina is gegenereerd wordt deze geïmporteerd in de huidige JSP pagina zodat de burger meteen de gegenereerde webpagina kan zien en invullen. Het proces belandt vervolgens in toestand: *generated web page of selected feature model* die in sectie 4.5 wordt besproken.

Het metamodel MMFM is al in sectie 4.3.1 weergegeven. In sectie 4.4.1 wordt het metamodel UI weergegeven. In sectie 4.4.2 wordt de implementatie van de processtap besproken.

4.4.1 Metamodel User Interface

In deze sectie zal het metamodel van de User Interface beschreven worden welke in het artikel [Mes07] is behandeld. In Figuur 4.17 is het UML metamodel MMUI weergegeven.



Figuur 4.17: UML metamodel MMUI [Mes07]

De opgestelde transformatieregels, die in het $XSL_{webpage}$ bestand staan, hebben betrekking op een bepaald gedeelte van dit metamodel. Uiteraard kunnen er andere transformatieregels opgesteld worden waarbij er meerdere entiteiten betrokken worden. In deze sectie zullen alleen die entiteiten worden toegelicht die gebruikt worden door de opgestelde transformatieregels. Aangezien het in dit onderzoek niet zozeer om een goede interface gaat, is er minder aandacht besteedt aan de interface van de webpagina en daardoor zijn er minder entiteiten gebruikt. De gebruikte entiteiten in dit onderzoek zijn: *UI component*, i*nput*, *text*, *textarea*, *hidden*, *select*, *radio*, *checkbox*, *output*, *button* en *label*.

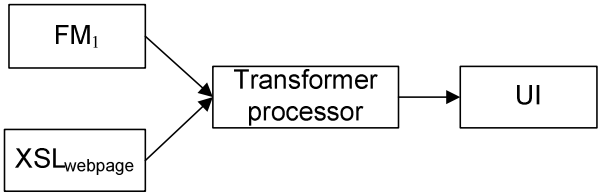
De *UI component* kan bestaan uit een *input* of uit een *output*. *Input* wordt gebruikt om invoer van de burgers te krijgen via de webpagina. *Input* kan van de vorm *text* zijn. De *text* kan *hidden*, *textarea* of *select* zijn. Bij *hidden* wordt de component niet getoond op de webpagina. Bij *textarea* wordt een veld getoond op de webpagina waar tekst geschreven kan worden. Bij *select* kan het een *radio* of een *checkbox* zijn. Dit zijn componenten op de webpagina waarop geklikt kan worden met de muis om een selectie te maken tussen de keuzemogelijkheden.

Output wordt gebruikt om iets te tonen op de webpagina waar niet op geschreven kan worden. In dit onderzoek wordt alleen *button* en *label* als *output* componenten gebruikt. De *button* zorgt ervoor dat een actie wordt uitgevoerd. In dit onderzoek wordt de *button* gebruikt om naar de volgende webpagina te gaan waarbij de ingevulde gegevens op de webpagina naar de webserver worden gestuurd. De *label* wordt gebruikt om de vraag voor een bepaald invoergegeven te tonen. Zo weet de burger bij welke invoerveld welke gegevens hij of zij moet invullen.

4.4.2 Implementatie van de processtap

In deze sectie zal de implementatie van de processtap worden beschreven. Allereerst zal het concept van de transformatie worden besproken. Vervolgens zal de aanroep van de transformatie worden besproken. Ten slotte zullen de transformatieregels worden besproken.

In Figuur 4.18 is de invoer en de uitvoer van de transformer weergegeven. Er wordt een transformer geïnitieerd waarbij het $XSL_{webpage}$ bestand aan de transformer wordt gekoppeld. Vervolgens wordt een aanroep gedaan om een transformatie uit te voeren. Bij de aanroep wordt een invoer en een uitvoer bestand aangegeven. Het invoer bestand is het FM_1 bestand waarin het feature model FM_1 staat. De output is een gegenereerd HTML bestand (UI) waarin het resultaat van de transformatie staat.



Figuur 4.18: invoer en uitvoer bestanden van de transformer om een webpagina te genereren

In Code 4.7 is de implementatie van de aanroep van de transformatie T_1 weergegeven in de JSP pagina (*JSP Combine FM*). De JSP pagina haalt het feature model op van het doorgeven van verhuizing waarbij de referentie features zijn geïmporteerd. Dit is het invoer bestand voor de transformer. Daarnaast wordt het $XSL_{webpage}$ bestand opgehaald waarin de transformatieregels staan voor de transformatie T_1. Vervolgens wordt de transformatie T_1 uitgevoerd waarbij het feature model FM_1 naar het model UI wordt getransformeerd. In de laatste regel wordt de gegenereerde webpagina in de huidige JSP geïmporteerd waardoor deze webpagina meteen zichtbaar wordt voor de burger.

```
1.    <%//dit zorgt ervoor dat een web pagina wordt gegenereerd.
2.
3.    // this is the input file of the tranformation
4.    File fInput = new File("combinedFMMovement.xml");
5.
6.    // this is the file were the transformationrules are written
7.    File fTransformationRules = new File("T1TransformationRules.xsl");
8.
9.    // this is the output file of the tranformation
10.   File fResult = new File("HTMLMovement.html");
11.
12.   StreamSource FM1 = new StreamSource(fInput);
13.   StreamSource XSLwebpage = new StreamSource(fTransformationRules);
14.   StreamResult UI = new StreamResult(fResult);
15.
16.   TransformerFactory factory3 = TransformerFactory.newInstance();
17.   Transformer T1 = factory3.newTransformer(XSLwebpage);
18.   T1.transform(FM1, UI); %>
19.
20.   <jsp:include page="HTMLMovement.html" flush="false" />
```

Code 4.7: implementatie van de aanroep van de transformatie T_1

In Tabel 4.2 zijn de gebruikte transformatieregels van de transformatie T_1 weergegeven. In de linker kolom zijn de namen van de transformatieregels weergegeven. In de middelste kolom zijn de condities weergegeven die geldig moeten zijn in het feature model FM_1. In de rechter kolom staan de transformatieacties die uitgevoerd worden wanneer de condities voor deze transformatieregel geldig zijn. Het resultaat van deze transformatieacties worden weggeschreven naar het model UI.

Transformatieregel	**Condities die geldig moeten zijn in het feature model FM_1 voor de transformatieacties**	**De transformatieacties van transformatie T_1**
TR_1	Eerste element	creëren elementen *html*, *head*, *title*, *body* en *form*
TR_2	Als element = SolitaryFeature ^ *child* element = Description	creëren element *label*
TR_3	Als element = SolitaryFeature ^ *child* element = Attribute ^ attribuut *child* element van element Attribute = *string*, *integer*, *float* en ‘’ (leeg)	creëren element *input* met attributen *type*(text), *name* en *value*
TR_4	Als element = FeatureGroup ^ # *child* element GroupedFeature ≠ 1	creëren element *label* en creëren element *input* met attributen *type*(radio), *name* en *value*

Tabel 4.2: Transformatieregels van de transformatie T_1

- ***TR_1***

  Bij het eerste element van het feature model FM_1 worden de elementen *HTML*, *head*, *title*, *body* en *form* gecreëerd, uiteraard in de syntax van HTML. Bij het element form worden twee input elementen in de webpagina gecreëerd. Het eerste element input wordt gecreëerd met attributen *name* (hidden), *type* (paramFileName) en *value* (naam van de rootfeature). Deze wordt gebruikt voor het doorsturen van de naam van de RootFeature wanneer de ingevulde gegevens van de webpagina naar de server worden gestuurd. Het tweede element wordt gecreëerd met attributen "*name*" (Submit), "*type*" (submit) en "*value*" (verzenden). Met deze wordt een knop getoond op de webpagina. Wanneer er op deze knop wordt gedrukt, wordt de ingevulde gegevens naar de webserver gestuurd.

  In Code 4.8 is de implementatie van de transformatieregel TR_1 weergegeven. Deze template zorgt ervoor dat de webpagina met een formulier wordt gecreëerd. Het element in het formulier zorgt ervoor dat de naam van het product wordt doorgegeven zodat de volgende JSP pagina weet om welke product het gaat. De conditie voor de transformatieregel TR_1 staat in regel 1 en de transformatieactie van deze transformatieregel staat in regel 2 t/m 17.

```
1.    <xsl:template match="/">
2.       <html> <head><title>a</title></head>
3.       <body>
4.          <form action="http://localhost:8080/test/test2/test.jsp" method="POST"
5.           name="aanvraag">
6.             <xsl:element name="input">
7.                <xsl:attribute name="type">hidden</xsl:attribute>
8.                <xsl:attribute name="name">paramFileName</xsl:attribute>
9.                <xsl:attribute name="value">
10.                  <xsl:value-of select="fm:FeatureModel/fm:RootFeature/@fm:value"/>
11.               </xsl:attribute>
12.            </xsl:element>
13.            <xsl:apply-templates />
14.            <input type="submit" value="Verzenden" name="Submit"/>
15.         </form>
16.      </body>
17.      </html>
18.   </xsl:template>
```

Code 4.8: implementatie van de transformatieregel TR_1 van de transformatie T_1

- ***TR_2***

  Als element = SolitaryFeature en *child element = Description* is, dan wordt een *label* in de webpagina gecreëerd met de waarde van de *description* erop. De vraag die aan de burger wordt gesteld om het desbetreffende gegeven te vragen is de waarde van het *child* element *description*.

  In Code 4.9 (zie volgende pagina) is de implementatie van de transformatieregel TR_2 van de transformatie T_1 weergegeven. Wanneer het element *description* wordt gevonden, dan wordt een label gecreëerd met daarop de waarde van dit element. Dit is de vraag die aan de burger gesteld wordt om deze SolitaryFeature een waarde te geven.

```
1.    <xsl:for-each select="fm:Annotation/fm:Description">
2.       <label style="width: 150px">
3.          <xsl:value-of select="@fm:value"/>
4.       </label>
5.    </xsl:for-each>
```

Code 4.9: implementatie van de transformatieregel TR_2 van de transformatie T_1

- ***TR_3***

Als element = SolitaryFeature en *child* element = Attribute en atribuut van het *child* element van element Attribute = *string*, *integer, float* en ‘’ (leeg) is, dan wordt een input element gecreëerd met attributen *type* (*tekst*), *name* (pathnaam) en *value* in de webpagina wanneer het attribuut *value* gelijk is aan *string*, *integer*, *float* en ‘’ (leeg). Hiermee wordt er een tekst veld op de webpagina getoond waarin de burger de waarde van de feature van het feature diagram (van het feature model FM_1) kan opgeven.

In Code 4.10 is de implementatie van de transformatieregel TR_3 van transformatie T_1 weergegeven. Wanneer het element *attribute* wordt gevonden, dan wordt er naar de elementen String, Integer en Float gekeken om te bepalen of de waarden van de attributen van deze elementen het “*type*” zelf is of dat het leeg is. Zodra de waarde van het attribuut hetzelfde type is of leeg is, dan wordt een input element met type text gecreëerd. De naam van dit element is de naam van SolitaryFeature samen met de namen van alle *parent* namen van de SolitaryFeature waarbij tussen elke naam een “/” teken staat. Zodra de waarde van het attribuut niet dezelfde is, dan betekent dat er al een waarde is gegeven voor deze *SolitaryFeature* en vervolgens wordt dit element overgeslagen.

```
1.    <xsl:for-each select="fm:Attribute">
2.       <xsl:choose>
3.          <xsl:when test="./fm:String/@fm:value='String'or
4.          ./fm:Integer/@fm:value = 'Integer' or ./fm:Float/@fm:value = 'Float' or
5.          ./fm:String/@fm:value = '' or ./fm:Integer/@fm:value = '' or
6.          ./fm:Float/@fm:value = '' " >
7.             <xsl:element name="input">
8.                <xsl:attribute name="type">Text</xsl:attribute>
9.                <xsl:attribute name="name">
10.                  <xsl:for-each select="ancestor-or-self::*">
11.                     <xsl:text>/</xsl:text>
12.                     <xsl:value-of select="@fm:value"/>
13.                  </xsl:for-each>
14.               </xsl:attribute>
15.               <xsl:attribute name="value"></xsl:attribute>
16.            </xsl:element>
17.         </xsl:when>
18.         <xsl:otherwise></xsl:otherwise>
19.      </xsl:choose>
20.   </xsl:for-each>
```

Code 4.10: implementatie van de transformatieregel TR_3 van de transformatie T_1

- ***TR_4***

  Als element = FeatureGroup en het aantal *child* element GroupedFeature ≠ 1 is, dan wordt er een label met de waarde van de *description* erop in de webpagina gecreëerd. Vervolgens wordt er naar de *GroupedFeatures* in dit *FeatureGroup* gekeken. Bij elke *GroupedFeature* wordt het element *input* met als attribuut *type* (*radio*), *name* (naam van de *FeatureGroup*) en *value* (*StringValue* van de *GroupedFeature*) gecreëerd. Het attribuut *name* is voor elke *GroupedFeature* binnen dezelfde *FeatureGroup* dezelfde, zodat het bekend is bij welke *FeatureGroup* de gemaakte keuze hoort.

  In Code 4.11 is de implementatie van de transformatieregels TR_4 van de transformatie T_1 weergegeven. Wanneer het element *attribute* wordt gevonden, dan wordt gekeken hoeveel *GroupedFeatures* er zijn. Als dit aantal gelijks is aan 1, dan betekend dit dat er al een keuze is gemaakt en wordt er verder niks mee gedaan. Anders is er nog geen keuze gemaakt en dus wordt een invoer element in de webpagina gecreëerd met type radio. Alle GroupedFeatures krijgen dezelfde naam zodat de selectie tussen deze features kan worden gemaakt. De condities voor de transformatieregel TR_4 staan in regel 1, 4 en 5 en de transformatieacties van deze transformatieregel staan in regel 6 t/m 27.

```
1.    <xsl:for-each select="fm:FeatureGroup">
2.       <br/>
3.       <xsl:choose>
4.          <xsl:when test="count(fm:GroupedFeature)=1"></xsl:when>
5.          <xsl:otherwise>
6.             <label style="width: 150px">
7.                <xsl:value-of select="fm:Annotation/fm:Description/@fm:value"/>
8.             </label>
9.             <xsl:for-each select="fm:GroupedFeature">
10.               <input>
11.                  <xsl:attribute name="type">radio</xsl:attribute>
12.                  <xsl:attribute name="name">
13.                     <xsl:for-each select= "ancestor::*">
14.                        <xsl:text>/</xsl:text>
15.                        <xsl:value-of select="@fm:value"/>
16.                     </xsl:for-each>
17.                  </xsl:attribute>
18.                  <xsl:attribute name="value">
19.                     <xsl:value-of select="@fm:value"/>
20.                  </xsl:attribute>
21.                  <xsl:for-each select="fm:Attribute/fm:String/
22.                  fm:StringProperties/fm:StringValue">
23.                     <xsl:value-of select="@fm:value" />
24.                  </xsl:for-each>
25.               </input>
26.            </xsl:for-each>
27.            <br/>
28.         </xsl:otherwise>
29.      </xsl:choose><xsl:apply-templates />
30.   </xsl:for-each>
```

Code 4.11: implementatie van de transformatieregel TR_4 van de transformatie T_1

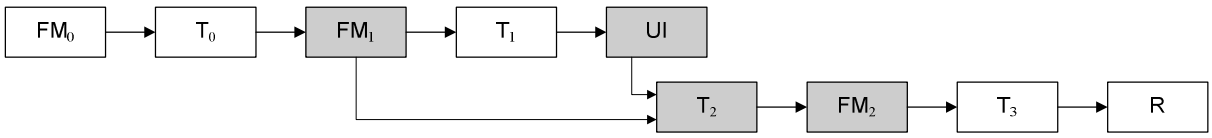
4.5 Toestand: generated web page of selected feature model

Het proces is nu in een toestand: *generated web page of selected feature model* beland. Nu wordt van de burger verwacht dat hij of zij deze webpagina gaat invullen. Een deel van de gegenereerde webpagina voor het doorgeven van verhuizing is in Figuur 4.19 (zie volgende pagina) afgebeeld. Het proces gaat verder wanneer de knop "*verzend*" is gedrukt, die naar de volgende JSP pagina (JSP Send Report) verwijst. In deze JSP pagina gaat het proces verder met de volgende processtap: *specialize selected feature model by means of input values.*

Figuur 4.19: een gedeelte van de gecreëerde webpagina voor het doorgeven van verhuizing

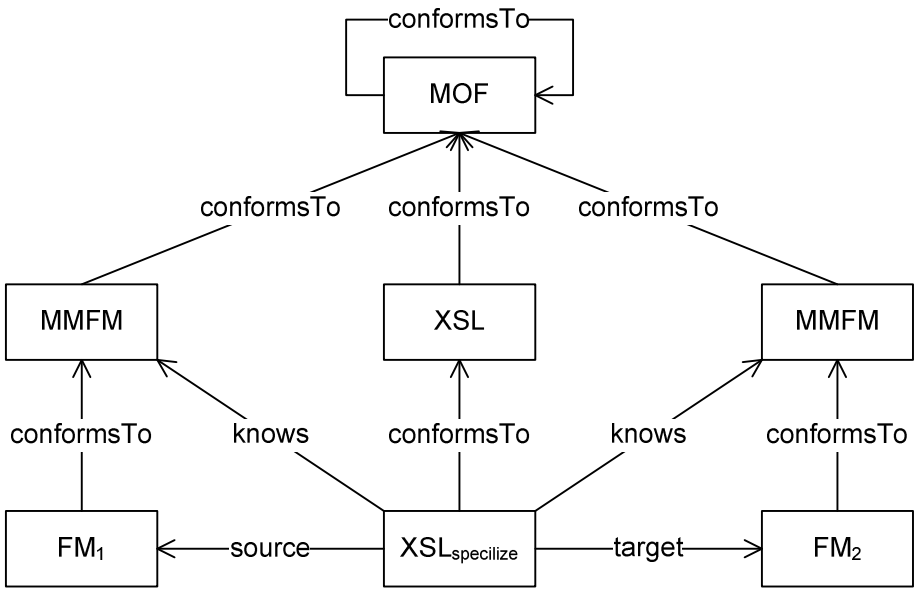
4.6 Processtap: specialize selected feature model by means of input values

Nadat de burger de gegevens op de webpagina heeft ingevuld, gaat het proces verder met de processtap: *specialize selected feature model by means of input values* die in JSP pagina (JSP Send Report) is geïmplementeerd. Deze processtap zorgt ervoor dat de ingevulde waarden op de webpagina weggeschreven worden naar het feature model FM_2. Wanneer de features van het feature model een waarde krijgen en of de keuzes beperkter of geselecteerd worden, heet dit het specialiseren van het feature model. Er zijn verschillende soorten methoden om het feature model te specialiseren. In deze sectie zal het specialiseren van het feature model met behulp van transformatie T_2 gaan. Deze transformatie is in Figuur 4.20 te zien waarbij de componenten die hier een rol spelen een donker achtergrondkleur hebben.



Figuur 4.20: de transformatie T_2 waarbij het feature model FM_1 wordt gespecialiseerd

In Figuur 4.21 (zie volgende pagina) is het transformatie overzicht weergegeven voor de transformatie T_2 die een feature model FM_1 naar het feature model FM_2 transformeert. FM_2 is een gespecialiseerde feature model van het feature model FM_1. De transformatieregels voor deze transformatie T_2 staan in $XSL_{specialize}$ bestand.



Figuur 4.21: het transformatie overzicht voor de transformatie T_2

In sectie 4.6.1 worden de methoden om een feature model te specialiseren besproken. In sectie 4.6.2 wordt de implementatie van de processtap besproken.

4.6.1 Methoden om een feature model te specialiseren

Aan de hand van de volgende methoden kan een feature model gespecialiseerd worden:

- ***Handmatig specialiseren***

  Een methode om het feature model te specialiseren is simpelweg handmatig intypen van de waarden in het feature model met behulp van een programma zoals word, notepad etc.

- ***Specialiseren met behulp van een programma waarmee feature modellen worden opgesteld***

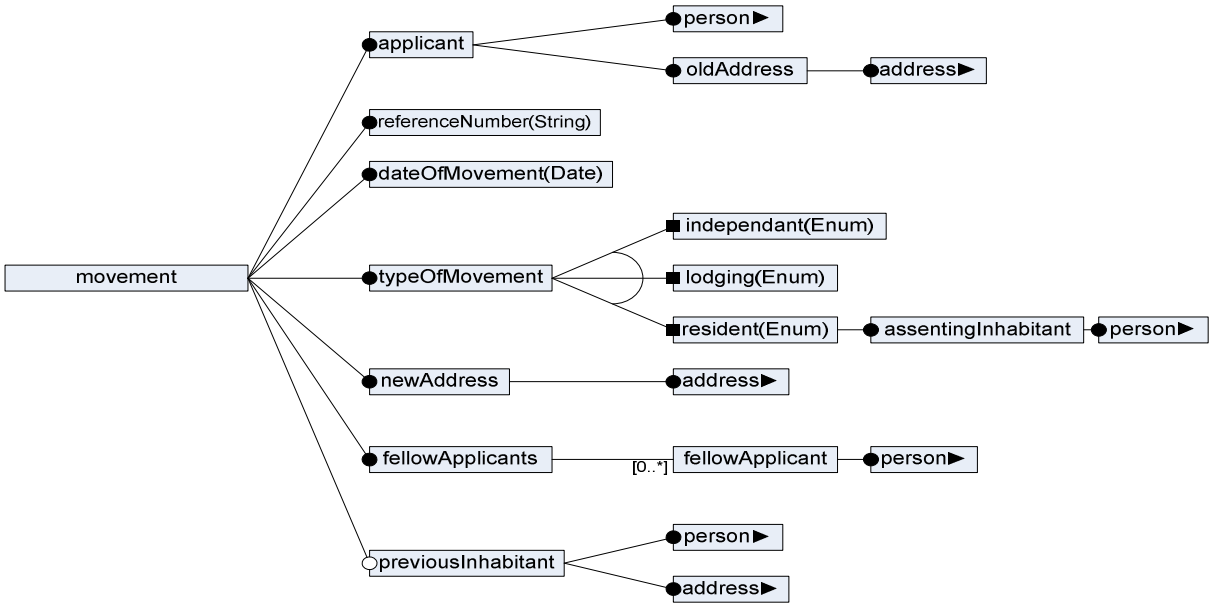
  Een ander methode om het feature model te specialiseren is met behulp van de programma's waarmee de feature modellen kunnen worden opgesteld zoals FeaturePlugin, Pure Variant en XFeature.

- ***Automatisch specialiseren***

  Nog een ander methode is dat het specialiseren automatisch gaat aan de hand van waarden die al opgegeven zijn. Deze waarden worden opgehaald en weggeschreven naar het feature model.

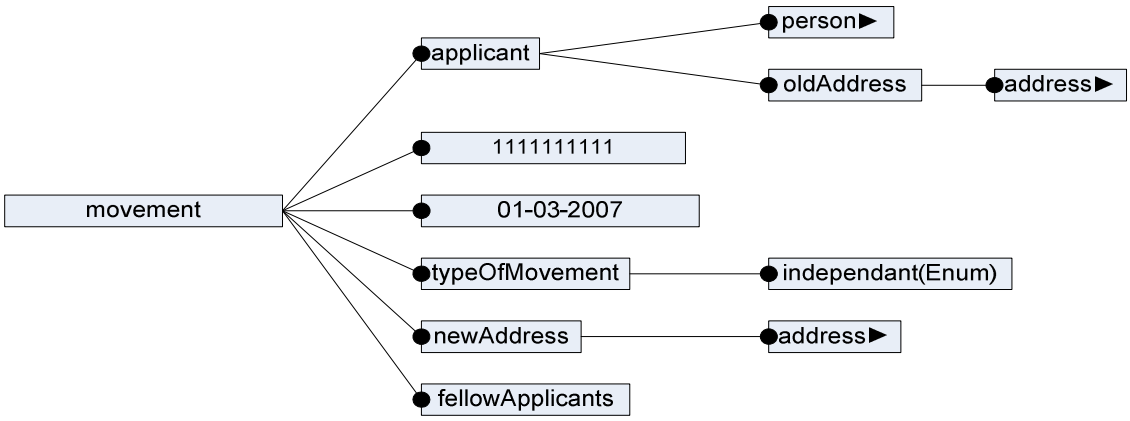
Het specialiseren van een feature model met behulp van feature model programma of handmatig specialiseren is erg omslachtig en tijdrovend. Wat beter en tijdefficiënter is dat het specialiseren automatisch gaat. In dit onderzoek zullen daarom de feature modellen automatisch worden gespecialiseerd met behulp transformaties.

In Figuur 4.22 is het feature model van het product: *verhuizing doorgeven* weergegeven dat nog niet is gespecialiseerd.



Figuur 4.22: niet gespecialiseerde feature model van het product: verhuizing doorgeven

In Figuur 4.23 is een voorbeeld van een gespecialiseerd feature model van het product: *verhuizing doorgeven* weergegeven. Hierin zijn het referentienummer 1111111111 en de datum 01-03-2007 van verhuizing ingevuld. Verder is er aangegeven dat deze persoon zelfstandig gaat wonen en verder geen medeverhuizers heeft.



Figuur 4.23: voorbeeld van een gespecialiseerd feature model van het product: verhuizing doorgeven

4.6.2 Implementatie van de processtap

In deze sectie zal de implementatie van de processtap worden besproken. Ten eerste zal het concept van de transformatie T_2 worden besproken. Daarna zal de aanroep van de transformatie T_2 worden besproken. Ten slotte zullen de transformatieregels van de transformatie T_2 worden besproken.

In deze processtap wordt het feature model FM_1 automatisch gespecialiseerd met behulp van transformatie T_2. Voordat deze transformatie wordt uitgevoerd moeten de ingevulde waarden op de webpagina in een XML opgeslagen worden, omdat XSLT met XML bestanden werkt. De implementatie van dit proces in JSP pagina (*JSP Send Report*) is in Code 4.12 weergegeven.

```
1.    try
2.    {
3.       String strPath = "C:\\Program Files\\Apache Software
4.       Foundation\\Tomcat 6.0\\webapps\\test\\test2\\xsl\\dataoutput.xml";
5.       File strFile = new File(strPath);
6.       Writer objWriter = new BufferedWriter(new FileWriter(strFile));
7.
8.       objWriter.write("<?xml version=\"1.0\" encoding=\"UTF-8\" ?> ");
9.       objWriter.write("<dataoutput> "  + "\n");
10.
11.      Enumeration params = request.getParameterNames();
12.      while( params.hasMoreElements() )
13.      {
14.         String paramName = (String) params.nextElement();
15.         if(request.getParameter(paramName)!= "")
16.         {
17.            String paramName2 = paramName.substring(1);
18.            String paramName3 = paramName2.replace("/", "-");
19.            objWriter.write("<" + paramName3 + " > " +
20.            request.getParameter( paramName ) + "</" + paramName3 + ">" + "\n");
21.         }
22.      }
23.      objWriter.write("</dataoutput> "  + "\n");
24.      objWriter.flush();
25.      objWriter.close();
26.   }
27.   catch (UnsupportedEncodingException e)
28.   {
29.      System.out.println("This VM does not support the Latin-1 character set.");
30.   }
31.   catch (IOException e)
32.   {
33.      System.out.println(e.getMessage());
34.   }
```

Code 4.12: implementatie van het opslaan van de ingevulde waarden op de webpagina naar het dataouput bestand

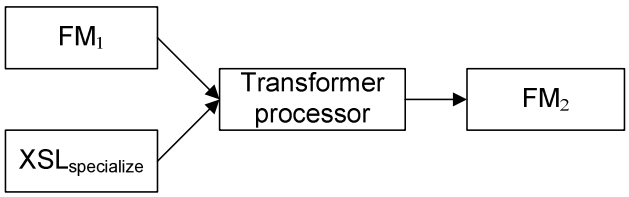
Eerst wordt het bestand aangemaakt en vervolgens worden de waarden, die ingevuld zijn op de webpagina, in dit bestand opgeslagen. Het eerste element wordt alleen gebruikt om het gedeelte van het bestand te selecteren waarop de transformatie uitgevoerd moet worden. Hierdoor begint het bestand met het element *dataoutput*. Vervolgens worden de waarden van velden, die niet leeg zijn, één voor één samen met hun veldnamen opgeslagen in het bestand totdat alle waarden op de webpagina zijn uitgelezen.

Een voorbeeld van een dataoutput bestand waarin de waarden van de webpagina zijn opgeslagen is in Code 4.13 (zie volgende pagina) weergegeven. Nadat de ingevulde waarden in het dataoutput bestand zijn opgeslagen, kan de transformatie T_2 worden uitgevoerd.

```
1.    <dataoutput>
2.       <movement-applicant-oldAddress-address-zipCode-Attribute>
3.          1111 pp
4.       </movement-applicant-oldAddress-address-zipCode-Attribute>
5.       <movement-applicant-oldAddress-address-street-name-Attribute>
6.          verwegstraat
7.       </movement-applicant-oldAddress-address-street-name-Attribute>
8.       <movement-applicant-oldAddress-address-city-name-Attribute >
9.          nooitgevonden stad
10.      </movement-applicant-oldAddress-address-city-name-Attribute>
11.   </dataoutput>
```

Code 4.13: een voorbeeld van opgeslagen gegevens van de webpagina in een dataouput bestand

In Figuur 4.24 is de invoer en de uitvoer van de transformer weergegeven. Er wordt een transformer geïnitieerd waarbij het $XSL_{specialize}$ bestand aan de transformer wordt gekoppeld. Vervolgens wordt een aanroep gedaan om een transformatie uit te voeren. Bij de aanroep wordt een invoer en een uitvoer bestand aangegeven. De invoer bestand is het FM_1 bestand waarin het feature model van het doorgeven van verhuizing staat. De uitvoer is het FM_2 bestand waarin het resultaat van de transformatie staat.



Figuur 4.24: invoer en uitvoer bestanden van de transformer om het feature model FM_1 te specialiseren

In Code 4.14 is de implementatie van de aanroep van de transformatie T_2 in de JSP pagina (*JSP Send Report*) weergegeven. De JSP pagina haalt het feature model voor het doorgeven van een verhuizing op. Dit is het invoerbestand voor de transformer. Daarnaast wordt het $XSL_{specialize}$ bestand opgehaald waarin de transformatieregels staan. Vervolgens wordt de transformatie T_2 uitgevoerd waarbij het feature model FM_1 naar het feature model FM_2 wordt getransformeerd. Het feature model FM_2 is het gespecialiseerde feature model van het feature model FM_1.

```
1.    <%// dit zorgt ervoor dat de waarden op de webpagina weggeschreven worden naar het
2.    feature model.
3.
4.    // this is the input file of the tranformation
5.    File fInput = new File("SelectedFM+".xml");
6.
7.    // this is the file were the transformationrules are written
8.    File fTransformationRules = new File("T2TransformationRules.xsl");
9.
10.   // this is the ouput file of the tranformation
11.   File fResult = new File("FormDataSpecializeFM"+SelectedFM+".xml");
12.
13.   StreamSource FM1= new StreamSource(fInput);
14.   StreamSource XSLspecialize = new StreamSource(fTransformationRules);
15.   StreamResult FM2 = new StreamResult(fResult);
16.
17.   TransformerFactory factory = TransformerFactory.newInstance();
18.   Transformer T2 = factory.newTransformer(XSLspecialize);
19.   T2.transform(FM1, FM2); %>
```

Code 4.14: implementatie van de aanroep van de transformatie T_2

In Tabel 4.3 zijn de transformatieregels van de transformatie T_2 weergegeven. In de linker kolom zijn de namen van de transformatieregels weergegeven. In de middelste kolom zijn de condities weergegeven die geldig moeten zijn in het feature model FM_1. In de rechter kolom staan de transformatieacties die uitgevoerd worden wanneer de condities voor deze transformatieregel geldig zijn. Het resultaat van deze transformatieacties worden weggeschreven naar het feature model FM_2.

Transformatieregel	**Condities die geldig moeten zijn in het feature model FM_1 voor de transformatieacties**	**De transformatieacties van transformatie T_2**
TR_1	Als element ≠ *attribute* of element ≠ *GroupedFeature*	Kopieer element met attributen
TR_2	Als element = *attribute* en Attribute *value = string, integer, float* en ‘’ (leeg)	Creëer de bijbehorende elementen van het element attribute en zet de waarde erbij wat er in het XML bestand staat
TR_3	Als element = *GroupedFeature* ^ *GroupedFeature = “”* ^ *GroupedFeature = selected*	Kopieer de geselecteerde *GroupedFeature*
TR_4	Als element = *GroupedFeature* ^ *GroupedFeature ≠ “”*	kopieer alle *GroupedFeature* binnen dezelfde *FeatureGroup*

Tabel 4.3: transformatieregels voor de transformatie T_2

- ***TR_1***

  Als het element in het feature model FM_1 niet gelijk is aan *attribute* of *GroupedFeature*, dan wordt dit element samen met zijn attributen in het feature model FM_2 gekopieerd. De implementatie van deze transformatieregel is in Code 4.15 weergegeven.

```
1.    <xsl:template  match="*">
2.       <xsl:copy>
3.          <xsl:copy-of  select="node[name() != 'fm:Attribute' and
4.          name() != 'fm:GroupedFeature']"/>
5.          <xsl:copy-of select="@*"/>
6.          <xsl:apply-templates />
7.       </xsl:copy>
8.    </xsl:template>
```

Code 4.15: implementatie van de transformatieregel TR_1 van de transformatie T_2

- ***TR_2***

  Als het element in het feature model FM_1 een *attribute* is, dan wordt die element en de attributen van dit element opnieuw gecreëerd in het feature model FM_2. Daarbij wordt de waarde van attribuut *value* uit het XML bestand opgehaald met behulp van template *m*. Wanneer er geen waarde voor dit element in het XML bestand staat, dan wordt er een ‘’(lege) waarde opgestuurd. In Code 4.16 (zie volgende pagina) is de implementatie van deze transformatieregel weergegeven. Daarin is alleen voor het datatype *String* weergegeven. Voor elke datatype moet er een code aanwezig zijn zodat het gecreëerde datatype overeenkomt met het datatype dat in het feature model FM_1 voorkomt. De implementatie van template m is in Code 4.17 (zie volgende pagina) weergegeven.

```
1.    <xsl:template  match="fm:Attribute">
2.       <xsl:variable name="varPathName">
3.          <xsl:for-each select="ancestor-or-self::*">
4.             <xsl:text>-</xsl:text>
5.             <xsl:value-of select="@fm:value"/>
6.          </xsl:for-each>
7.       </xsl:variable>
8.       <xsl:variable name="varPathNameLength">
9.          <xsl:value-of select= "string-length($varPathName)"/>
10.      </xsl:variable>
11.      <xsl:variable name="varPathName2"select="substring
12.      ($varPathName,2,$varPathNameLength)"/>
13.      <xsl:choose>
14.         <xsl:when test="./fm:String/@fm:value='String'or
15.         ./fm:String/@fm:value=''" >
16.            <xsl:element name="fm:String">
17.               <xsl:attribute name="fm:value">
18.                  <xsl:call-template name="m">
19.                     <xsl:with-param name="parPathName">
20.                        <xsl:value-of select="$varPathName2"/>
21.                     </xsl:with-param>
22.                  </xsl:call-template>
23.               </xsl:attribute>
24.            </xsl:element>
25.         </xsl:when>
26.         <xsl:otherwise></xsl:otherwise>
27.      </xsl:choose>
28.   </xsl:template>
```

Code 4.16: implementatie van de transformatieregel TR_2 van de transformatie T_2

```
1.    <xsl:template mode="m" name="m" match="//dataoutput">
2.       <xsl:param name="parPathName"/>
3.       <xsl:variable name="docA"  select="document('dataoutput.xml')"/>
4.       <xsl:variable name="searchText">
5.          <xsl:value-of select="$parPathName"/>
6.       </xsl:variable>
7.       <xsl:value-of select="$docA//*[name(.)=$searchText]"/>
8.    </xsl:template>
```

Code 4.17: implementatie van de template m van de transformatieregel TR_2 van de transformatie T_2

- ***TR_3 en TR_4***

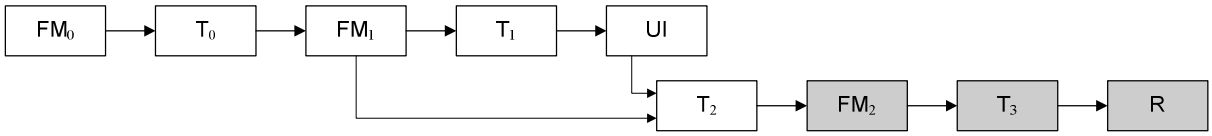
Als het element van het feature model FM_1 een *GroupedFeature* is, dan wordt er gekeken in het dataoutput bestand, met behulp van template *m* (zie Code 4.17), of er een keuze is gemaakt tussen de GroupedFeatures. Zodra er een keuze is gemaakt wordt de geselecteerde GroupedFeature in het feature model FM_2 gekopieerd. Anders worden alle GroupedFeatures binnen deze FeatureGroup in het feature model FM_2 gekopieerd. In Code 4.18 (zie volgende pagina) is de implementatie van deze transformatieregel TR_3 van de transformatie T_2 weergegeven. De condities voor de transformatieregel TR_3 staan in regel 1, 21 en 22 en de transformatieactie van deze transformatieregel staat in regel 23. De condities voor de transformatieregel TR_4 staan in regel 1, 21 en 26 en de transformatieactie van deze transformatieregel staat in regel 27.

```
1.    <xsl:template match="fm:GroupedFeature">
2.       <xsl:variable name="varPathName">
3.          <xsl:for-each select="ancestor::*">
4.             <xsl:text>-</xsl:text>
5.             <xsl:value-of select="@fm:value"/>
6.          </xsl:for-each>
7.       </xsl:variable>
8.       <xsl:variable name="varPathNameLength">
9.          <xsl:value-of select= "string-length($varPathName)"/>
10.      </xsl:variable>
11.      <xsl:variable name="varPathName2"select="substring
12.      ($varPathName,2,$varPathNameLength)"/>
13.      <xsl:variable name="varGetValue">
14.         <xsl:call-template name="m">
15.            <xsl:with-param name="parPathName">
16.               <xsl:value-of select="$varPathName2"/>
17.            </xsl:with-param>
18.         </xsl:call-template>
19.      </xsl:variable>
20.      <xsl:choose>
21.         <xsl:when test="$varGetValue != '' " >
22.            <xsl:if test="contains($varGetValue,@fm:value)">
23.               <xsl:copy-of  select="."/>
24.            </xsl:if>
25.         </xsl:when>
26.         <xsl:otherwise>
27.            <xsl:copy-of  select="."/>
28.         </xsl:otherwise>
29.      </xsl:choose>
30.   </xsl:template>
```

Code 4.18: implementatie van de transformatieregels TR_3 en TR_4 van de transformatie T_2

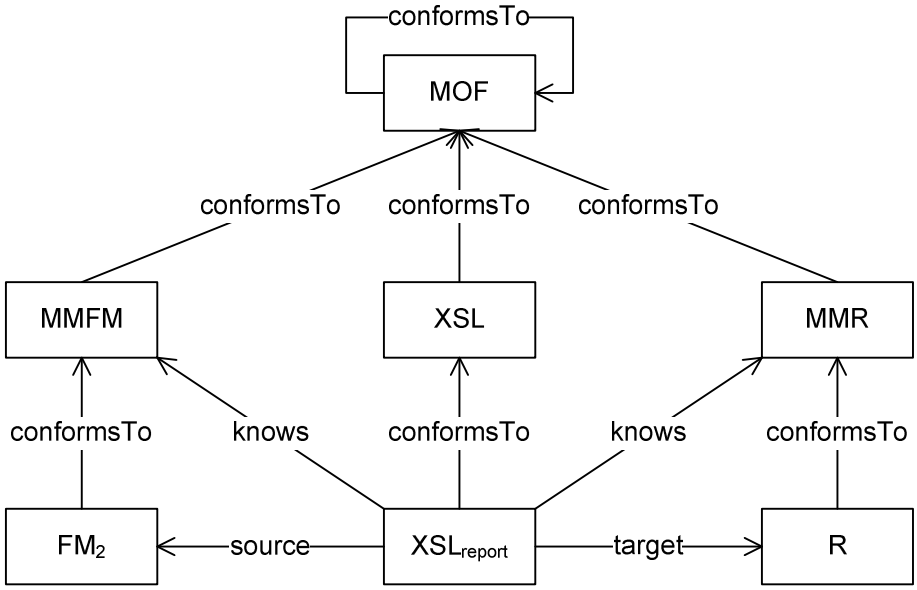
4.7 Processtap: generate report of selected product

Nadat de ingevulde gegevens van de burger weggeschreven zijn naar het feature model, gaat het proces verder met de processtap: *generate report of selected product* die in JSP pagina (*Send Report*) is geïmplementeerd. Alle aanvraaggegevens zullen verzonden worden naar de eMAXX Mid Office die voor het verder afhandelen van de aanvraag zorgt. De eMAXX Mid Office heeft een ander interface dan het feature model waardoor er in deze processtap het feature model omgezet wordt naar deze interface. De interfaces van de eMAXX Mid Office zijn eerder door eMAXX opgesteld. In dit onderzoek zijn de feature modellen afgeleid uit deze interfaces. In sectie 4.3.3 is deze afleiding gegeven waarin beschreven is hoe de feature modellen aan de hand van (eerder opgestelde) XML schema's worden opgesteld. Het omzetten van het feature model naar een rapport gaat met behulp van transformatie T_3 die in Figuur 4.25 te zien is waarbij de componenten die hier een rol spelen een donker achtergrondkleur hebben.



Figuur 4.25: de transformatie T_3 waarbij het rapport R van het geselecteerde product wordt gegenereerd

In Figuur 4.26 (zie volgende pagina) is het transformatie overzicht weergegeven voor de transformatie T_3 die een feature model FM_2 naar een rapport R transformeert. In rapport R zitten alle benodigde aanvraaggegevens om de desbetreffende aanvraag in te dienen. Rapport R correspondeert met het metamodel MMR (dat een XML schema is, die eerder door eMAXX is opgesteld). De transformatieregels voor deze transformatie T_3 staan in het XSL_{report} bestand.



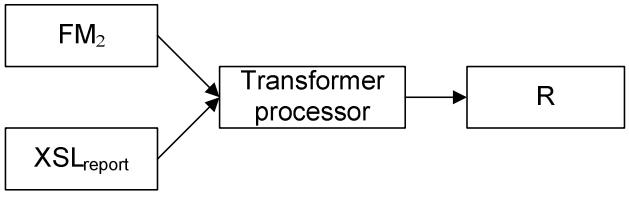
Figuur 4.26: het transformatie overzicht voor de transformatie T_3

In sectie 4.3.3 is het XML schema van het product "*doorgeven verhuizing*" als voorbeeld weergegeven. Elke product heeft zo zijn eigen XML schema. Aangezien de feature modellen vanuit deze XML schema's zijn opgesteld, kunnen deze feature modellen ook terug omgezet worden naar instanties van deze XML schema's. Deze instanties zijn de rapporten die naar de eMAXX Mid Office worden verzonden. In de volgende sectie wordt de implementatie van de processtap besproken.

4.7.1 Implementatie van de processtap

In deze sectie zal de implementatie van de processtap worden besproken. Ten eerste zal het concept van de transformatie T_3 worden besproken. Daarna zal de aanroep van de transformatie T_3 worden besproken, Ten slotte zullen de transformatieregels van de transformatie T_3 worden besproken.

In Figuur 4.27 is de invoer en de uitvoer van de transformer weergegeven. Er wordt een transformer geïnitieerd waarbij het XSL_{report} bestand aan de transformer wordt gekoppeld. Vervolgens wordt een aanroep gedaan om een transformatie uit te voeren. Bij de aanroep wordt een invoer en een uitvoer bestand aangegeven. Het invoer bestand is het FM_2 bestand waarin het feature model van het doorgeven van verhuizing staat. De uitvoer is het R bestand waarin het resultaat van de transformatie staat.



Figuur 4.27: invoer en uitvoer bestanden van de transformer om rapport R te genereren

In Code 4.19 is de implementatie van de aanroep van de transformatie T_3 in de JSP pagina (*JSP Send Report*) weergegeven. De JSP pagina haalt het gespecialiseerde feature model voor het doorgeven van een verhuizing op. Daarnaast wordt het XSL_{report} bestand opgehaald waarin de transformatieregels staan. Vervolgens wordt de transformatie T_3 uitgevoerd waarbij het feature model FM_2 naar rapport R wordt getransformeerd.

```
1.    <%// dit zorgt ervoor dat rapport R wordt gegenereerd.
2.
3.    // this is the input file of the tranformation
4.    File fInput = new File("SelectedFM+".xml");
5.
6.    // this is the file were the transformationrules are written
7.    File fTransformationRules = new File("T3TransformationRules.xsl");
8.
9.    // this is the ouput file of the tranformation
10.   File fResult = new File("Report"+SelectedFM+".xml");
11.
12.   StreamSource FM2= new StreamSource(fInput);
13.   StreamSource XSLreport = new StreamSource(fTransformationRules);
14.   StreamResult R = new StreamResult(fResult);
15.
16.   TransformerFactory factory = TransformerFactory.newInstance();
17.   Transformer T3 = factory.newTransformer(XSLreport);
18.   T3.transform(FM2, R); %>
```

Code 4.19: implementatie van de aanroep van de transformatie T_3

In Tabel 4.4 zijn de transformatieregels van de transformatie T_3 weergegeven. Het resultaat van deze transformatieacties wordt weggeschreven naar het rapport R.

Transformatieregel	Condities die geldig moeten zijn in het feature model FM_2 voor de transformatieacties	De transformatieacties van transformatie T_3
TR_1	Als element = RootFeature	Creëer element met de naam het attribuut fm:value van het feature.
TR_2	Als element = SolitaryFeature ^ ./fm:FeatureGroup/@fm:value = 'FeatureGroup'	Creëer element met de naam het attribuut fm:value van het SolitaryFeature. Het attribuut van GroupedFeature is de waarde van dit element.
TR_3	Als element = SolitaryFeature ^ ./fm:FeatureGroup/@fm:value != 'FeatureGroup'	Creëer element met de naam het attribuut fm:value van het SolitaryFeature. De waarde van dit gecreëerde element is de waarde van het attribuut *fm:value* van de child element van de child element Attibute van het element.

Tabel 4.4: transformatieregels van de transformatie T_3

- ***TR_1***

  Als *element = RootFeature*, dan wordt een element gecreëerd met de naam afkomstig van de waarde van het attribuut *fm:value* van het *RootFeature*.

- ***TR_2***

  Als *element = SolitaryFeature* en *./fm:FeatureGroup/@fm:value = 'FeatureGroup'*, dan wordt een element gecreëerd met de naam afkomstig van de waarde van het attribuut *fm:value* van het *SolitaryFeature*. De waarde van het attribuut *fm:value* van *GroupedFeature* van deze *FeatureGroup* is de waarde van dit gecreëerde element.

- ***TR_3***

  Als *element = SolitaryFeature* en *./fm:FeatureGroup/@fm:value = 'FeatureGroup'*, dan wordt een element gecreëerd met de naam afkomstig van de waarde van het attribuut *fm:value* van het *SolitaryFeature*. De waarde van dit gecreëerde element is de waarde van het attribuut *fm:value* van de child element van de child element Attibute van het element.

In Code 4.20 is het XSL_{report} bestand weergegeven voor het genereren van rapport R. De conditie voor de transformatieregel TR_1 staat in regel 1 en de transformatieactie van deze transformatieregel staat in regel 2 t/m 7. De condities voor de transformatie regel TR_2 staan in regel 9 en in regel 14 en de transformatieactie van deze transformatieregel staat in regel 15 t/m 18. De condities voor de transformatie regel TR_3 staan in regel 9, regel 14 en in regel 20 en de transformatieactie van deze transformatieregel staat in regel 21 t/m 24.

```
1.    <xsl:template match="fm:RootFeature">
2.          <xsl:variable name="nameElement">
3.             <xsl:value-of select="@fm:value"/>
4.          </xsl:variable>
5.          <xsl:element name="{@fm:value}">
6.             <xsl:apply-templates/>
7.          </xsl:element>
8.    </xsl:template>
9.    <xsl:template match="fm:SolitaryFeature">
10.      <xsl:variable name="nameElement2">
11.         <xsl:value-of select="@fm:value"/>
12.      </xsl:variable>
13.      <xsl:choose>
14.         <xsl:when test="./fm:FeatureGroup/@fm:value = 'FeatureGroup'">
15.            <xsl:element name="{$nameElement2}">
16.               <xsl:value-of select="./fm:FeatureGroup/fm:GroupedFeature/@fm:value"/>
17.               <xsl:apply-templates/>
18.            </xsl:element>
19.         </xsl:when>
20.         <xsl:otherwise>
21.            <xsl:element name="{$nameElement2}">
22.               <xsl:value-of select="./fm:Attribute/*/@fm:value"/>
23.               <xsl:apply-templates/>
24.            </xsl:element>
25.         </xsl:otherwise>
26.      </xsl:choose>
27.   </xsl:template>
```

Code 4.20: implementatie van de transformatieregels van de transformatie T3

4.8 Processtap: send report of selected product to Mid Office

Nadat het rapport is gegenereerd, wordt deze verzonden naar de eMAXX Mid Office. De communicatie met de eMAXX Mid Office gaat met behulp van SOAP.

Het SOAP-protocol bestaat uit drie onderdelen:

- Een envelop die een framework definieert voor het beschrijven van wat in een bericht staat en hoe het te verwerken.
- Een set van encodeerregels voor de expressie van 'instances' van applicatiegedefinieerde datatypen.
- Een conventie voor de representatie van 'remote procedure calls' en antwoorden. SOAP kan in potentie worden gebruikt in combinatie met een grote verscheidenheid aan andere protocollen.

Om e-commerce te kunnen bedrijven is het noodzakelijk dat verschillende bedrijven, hun systemen en hun applicaties met elkaar kunnen communiceren, onafhankelijk van besturingsysteem, programmeertaal en objectmodel. Idealiter zou een onderneming geen rekening hoeven moeten houden met de gebruikte technologieën van de andere. SOAP is het protocol dat in systeemonafhankelijkheid voorziet - onafhankelijkheid van taal, technologie, device en technische implementatie [Wik06c].

De implementatie van de SOAP aanroep in pseudo code is in Code 4.21 weergegeven. Het antwoord van de aanroep wordt eerst als string opgeslagen (zie regel 7) en vervolgens wordt die string weggeschreven naar het responseOfCall bestand (zie regel 9).

```
1.     public class SOAPCall
2.    {
3.       Public static String invoke(String xmldoc, StringcallPropertiesID, String
4.       callPropeertiesUrl, String username, String password, String domain, String
5.       correlationId )throws IOException, SAXException
6.       {
7.          String responceMessageStrGevuld = call(soapenvelopeGevuld,
8.          callPropertiesID, callPropertiesUrl);
9.          writeStringToFile( "..\\responseOfCall.xml", responceMessageStrGevuld);
10.      }
11.
12.      private static String call( String message, String soapAction, String url ) throws
13.      IOException
14.     {
15.         Connection = new Connection();
16.         Connection.Open();
17.         Connection.sendMessage();
18.         String responseMessage = Connection.getResponseBodyAsString ();
19.         Connection.Close();
20.         Return responseMessage;
21.      }
22.   }
```

Code 4.21: implementatie van de SOAP aanroep in pseudo code

In het resultaat staat dat het met succes is verstuurd of een foutmelding. Vervolgens wordt de burger geïnformeerd dat het e-formulier met succes is verstuurd of niet. Dit is tevens het einde van het hele proces.

5 Model gedreven e-formulier generator met ondersteuning van eMAXX Mid Office

In dit hoofdstuk zal de model gedreven e-formulier generator met ondersteuning van de eMAXX Mid Office worden ontworpen. Deze generator is een uitbreiding van de model gedreven e-formulier generator zonder interactie, die in hoofdstuk 4 is besproken, en daarom zal in dit hoofdstuk alleen de uitbreidingen op die generator worden beschreven.

In sectie 5.1 wordt het overzicht van de generatie proces besproken. In de daarop volgende secties worden de processtappen, die zijn toegevoegd voor deze uitbreiding, besproken. De overige processtappen zijn al in hoofdstuk 4 behandeld. In sectie 5.2 wordt de processtap: *get Applicant id* besproken. In sectie 5.3 wordt de processtap: *interaction with eMAXX Mid Office* besproken. In sectie 5.4 wordt de processtap: *specialize selected feature model by means of earlier saved data* besproken.

5.1 Overzicht generatie proces

In deze sectie wordt het generatie proces met de model gedreven e-formulier generator met ondersteuning de eMAXX Mid Office weergegeven. Door gebruik te maken van een eMAXX Mid Office functie zal een deel van het selectieprobleem dat in sectie 2.2.2 staat worden opgelost. Een deel van het selectieprobleem was dat het overbodig is om de gegevens, die opgeslagen zijn in een back office systeem, alsnog aan de burger te vragen. Nu worden deze opgeslagen gegevens opgehaald met behulp van een eMAXX Mid Office functie en worden deze gegevens niet aan de burger gevraagd.

Wanneer de opgehaalde gegevens niet volledig zijn of niet aanwezig zijn, dan worden deze ontbrekende gegevens alsnog aan de burger gevraagd. Zo wordt het probleem van het relatie met back office systemen (die in sectie 2.1.2 is weergegeven) opgelost.

Een functie van de eMAXX Mid Office is bijvoorbeeld "*getPersonDetails*". Hiermee kunnen de persoons- en adresgegevens worden opgehaald van een persoon als zijn A-nummer of BSN nummer bekend is. In dit hoofdstuk zal deze functie als voorbeeld dienen en zal alleen het BSN nummer gevraagd worden.

Om gebruik te maken van de eMAXX Mid Office functie "*getPersonDetails*" moeten er drie processtappen gebouwd worden.

- ***Get Applicant id***

  Deze processtap moet ervoor zorgen dat de benodigde BSN, die nodig is voor de functie "*getPersonDetails*", aan de burger wordt gevraagd.

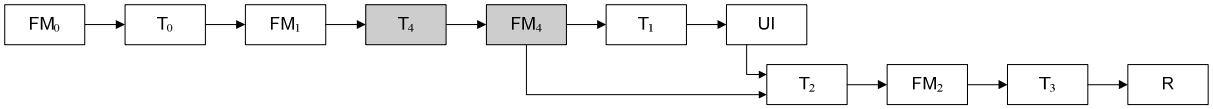
- ***Interaction with eMAXX Mid Office***

  De eMAXX Mid Office functie "*getPersonDetails*" wordt aangeroepen. Het resultaat wordt opgeslagen in het *responseOfCall* bestand.

- ***Specialize selected feature model by means of earlier saved data***

  De gegevens, die in *responseOfCall* bestand zijn opgeslagen, worden weggeschreven naar het feature model. Het wegschrijven van deze gegevens wordt met behulp van een transformatie gedaan.

Vergeleken met de generator in hoofdstuk **Fout! Verwijzingsbron niet gevonden.** wordt er in deze generator één transformatie meer uitgevoerd. In Figuur 5.1 zijn de transformaties van deze generator weergegeven. Deze figuur is een uitbreiding op Figuur 4.2. De transformatie en het feature model die er zijn bijgekomen zijn met donkere achtergrondkleur aangegeven in Figuur 5.1.



Figuur 5.1: transformaties van de model gedreven e-formulieren generator met ondersteuning van eMAXX Mid Office

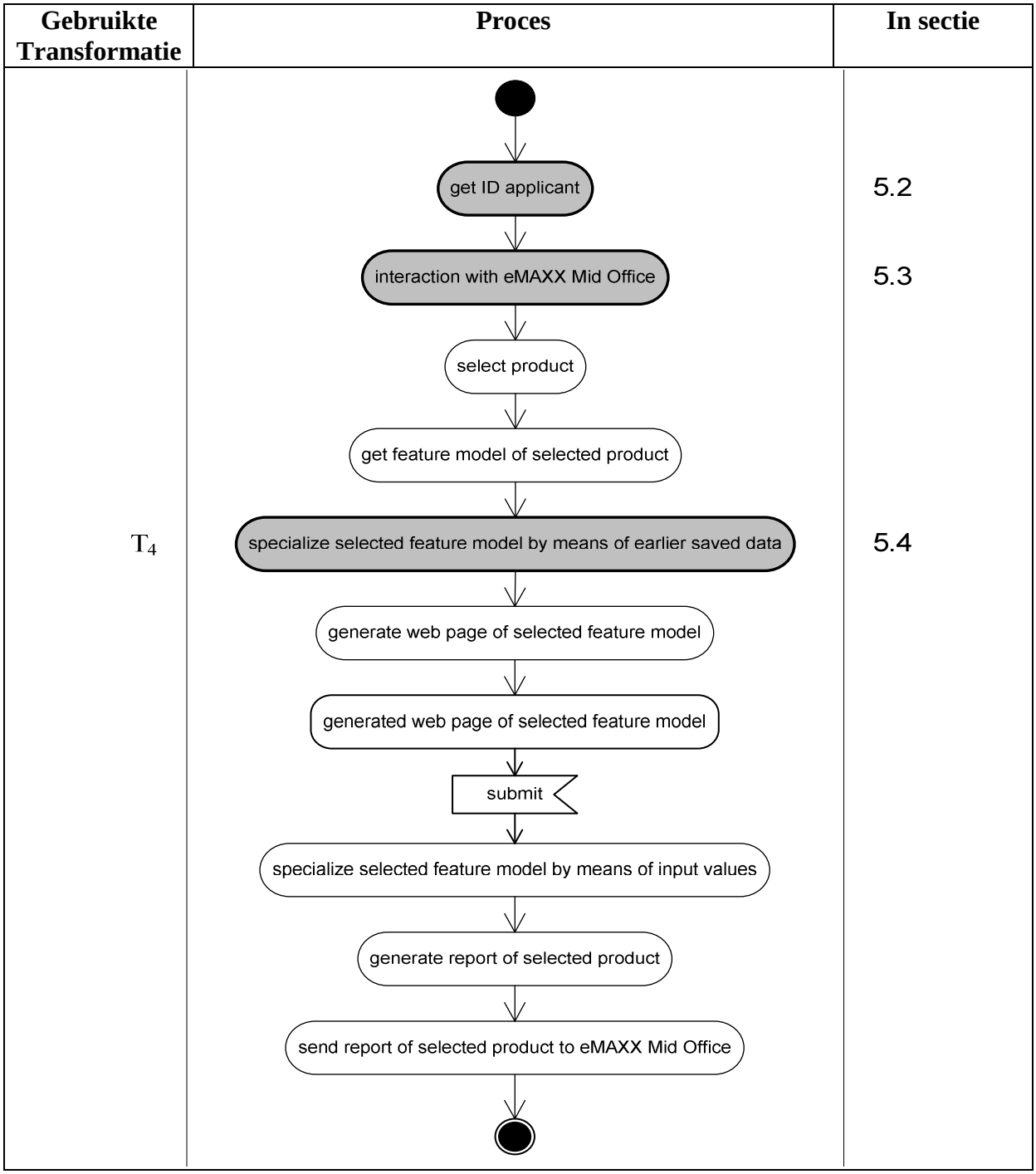
FM_0	=	basis feature model
T_0	=	transformatie voor het importeren van referentie feature modellen
FM_1	=	feature model met geïmporteerde referenties
T_4	**=**	**transformatie voor het specialiseren van het feature model**
FM_4	**=**	**feature model die deels gespecialiseerd is**
T_1	=	transformatie voor het genereren UI
UI	=	User Interface
T_2	=	transformatie voor het specialiseren van het feature model
FM_2	=	feature model die gespecialiseerd is
T_3	=	transformatie voor het genereren van een rapport
R	=	rapport dat naar de eMAXX Mid Office wordt verzonden

De transformatie T_4 transformeert het feature model FM_1 naar het feature model FM_4 met behulp van de waarden die opgehaald zijn via "*getPersonDetails*". FM_4 is dezelfde feature model als feature model FM_1 maar dit keer is het feature model (deels) gespecialiseerd.

De transformatie T_2 was een transformatie van het feature model FM_1 naar het feature model FM_2. Nu is deze transformatie een transformatie van het feature model FM_4 naar het feature model FM_2. De transformatieregels van de transformatie T_2 blijft ongewijzigd. Alleen de invoer van deze transformatie wordt gewijzigd.

De overige transformaties blijven hetzelfde als die in hoofdstuk 4. In dit hoofdstuk zal alleen de transformatie T_4 uitvoerig worden besproken.

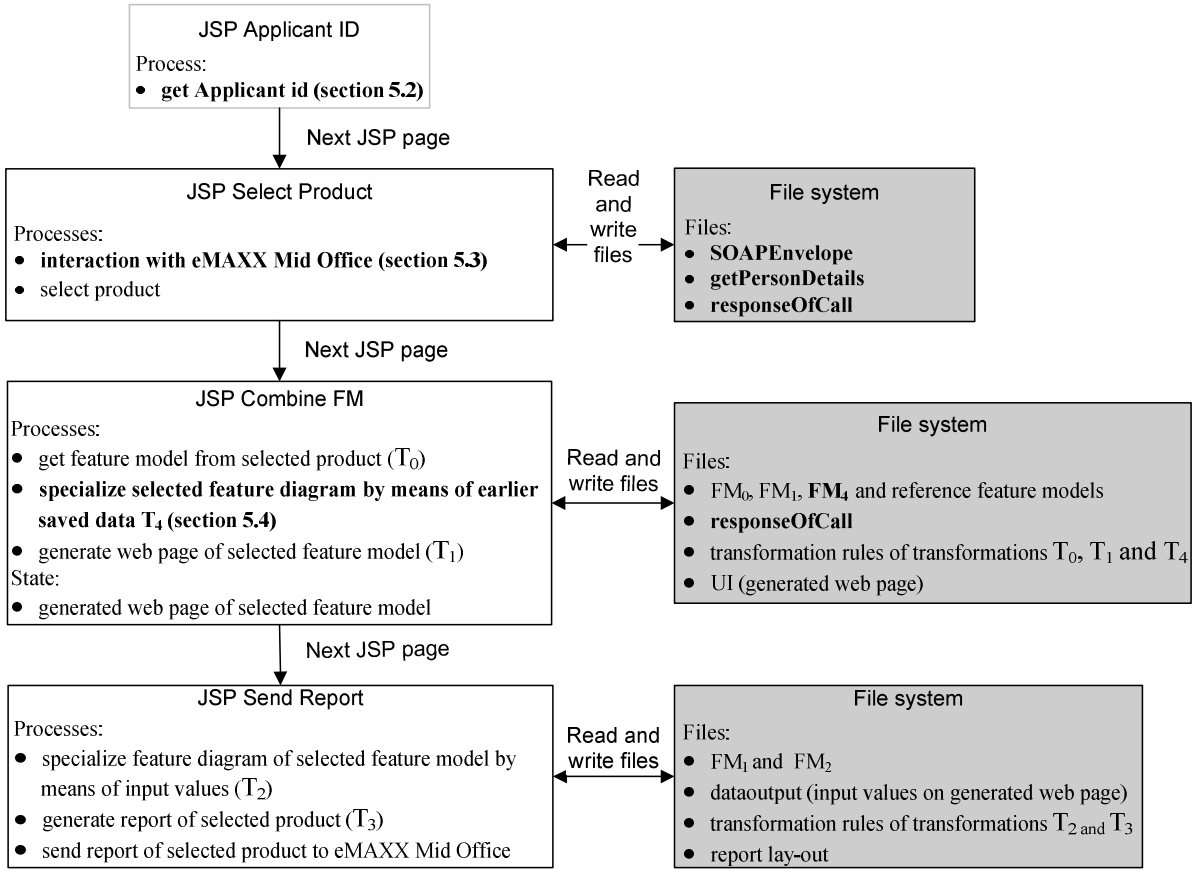
In Figuur 5.2 (zie volgende pagina) is het UML diagram van de e-formulier generator met ondersteuning van eMAXX Mid Office weergegeven. Dit is een uitbreiding van het UML diagram die in Figuur 4.3 is weergegeven. Het verschil tussen deze twee diagrammen is dat er nu drie extra processtapen zijn bijgekomen namelijk: *get Applicant id*, *interaction with eMAXX Mid Office* en *specialize selected feature model by means of earlier saved data*. Deze processtappen hebben een donkere achtergrondkleur in het Figuur 5.2 (zie volgende pagina).



Figuur 5.2: UML activiteitdiagram van de processtappen voor het invullen van een e-formulier met behulp van de model gedreven e-formulier generator met eMAXX Mid Office ondersteuning

De model gedreven e-formulier generator met ondersteuning van de eMAXX Mid Office is opgebouwd uit vier JSP pagina's (zie Figuur 5.3 op de volgende pagina). De Figuur 5.3 is een uitbreiding van de Figuur 4.4 waarin de uitbreidingen dik zijn gedrukt. In de eerste JSP pagina (*JSP Applicant ID*) wordt de processtap: *get Applicant id* uitgevoerd. In de tweede JSP pagina (*JSP Select Product*) worden de processtappen: *interaction with eMAXX Mid Office* en *select product* uitgevoerd. Deze JSP pagina haalt de benodigde bestanden van de webserver op. Het resultaat van de aanroep naar de eMAXX Mid Office wordt naar dezelfde webserver weggeschreven.

In de derde JSP pagina (*JSP Combine FM*) worden de processtappen: *get feature model of selected product, specialize selected feature model by means of earlier saved data* en *generate web page of selected feature model* uitgevoerd. Deze JSP pagina haalt de benodigde bestanden van de webserver op. De resultaten van de transformaties worden weggeschreven naar dezelfde webserver. De gegenereerde web pagina wordt in deze JSP pagina ingeladen waardoor het aanvraagproces in de toestand: *generated web page of selected feature model* terecht komt. In de laatste JSP pagina (*JSP Send Report*) worden de processtappen: *specialize selected feature model by means of input values, generate report of selected product* en *send report of selected product to eMAXX Mid Office* uitgevoerd. Deze JSP haalt ook de benodigde bestanden van dezelfde webserver op. De resultaten worden ook hier naar dezelfde webserver weggeschreven.



Figuur 5.3: overzicht JSP pagina's van de model gedreven e-formulier generator met ondersteuning eMAXX Mid Office

De burger gaat naar de webpagina van de desbetreffende gemeente om een aanvraag te doen. Het proces begint met het vragen de BSN (burgerservicenummer) van de burger. Nadat de burger zijn BSN heeft ingevuld, gaat de generator interactie met eMAXX Mid Office uitvoeren om de opgeslagen gegevens (bijvoorbeeld in een back office systeem) van de burger op te halen aan de hand van de ingevulde BSN. De opgehaalde gegevens worden op de webserver van de desbetreffende gemeente in een XML bestand opgeslagen.

Daarna gaat de burger een product selecteren. De generator haalt het bijbehorende feature model van het geselecteerde product van de webserver van de desbetreffende gemeente op. Vervolgens wordt dit feature model gespecialiseerd met de eerder op de webserver opgeslagen gegevens van de burger.

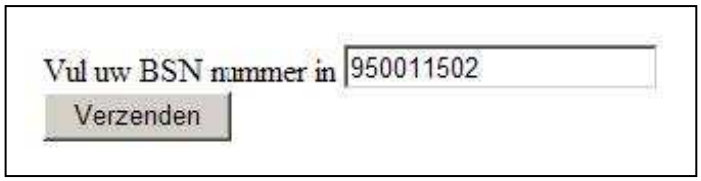
Vervolgens worden de features van het feature model waarvan de waarden nog ontbreken getransformeerd naar een webpagina om deze door de burger in te laten vullen. Nadat de burger de ontbrekende waarden heeft ingevuld en op "*verzend*" knop heeft gedrukt, wordt het feature model nog eens gespecialiseerd maar deze keer aan de hand van ingevulde waarden op de webpagina. Deze ingevulde waarden worden zoals eerder aangegeven eerst opgeslagen in een XML bestand en daarna pas gebruikt om het feature model te specialiseren via een transformatie.

Daarna gaat de generator het feature model transformeren naar het formaat van het rapport. Vervolgens wordt dit rapport verzonden naar het eMAXX Mid Office. Dit is tevens het einde van het hele proces.

In dit hoofdstuk zullen alleen de processtappen die een donkere achtergrondkleur hebben in Figuur 5.2 worden besproken. De overige processtappen zijn al in hoofdstuk 4 behandeld. In sectie 5.2 wordt de processtap: *get Applicant id* besproken. In sectie 5.3 wordt de processtap: *interaction with eMAXX Mid Office* besproken. In sectie 5.4 wordt de processtap: *specialize selected feature model by means of earlier saved data* besproken.

5.2 Processtap: get Applicant id

Het ophalen van de identiteit van de aanvrager wordt met behulp van een webpagina gedaan (*JSP Applicant ID*). Op de webpagina wordt aan de aanvrager gevraagd om zijn BSN in te vullen (zie Figuur 5.4). Aan de hand van dit BSN kunnen de persoons- en adresgegevens van de burger worden opgehaald die opgeslagen zijn. Deze webpagina is handmatig geïmplementeerd. Dit is gedaan om meteen te testen of het ophalen van persoonsgegevens goed gaat. In hoofdstuk 6 zal deze webpagina ook gegenereerd moeten worden door de generator wanneer dit nodig is.



Figuur 5.4: Webpagina voor het opvragen van de ID van de aanvrager

Nadat de aanvrager zijn BSN heeft ingevuld en op verzenden knop heeft gedrukt, wordt de ingevulde gegeven doorgestuurd naar de volgende JSP pagina (*JSP Select Product*) waarin dit gegeven worden gebruikt om de persoons- en adresgegeven van de aanvrager op te vragen (bijvoorbeeld uit een back office systeem) via de eMAXX Mid Office. Deze processtap wordt in de volgende sectie beschreven.

5.3 Processtap: interactie met eMAXX Mid Office

Nadat de burger zijn BSN heeft ingevuld en een product heeft geselecteerd, wordt het feature model van het geselecteerde product opgehaald zoals in sectie 4.3 is beschreven. Vervolgens wordt in deze processtap interactie met de eMAXX Mid Office uitgevoerd. In deze sectie zal deze interactie worden besproken. Daarbij zal ook de implementatie van dit proces worden weergegeven.

Een feature model moet niet alleen gespecialiseerd worden door de ingevulde waarden op een webpagina maar kan ook gespecialiseerd worden aan de hand van gegevens van de burgers die opgeslagen zijn en die via de eMAXX Mid Office worden opgehaald.

Voordat er een aanroep wordt gedaan, zal eerst het XML bericht klaar moeten zijn. Het XML bericht bestaat uit twee delen. De eerste deel is de envelop waarin het adres en de functie staan. De tweede deel is de body waarin de benodigde informatie voor de functie staat. In dit hoofdstuk wordt de functie "*getPersonDetails*" gebruikt die een BSN nodig heeft om uitgevoerd te kunnen worden. Het BSN van de vorige sectie wordt in de body van dit XML bericht opgeslagen. De implementatie van deze processtap in JSP pagina (*JSP Combine FM*) is Code 5.1 weergegeven. Daar waar de achtergrond donker is in de code, wordt het BSN opgehaald en weggeschreven in dit XML bericht.

```
1.    File fSoapEnvelope = new File("soapenvelopgevuld.xml");
2.    Write.fSoapEnvelope("
3.       <SOAP-ENV:Envelope>
4.          <SOAP-ENV:Body>
5.             <gba:getPersonDetails>
6.                <prop:requestProperties>
7.                   <prop:user>demo</prop:user>
8.                   <prop:password>demo</prop:password>
9.                   <prop:domain>MidOffice</prop:domain>
10.                  <prop:correlationId/>
11.               </prop:requestProperties>
12.               <gba:personDetailsRequestMessage>
13.                  <gba:persons>
14.                     <gba:person>
15.                        <gba:id xsi:nil=\"true\"/>
16.                        <gba:bsn>
17.                           request.getParameter("person_bsn")
18.                        </gba:bsn>
19.                     </gba:person>
20.                  </gba:persons>
21.               </gba:personDetailsRequestMessage>
22.            </gba:getPersonDetails>
23.         </SOAP-ENV:Body>
24.      </SOAP-ENV:Envelope>
25.   ");
```

Code 5.1: implementatie voor het opslaan van het verzonden nummer in XML bericht voor de aanroep naar eMAXX Mid Office

Er is een poging gedaan om een SOAP aanroep naar eMAXX Mid Office te doen in een XSL bestand. XSLT ondersteunt geen aanroepen naar buiten toe. Wel kan de XSLT processor worden uitgebreid met Java code. Dankzij deze uitbreiding is het alsnog mogelijk om in XSL bestand een aanroep naar buiten te doen. Deze Java code is geschreven en de aanroep werd in het XSL bestand gedaan. In een later stadium van het onderzoek zijn de processen aan elkaar gekoppeld met JSP pagina's. Vanuit een JSP pagina kan ook een SOAP aanroep worden gedaan naar eMAXX Mid Office. Doordat de processen via de JSP pagina's aan elkaar gekoppeld zijn, is de aanroep naar eMAXX Mid Office verschoven naar de JSP pagina.

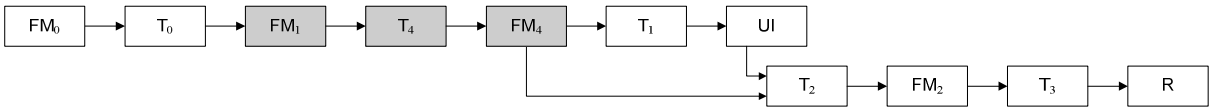
De implementatie van de SOAP aanroep in bijna hetzelfde als het versturen van het rapport naar de eMAXX Mid office die in Code 4.21 is weergegeven. Het antwoord op de aanroep wordt opgeslagen in *responseOfCall* bestand. Dit bestand bevat of gegevens van de aanvrager of een foutmelding. Bij een foutmelding kan de generator geen overeenkomende feature vinden en gaat het proces verder. Wanneer er wel overeenkomende features zijn, dan worden deze waarden weggeschreven naar het feature model. Een gedeelte van een voorbeeld bestand (indien wel goed gaat) is in Code 5.2 weergegeven.

```
1.    <gba:gba>
2.       <gba:basicInfo>
3.          <gba:aNumber>2301313497</gba:aNumber>
4.          <gba:bsn>950011502</gba:bsn>
5.          <gba:citizenNumber>950011502</gba:citizenNumber>
6.          <gba:citizenNumberSupplement/>
7.          <gba:dateOfBirth>19620000</gba:dateOfBirth>
8.          <gba:cityOfBirth>Leeuwarden</gba:cityOfBirth>
9.          <gba:municipalityOfBirth/>
10.         <gba:gender>v</gba:gender>
11.         <gba:lastNamePrefix/>
12.         <gba:initials>T</gba:initials>
13.         <gba:firstNames>Tina</gba:firstNames>
14.         <gba:lastName>Mol</gba:lastName>
15.      </gba:basicInfo>
16.   </gba:gba>
```

Code 5.2: een gedeelte van het responseOfCall bestand waarin het antwoord op de aanroep staat

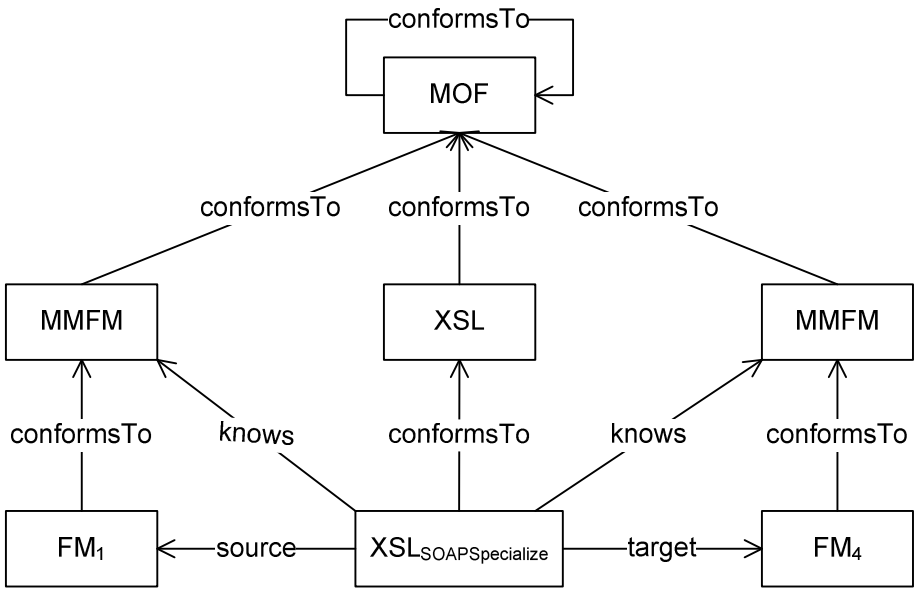
5.4 Processtap: specialize selected feature model by means of earlier saved data

Nadat er een interactie heeft plaatsgevonden met de eMAXX Mid Office, wordt in dezelfde JSP pagina (*JSP Combine FM*) de processtap: *specialize selected feature model by means of earlier saved data* uitgevoerd. In deze processtap worden de opgehaalde gegevens weggeschreven naar het feature model. In deze sectie zal het specialiseren van het feature model aan de hand van eerder opgeslagen gegevens worden besproken. Het specialiseren gaat met behulp van de transformatie T_4, die samen met de componenten die een rol spelen in Figuur 5.5 met een donkere achtergrondkleur is weergegeven.



Figuur 5.5: de transformatie T_4 waarbij het feature model FM_1 wordt gespecialiseerd

Zoals eerder aangegeven wordt het transformatie overzicht, die in sectie 3.1.3 is weergegeven, gebruikt voor de transformaties. In Figuur 5.6 (zie volgende pagina) is het transformatie overzicht weergegeven voor transformatie T_4 die het feature model FM_1 naar het feature model FM_4 transformeert. De invoer van de transformatie is het feature model FM_1. De uitvoer van de transformatie is het feature model FM_4. Om een transformatie uit te voeren moeten de transformatieregels worden opgesteld. De transformatieregels voor deze transformatie T_4 staan in het $XSL_{SOAPSpecialize}$ bestand.



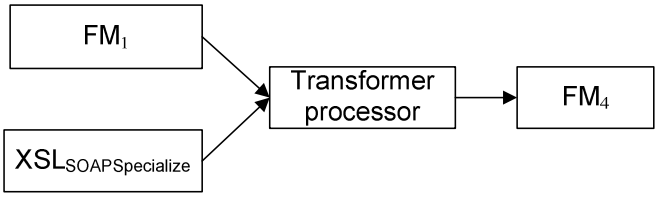
Figuur 5.6: het transformatie overzicht voor de transformatie T_4

In sectie 5.4.1 wordt de implementatie van de processtap weergegeven.

5.4.1 Implementatie van de processtap

In deze sectie zal de implementatie van de processtap worden weergegeven. Allereerst zal het concept van de transformatie worden besproken. Vervolgens zal de aanroep van de transformatie worden besproken. Ten slotte zullen de transformatieregels worden besproken.

In Figuur 5.7 is de invoer en de uitvoer van de transformer weergegeven. Er wordt een transformer geïnitieerd waarbij het $XSL_{SOAPSpecialize}$ bestand aan de transformer wordt gekoppeld. Vervolgens wordt een aanroep gedaan om een transformatie uit te voeren. Bij de aanroep wordt een invoer en een uitvoer bestand aangegeven. De invoer bestand is het FM_1 bestand waarin het feature model FM_1 staat. De output is het feature model FM_4 waarbij een deel van het feature model is gespecialiseerd.



Figuur 5.7: invoer en uitvoer bestanden van de transformer om het feature model FM_1 te specialiseren

In Code 5.3 (zie volgende pagina) is de implementatie van de aanroep van de transformatie T_4 weergegeven in de JSP pagina. De JSP pagina haalt het feature model op van het doorgeven van verhuizing waarbij de referentie features zijn geïmporteerd. Dit is het invoer bestand voor de transformer. Daarnaast wordt het $XSL_{SOAPSpecialize}$ bestand opgehaald waarin de transformatieregels staan voor de transformatie T_4. Vervolgens wordt de transformatie T_4 uitgevoerd waarbij het feature model FM_1 naar het feature model FM_4 wordt getransformeerd.

```
1.    //this is the input file of the tranformation
2.    File fInput = new File("FormDataSpecializeFD"+SelectedFD+".xml");
3.
4.    // this is the file were the transformationrules are written
5.    File fTransformationRules = new File("soapResponseToFD.xsl");
6.
7.    // this is the ouput file of the tranformation
8.    File fResult = new File("SoapSpecializeFD"+SelectedFD+".xml");
9.
10.   StreamSource  FM1 = new StreamSource(fInput);
11.   StreamSource   XSLSOAPSpecialize = new StreamSource(fTransformationRules);
12.   StreamResult  FM4 = new StreamResult(fResult);
13.
14.   TransformerFactory factory2 = TransformerFactory.newInstance();
15.   Transformer  T4 = factory2.newTransformer( XSLSOAPSpecialize );
16.   T4.transform( FM1 , FM4 );
```

Code 5.3: implementatie van de aanroep van de transformatie T_4

In Tabel 5.1 zijn de transformatieregels van de transformatie T_4 weergegeven. In de linker kolom zijn de namen van de transformatieregels weergegeven. In de middelste kolom zijn de condities weergegeven wat geldig moet zijn in het feature model FM_1. In de rechter kolom staan de transformatieacties die uitgevoerd worden wanneer de condities voor deze transformatieregel geldig zijn. Het resultaat van deze transformatieacties worden weggeschreven naar het feature model FM_4.

Transfor-matieregel	**Condities die geldig moeten zijn in het feature model FM_1 voor de transformatieacties**	**De transformatieacties van transformatie T_4**
TR_1	Als element ≠ *fm:String* and element ≠ *fm:Integer and* element ≠ *fm:Float and* element ≠ *fm:GroupedFeature*	Kopieer element met attributen
TR_2	Als element = *fm:String* of element = *fm:Integer of* element = *fm:Float*	Creëer dit elemtn en zet de waarde erbij wat er in het XML bestand staat
TR_3	Als element = *GroupedFeature* ^ *GroupedFeature* ≠ “” ^ *GroupedFeature = selected*	Kopieer de geselecteerde *GroupedFeature*
TR_4	Als element = *GroupedFeature* ^ *GroupedFeature* = “”	kopieer alle *GroupedFeature* binnen dezelfde *FeatureGroup*

Tabel 5.1: transformatieregels voor de transformatie T_4

- ***TR_1***

  Als het element in het feature model FM_1 niet gelijk is aan *fm:String* en *fm:Integer* en *fm:Float* en *fm:GroupedFeature*, dan wordt dit element samen met zijn attributen in het feature model FM_4 gekopieerd. De implementatie van deze transformatieregel is in Code 5.4 (zie volgende pagina) weergegeven

```
1.    <xsl:template  match="*">
2.       <xsl:copy>
3.          <xsl:copy-of  select="node[name() != 'fm:String' and name() !=
4.          'fm:Integer' and  name() != 'fm:Float' and name() !=
5.          fm:GroupedFeature']"/>
6.          <xsl:copy-of select="@*"/>
7.          <xsl:apply-templates />
8.       </xsl:copy>
9.    </xsl:template>
```

Code 5.4: implementatie van de transformatieregel TR_1 van de transformatie T_4

- ***TR_2***

Als het element in het feature model FM_1 gelijk is aan *fm:String* of *fm:Integer* of *fm:Float*, dan wordt de elementen en de attributen van dit element opnieuw gecreëerd in het feature model FM_4. Daarbij wordt de waarde van attribuut *fm:value* uit het XML bestand opgehaald met behulp van template *m*. Wanneer er geen waarde voor dit element in het XML bestand staat, dan wordt er een ‘’(lege) waarde opgestuurd. In Code 5.5 is de implementatie van deze transformatieregel weergegeven. Daarin is alleen voor het datatype: *String* weergegeven. Voor elke datatype moet een aparte code aanwezig zijn zodat het gecreëerde datatype overeenkomt met het datatype die in het feature model FM_1 voorkomt. De implementatie van template *m* is in Code 5.6 (zie volgende pagina) weergegeven.

```
1.    <xsl:template  match="fm:String">
2.       <xsl:variable name="varSF" select="../../@fm:value"/>
3.       <xsl:variable name="varPSF" select="../../../@fm:value"/>
4.       <xsl:element name="fm:String">
5.          <xsl:attribute name="fm:value">
6.             <xsl:call-template name="m">
7.                <xsl:with-param name="parSF">
8.                   <xsl:value-of select="$varSF"/>
9.                </xsl:with-param>
10.               <xsl:with-param name="parPSF">
11.                  <xsl:value-of select="$varPSF"/>
12.               </xsl:with-param>
13.            </xsl:call-template>
14.         </xsl:attribute>
15.      </xsl:element>
16.   </xsl:template>
```

Code 5.5: implementatie van de transformatieregel TR_2 van de transformatie T_4 voor het String element

```
1.    <xsl:template mode="m" name="m">
2.       <xsl:param name="parSF"/>
3.       <xsl:param name="parPSF"/>
4.       <xsl:variable name="docA" select="document
5.       ('debugresponsegevuldenvelop2.xml')"/>
6.       <xsl:variable name="searchText">
7.          gba:<xsl:value-of select="$parSF"/>
8.       </xsl:variable>
9.       <xsl:variable name="search[Parent]">
10.         gba:<xsl:value-of select="$parPSF"/>
11.      </xsl:variable>
12.      <xsl:choose>
13.         <xsl:when test="$parSF='name' or $parSF='id'">
14.            <xsl:for-each select="$docA//*[name(.)=$searchText]">
15.               <xsl:if test="[parent] = $search[Parent]">
16.                  <xsl:value-of select="."/>
17.               </xsl:if>
18.            </xsl:for-each>
19.         </xsl:when>
20.         <xsl:otherwise>
21.            <xsl:value-of select="$docA//*[name(.)=$searchText]"/>
22.         </xsl:otherwise>
23.      </xsl:choose>
24.   </xsl:template>
```

Code 5.6: implementatie van de template *m* van de transformatieregel TR_2 van de transformatie T_4

- ***TR_3 en TR_4***

Als het element van het feature model FM_1 een *GroupedFeature* is, dan wordt er gekeken in het XML bestand van de aangeroepen functie, met behulp van template *m* (zie Code 5.6), of er een waarde in het bestand staat. Als er een waarde in staat, die gelijk is aan de waarde. Als er een waarde staat, dan wordt die GroupedFeature in het feature model FM_4 gekopieerd. Anders worden alle GroupedFeatures binnen deze FeatureGroup in het feature model FM_4 gekopieerd. In Code 5.7 is de implementatie van deze transformatieregel TR_3 van de transformatie T_4 weergegeven. De condities voor de transformatieregel TR_3 staan in regel 1, 11 en 12 en de transformatieactie van deze transformatieregel staat in regel 11. De condities voor de transformatieregel TR_4 staan in regel 1, 11 en 16 en de transformatieactie van deze transformatieregel staat in regel 17.

```
1.    <xsl:template match="fm:GroupedFeature">
2.       <xsl:variable name="varSF" select="../../@fm:value"/>
3.       <xsl:variable name="varGetValue">
4.          <xsl:call-template name="m">
5.             <xsl:with-param name="parSF">
6.                <xsl:value-of select="$varSF"/>
7.             </xsl:with-param>
8.          </xsl:call-template>
9.       </xsl:variable>
10.      <xsl:choose>
11.         <xsl:when test="$varGetValue != '' " >
12.            <xsl:if test = " contains($varGetValue,@fm:value)">
13.               <xsl:copy-of  select="."/>
14.            </xsl:if>
15.         </xsl:when>
16.         <xsl:otherwise>
17.            <xsl:copy-of  select="."/>
18.         </xsl:otherwise>
19.      </xsl:choose>
20.   </xsl:template>
```

Code 5.7: implementatie van de transformatieregels TR_3 en TR_4 van de transformatie T_4

In de onderstaande Code 5.8 is een deel van het resultaat (het feature model FM_4) van deze transformatie weergegeven.

```
1.    <fm:SolitaryFeature fm:value="initials">
2.       <fm:Annotation fm:value="Annotation">
3.          <fm:Description fm:value="Aanhef"/>
4.       </fm:Annotation>
5.       <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
6.       <fm:Attribute fm:value="Attribute">
7.          <fm:String fm:value="T"/>
8.       </fm:Attribute>
9.    </fm:SolitaryFeature>
10.   <fm:SolitaryFeature fm:value="firstNames">
11.      <fm:Annotation fm:value="Annotation">
12.         <fm:Description fm:value="Voorletter(s)"/>
13.      </fm:Annotation>
14.      <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
15.      <fm:Attribute fm:value="Attribute">
16.         <fm:String fm:value="Tina"/>
17.      </fm:Attribute>
18.   </fm:SolitaryFeature>
19.   <fm:SolitaryFeature fm:value="middleName">
20.      <fm:Annotation fm:value="Annotation">
21.         <fm:Description fm:value="Tussenvoegsel"/>
22.      </fm:Annotation>
23.      <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
24.      <fm:Attribute fm:value="Attribute">
25.         <fm:String fm:value=""/>
26.      </fm:Attribute>
27.   </fm:SolitaryFeature>
```

Code 5.8: gedeelte van het feature model FM_4 na de transformatie T_4

6 Model gedreven e-formulier generator met workflow

In dit hoofdstuk zal de model gedreven e-formulier generator met workflow worden beschreven. Deze generator is een uitbreiding van de model gedreven e-formulier generator met ondersteuning van de eMAXX Mid Office, die in hoofdstuk 5 is besproken, en daarom zullen in dit hoofdstuk alleen de uitbreidingen op die generator worden beschreven.

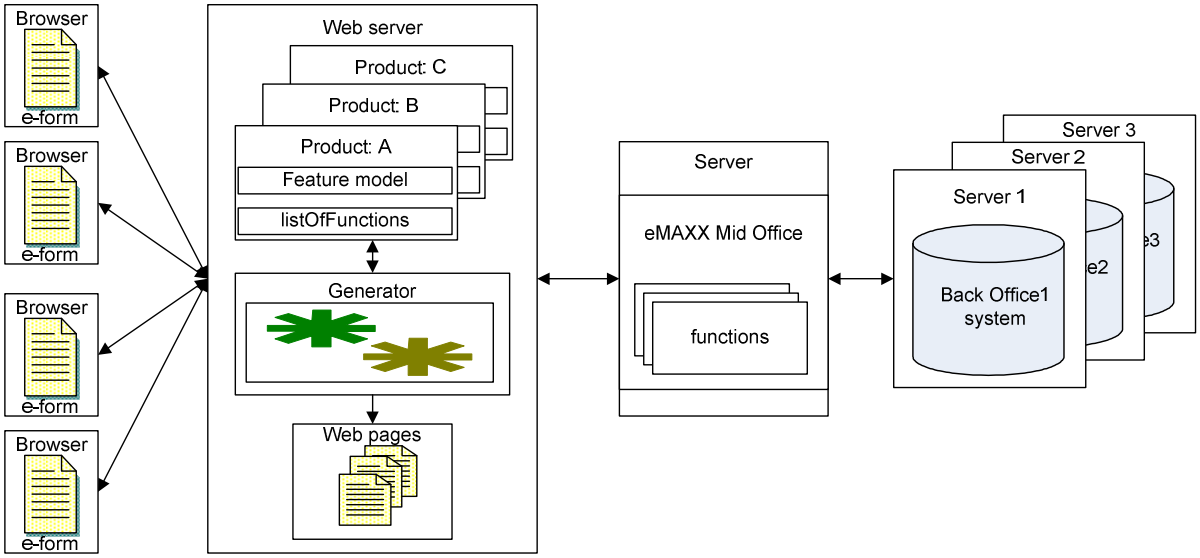
In sectie 6.1 wordt het overzicht van de generatie proces besproken. In sectie 6.2 wordt de processtap: *get product properties of selected product* besproken. In sectie 6.3 wordt de processtap: *select obligatory features* besproken. In sectie 6.4 wordt de processtap: *generate web page of selected obligatory features* besproken. In sectie 6.5 wordt de processtap: *specialize selected fm by means of input values* besproken. In sectie 6.6 wordt de processtap: *execute all possible functions* besproken. In sectie 6.7 wordt de processtap: *generate report of selected product* besproken.

6.1 Overzicht generatie proces

In deze sectie wordt het generatie proces beschreven. Door de benodigde gegevens te verspreiden over meerdere webpagina's (indien nodig), wordt het oriëntatieprobleem deels opgelost. Zoals eerder in sectie 2.2.1 is aangegeven, is er geen goed overzicht wanneer alle gegevens (als dat er veel zijn) op één webpagina worden gevraagd. Om een beter overzicht te krijgen worden deze gegevens over meerdere webpagina's verspreid.

De vorige twee generatoren werken alleen met het geselecteerde feature model. Deze generator gebruikt naast het geselecteerde feature model ook een lijst van functies, die de functies bevat die gebruikt kunnen worden bij het geselecteerde feature model. In deze generator zal meerdere keren (indien nodig) een functie worden aangeroepen terwijl in hoofdstuk 5 maar één keer een eMAXX Mid Office functie wordt aangeroepen. Deze functies zijn bedoelt om het feature model verder te specialiseren. Deze functies kunnen de functies van de eMAXX Mid Office zijn zoals getPersonDetails die in hoofdstuk 5 is gebruikt. Ook kunnen deze functies zelfgedefinieerde functies zijn waarmee het feature model verder wordt gespecialiseert zoals de "deduceS*alutationFromGender*" functie waarmee de feature *salutation* wordt gespecialiseert aan de hand van de feature *gender*.

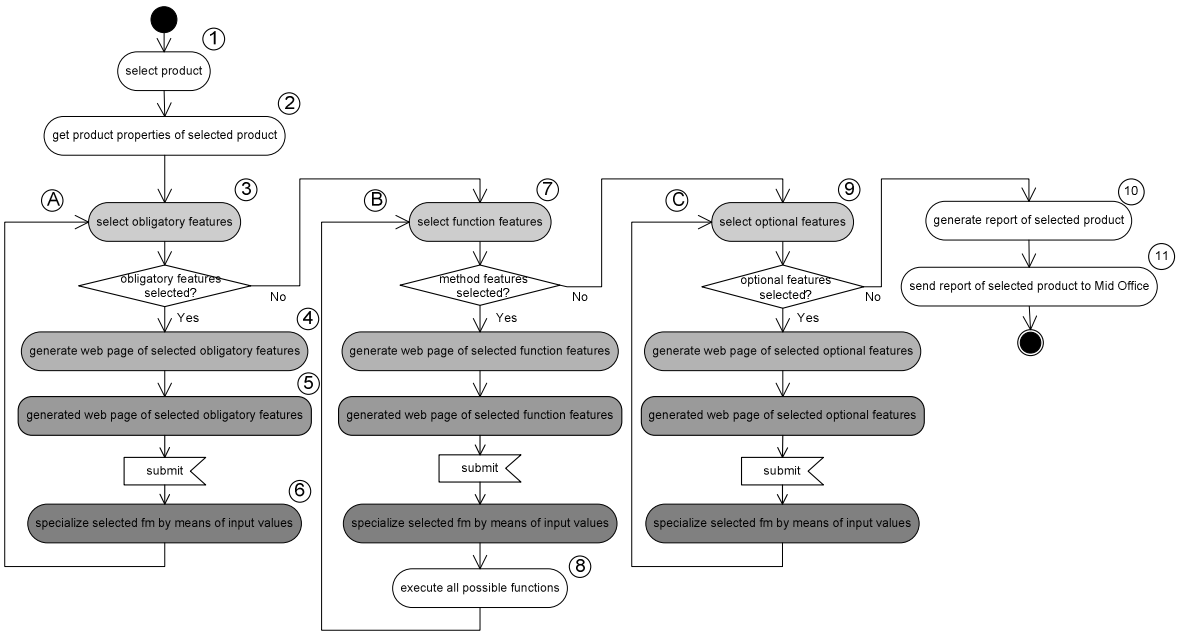
Aan de hand van deze twee producteigenschappen wordt bepaald wanneer er wat gebeurt in de generator. In Figuur 6.1 (zie volgende pagina) is een abstracte weergave van de communicatie van verschillende onderdelen met de generator van een willekeurige gemeente weergegeven. Dit figuur is bijna hetzelfde als Figuur 4.1. Het verschil is dat dit figuur niet alleen een feature model bevat maar ook een lijst van functies heeft.



Figuur 6.1: abstracte weergave van de communicatie van verschillende onderdelen met workflow uitbreiding van de generator van een gemeente

Alle transformaties die in hoofdstuk 5 zijn gebruikt, worden ook in dit hoofdstuk gebruikt. Alleen zijn de transformaties die in dit hoofdstuk worden gebruikt niet gekoppeld zoals in hoofdstuk 5 maar zijn deze losgekoppeld van elkaar. In de processtappen waar de transformaties plaatsvinden zullen de gebruikte transformaties worden weergegeven.

In Figuur 6.2 is het UML activiteitdiagram van de e-formulieren generator met workflow weergegeven. Dit is een uitbreiding van het diagram dat in Figuur 5.2 is weergegeven. De workflow kan beschreven worden als beweging van documenten en taken door een bedrijfsproces. Het workflow kan een opeenvolgende progressie van werk activiteiten of een complexe reeds processen zijn die elk gelijktijdig plaatsvinden. Uiteindelijk beïnvloeden ze elkaar volgens een reeks regels, routes en rollen [DiC97].



Figuur 6.2: UML activiteitdiagram van de processtappen voor het invullen van een e-formulier met behulp van de model gedreven e-formulier generator zonder interactie met workflow

De workflow van de eerste twee generatoren zijn rechttoe rechtaan processtappen die elkaar opvolgen. Hierbij maakt het niet uit wat de eindgebruiker invult. In de vorige generatoren worden de ingevulde waarden weggeschreven naar het feature model en vervolgens wordt een rapport gegenereerd van dit feature model. Er worden verder geen beslissingen genomen door de generatoren aan de hand van de ingevulde waarden. Daarentegen is het workflow van de deze laatste generator complex. In Figuur 6.2 is te zien dat er drie beslispunten zijn waarbij het (wel) uitmaakt wat de eindgebruiker invult. Aan de hand van wat de eindgebruiker heeft ingevuld wordt er bepaald wat er in de volgende stap gaat gebeuren. Wanneer de eindgebruiker bijvoorbeeld geen waarde invult voor verplichte velden, dan worden deze velden opnieuw gegenereerd om alsnog door de eindgebruiker in te laten.

In Figuur 6.2 zijn er drie lussen van processtappen te zien namelijk A, B en C. Deze lussen bestaan uit soortgelijke processtappen die bijna hetzelfde doen in vergelijkbare lussen zijn met dezelfde achtergrondkleur aangegeven en zullen in één sectie worden besproken. De verschillen tussen deze processtappen zullen in de desbetreffende sectie worden besproken.

Hieronder is het doorlopen van de processtappen beschreven.

1. Als eerst wordt een product geselecteerd door een eindgebruiker. Het proces gaat door met processtap 2.

2. In deze processtap worden de producteigenschappen opgehaald van het geselecteerde product. Dit zijn het feature model en de bijbehorende lijst van functies. Vervolgens gaat het proces verder met processtap 3.

3. Deze processtap is het begin van lus A. In deze processtap worden de features van het feature model geselecteerd die aan de eindgebruiker gevraagd moeten worden. Als er features zijn geselecteerd gaat het proces verder met processtap 4. Anders gaat het proces verder met processtap 7.

4. In deze processtap wordt een webpagina gegenereerd op basis van de geselecteerde features uit de processtap 3. Vervolgens komt het proces in toestand 5 terecht.

5. In deze toestand wordt de webpagina door de eindgebruiker ingevuld. Nadat deze is ingevuld en op de verzendknop wordt gedrukt, worden de ingevulde waarden naar de server gestuurd en het proces gaat verder met processtap 6.

6. In deze procestap wordt het feature model gespecialiseerd aan de hand van de ingevulde waarden op de webpagina. Vervolgens gaat het proces terug naar processtap 3.

7. Deze processtap is het begin van lus B. In deze processtap worden de features geselecteerd waarmee in de volgende stappen een functie aangeroepen kan worden. Deze functie is een eMAXX Mid Office functie waarmee interactie met de eMAXX Mid Office wordt uitgevoerd of een zelfgedefinieerde functie die het feature model rechtstreeks specialiseert. Wanneer er features zijn geselecteerd worden de processtappen vergelijkbaar met 3 t/m 6 doorlopen en gaat het proces vervolgens door naar processtap 8. Anders gaat het proces verder met processtap 9.

8. In deze processtap wordt het feature model gespecialiseerd door alle uitvoerbare functies uit het lijst van functies uit te voeren. Vervolgens gaat het proces terug naar processtap 7.

9. Deze processtap is het begin van lus C. In deze processtap worden features geselecteerd die optioneel zijn. Deze features worden aan de eindgebruiker gevraagd, maar de eindgebruiker hoeft deze features geen waarde toe te kennen. Wanneer er features zijn geselecteerd worden de processtappen vergelijkbaar met 3 t/m 6 doorlopen en gaat het proces terug naar processtap 9. Anders gaat het proces verder met processtap 10.

10. In deze processtap wordt een rapport gegenereerd van het volledige feature model. Dit rapport is nodig om alle verzamelde gegevens naar de eMAXX Mid Office te sturen. Het proces gaat verder met processtap punt 11.

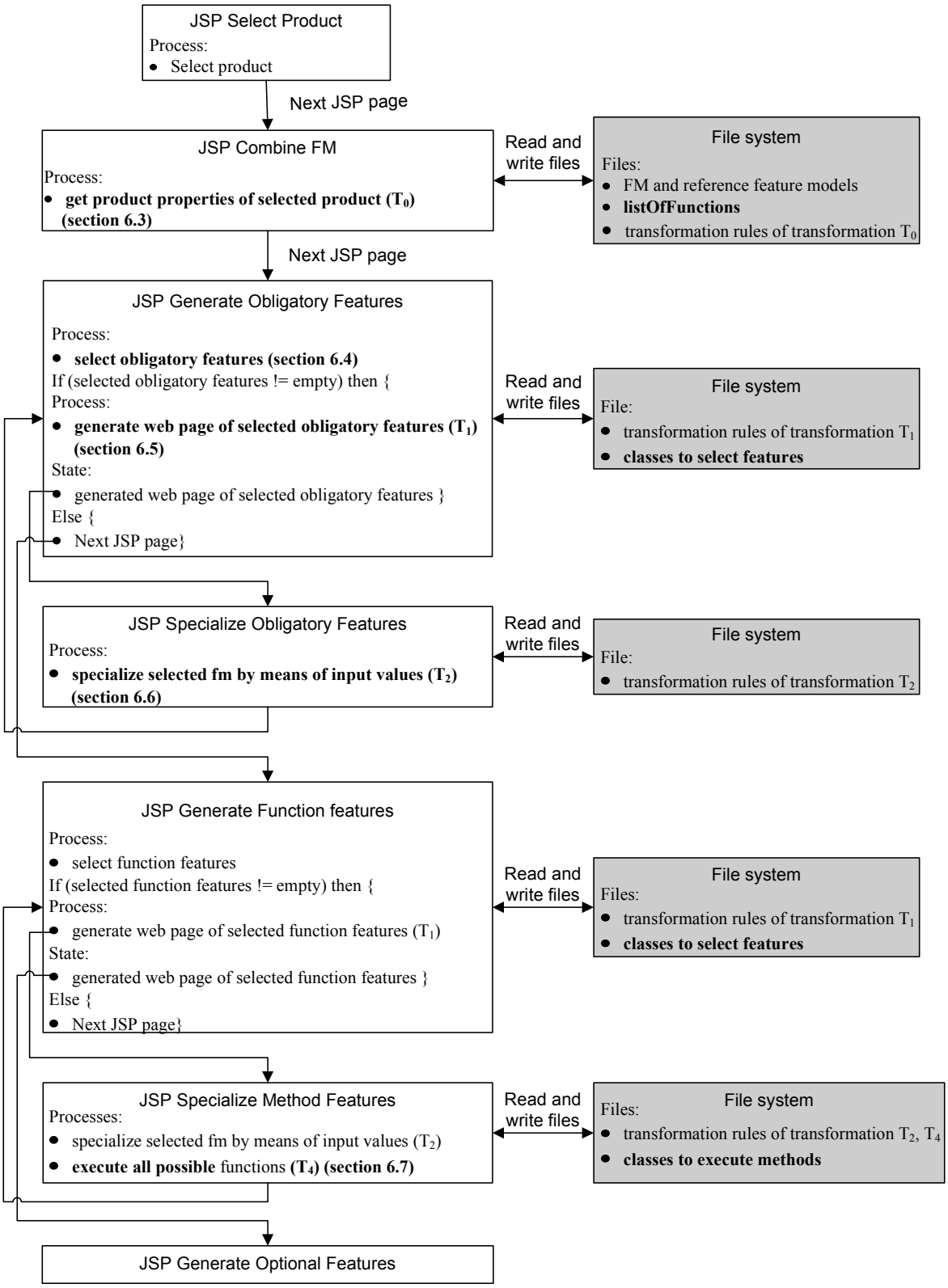
11. In deze processtap wordt het gegenereerde rapport verzonden naar de eMAXX Mid Office. De eindgebruiker krijgt nog een bevestiging te zien of het goed is verzonden of niet. Dit is tevens het einde van het proces.

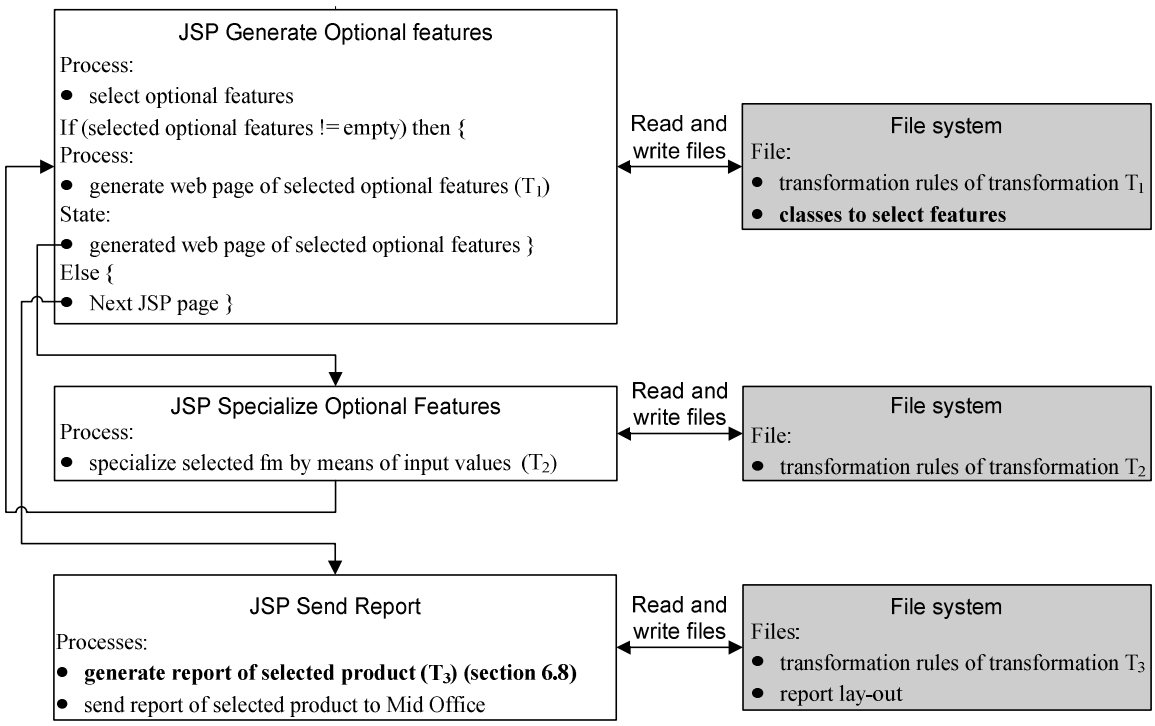
De workflow in dit hoofdstuk is een workflow die meerdere keren functies aanroept (indien nodig) en meerdere keren een webpagina genereert (indien nodig). Er is voor deze opzet gekozen om de eindgebruikers zo min mogelijk in te laten vullen en de features die gemeenschappelijke eigenschappen hebben in dezelfde lus af te handelen.

Zo worden er in lus A de verplichte features die altijd aan de burger gevraagd moeten worden gevraagd. Als deze niet worden ingevuld worden ze nogmaals gevraagd totdat de eindgebruiker deze verplichte features een waarde geeft. In lus B worden de features gevraagd waarmee functies uitgevoerd kunnen worden. Nadat een functie is uitgevoerd wordt deze verwijderd uit de lijst van functies. Deze lus wordt doorlopen totdat er geen functies meer uit te voeren zijn. In lus C worden de optionele features gevraagd. Als er geen gegevens worden ingevuld worden deze features verwijderd uit het feature model.

De nadruk van dit onderzoek is dat er gekeken wordt of het mogelijk is om automatisch webpagina’s voor de e-formulieren te genereren. Voor een vervolgonderzoek zal het een mooie opdracht zijn om te onderzoeken of er een andere opzet voor de workflow gekozen kan worden voor een betere werking van de generator.

De eerder uitgelegde processtapen van de model gedreven e-formulier generator met workflow zijn in JSP pagina's geïmplementeerd. De opbouw van deze JSP pagina's is in Figuur 6.3 weergegeven.





Figuur 6.3: overzicht JSP pagina's van de model gedreven e-formulier generator met workflow

6.2 Processtap: get product properties of selected product

Nadat een burger een product heeft geselecteerd, worden de bijbehorende producteigenschappen opgehaald. Deze processtap verschilt niet veel van de processtap: *get feature model of selected product* die in sectie 4.3 is weergegeven. Het verschil is namelijk dat er in deze processtap meerdere producteigenschappen worden opgehaald. Bij de vorige twee generatoren is het feature model enige producteigenschap die opgehaald wordt. In deze generator wordt ook nog eens een lijst van functies opgehaald. Hierin staan de functies die van toepassing zijn op het desbetreffende product.

In hoofdstuk 5 werd gebruik gemaakt van eMAXX Mid Office functie "*getPersonDetails*". Naast deze functie bevat de eMAXX Mid Office ook nog andere functies zoals "*lookUpPerson*". Ook kunnen er andere functies zijn die geen eMAXX Mid Office functies die gebruikt kunnen worden om het feature model te specialiseren zoals bijvoorbeeld de "*deduceSalutationFromGender*" functie die de feature *salutation* afleid van de feature *gender*. Met behulp van deze lijst weet de generator welke functies bij welke feature model uitgevoerd kunnen worden. De lijst, waarin alle functies voor een feature model staan zal hierna als *lijst van functies* worden genoemd.

Deze *lijst van functies* wordt in de processtap: *select obligatory feature* in sectie 6.3 voor het eerst gebruikt en zal in die sectie uitvoerig worden beschreven. In de volgende sectie wordt de implementatie van de processtap beschreven.

6.2.1 Implementatie van de processtap

In sectie 4.3 werd het feature model opgehaald en werd het feature model na de transformatie, waarbij de referentiefeatures in het feature model werden geïmporteerd, weer weggeschreven naar een bestand. Het nadeel hiervan is dat er maar één persoon tegelijk een aanvraag kan uitvoeren. Zodra er meerdere mensen dezelfde aanvraag willen indienen zullen de burgers met hetzelfde bestand werken en dit kan niet. In dit hoofdstuk worden alle benodigde bestanden opgehaald en in een sessie opgeslagen. Sessies kunnen objecten opslaan zoals het geselecteerde feature model. Alle objecten in een sessie blijven bestaan totdat de sessie is afgesloten. De sessie wordt afgesloten wanneer de burger de webpagina verlaat. Elke burger die een aanvraag doet heeft een eigen sessie waardoor er geen gegevens kunnen worden weggeschreven naar een feature model van een ander burger. Als de resultaten van de transformaties nodig zijn voor verdere processtappen, dan worden deze ook in de sessie opgeslagen.

De implementatie van het ophalen van de producteigenschappen is in Code 6.1 weergegeven. Het resultaat van de transformatie wordt in de sessie opgeslagen (zie regel 21 in Code 6.1). De lijst van functies die opgehaald is wordt ook in de sessie opgeslagen (zie regel 29 in Code 6.1).

```
1.    String sSelectedFM = request.getParameter("selectionProduct");
2.    String FileNameFM =  "..\\webapps\\Generator3\\xml\\"+sSelectedFM+"XFeatureModel.xfm";
3.
4.    //this is the input of the transformation
5.    File fFM = new File(FileNameFM);
6.
7.    //this is the file were the transformation rules are written
8.    File fTransformationRules = new
9.    File("..\\webapps\\Generator3\\xsl\\TransformationRulesToCombineFM.xsl");
10.
11.   StreamSource FM = new StreamSource(fFM);
12.   StreamSource XSL_CombineFM = new StreamSource(fTransformationRules);
13.   DOMResult resultFM = new DOMResult();
14.   StringWriter swCombineFM = new StringWriter();
15.
16.   TransformerFactory factory = TransformerFactory.newInstance();
17.   Transformer tCombineFM = factory.newTransformer(XSL_CombineFM);
18.   tCombineFM.transform(FM, resultFM);
19.
20.   //this is the session where the output of the transformation will be saved
21.   session.setAttribute( "selectedFM", resultTCombineFM );
22.
23.   String FileNameListOfFunctions = "..\\webapps\\Generator3\\xml\\" + sSelectedFM +
24.   "ListOfFunctions.xml";
25.   DocumentBuilderFactory dBFactory = DocumentBuilderFactory.newInstance (  );
26.   DocumentBuilder builder = dBFactory.newDocumentBuilder (  );
27.   Document listOfFunctions = builder.parse (FileNameListOfFunctions);
28.   //this is the session where the listOfFunctions will be saved
29.   session.setAttribute("listOfFunctions", listOfFunctions);
```

Code 6.1: implementatie van het ophalen van de producteigenschappen van een geselecteerd product

6.3 Processtap: select obligatory features

In deze sectie zal de processtap: *select obligatory features* worden bespoken. In deze processtap worden de verplichte features geselecteerd uit het geselecteerde feature model die naar een webpagina worden getransformeerd. De processtappen *select function features* en *select optional features* selecteren ook features die naar een webpagina worden getransformeerd. Deze processtappen verschillen niet veel van elkaar waardoor in dit hoofdstuk alleen de processtap: *select obligatory feature* uitvoerig wordt besproken. Ook worden de verschillen tussen deze processtapen hier besproken.

Er zijn twee verschillen tussen deze drie processtappen namelijk:

1. ***Het gebruikte selectiealgoritme om features te selecteren***

   Ieder processtap heeft een eigen selectiealgoritme die features selecteert waarbij de condities voor het selecteren van features anders zijn. In sectie 6.3.1 zullen de werkingen van deze verschillende selectiealgoritmen worden besproken.

2. ***Het refereren naar de volgende processtap***

   Ieder processtap refereert naar een ander volgende processtap, deze zijn te vinden in Figuur 6.2 (zie pagina 77).

6.3.1 Werking van de selectiealgoritmen

In deze sectie zullen de gebruikte selectiealgoritmen worden besproken. In sectie 6.3.1.1 wordt de werking van het selectiealgoritme *ObligatoryFeatureSelect* besproken. In sectie 6.3.1.2 wordt de werking van het selectiealgoritme *FunctionFeatureSelect* besproken. In sectie 6.3.1.3 wordt de werking van het selectiealgoritme *OptionalFeatureSelect* besproken.

6.3.1.1 De werking van het selectiealgoritme ObligatoryFeatureSelect

Het selectiealgoritme *ObligatoryFeatureSelect* moet de features selecteren die aan alle condities voldoen die hieronder zijn aangegeven:

1. ***Feature en zijn parent features moeten verplicht zijn***

   De geselecteerde feature en zijn *parent* features ervan moeten verplicht zijn.

2. ***Feature mag geen waarde hebben***

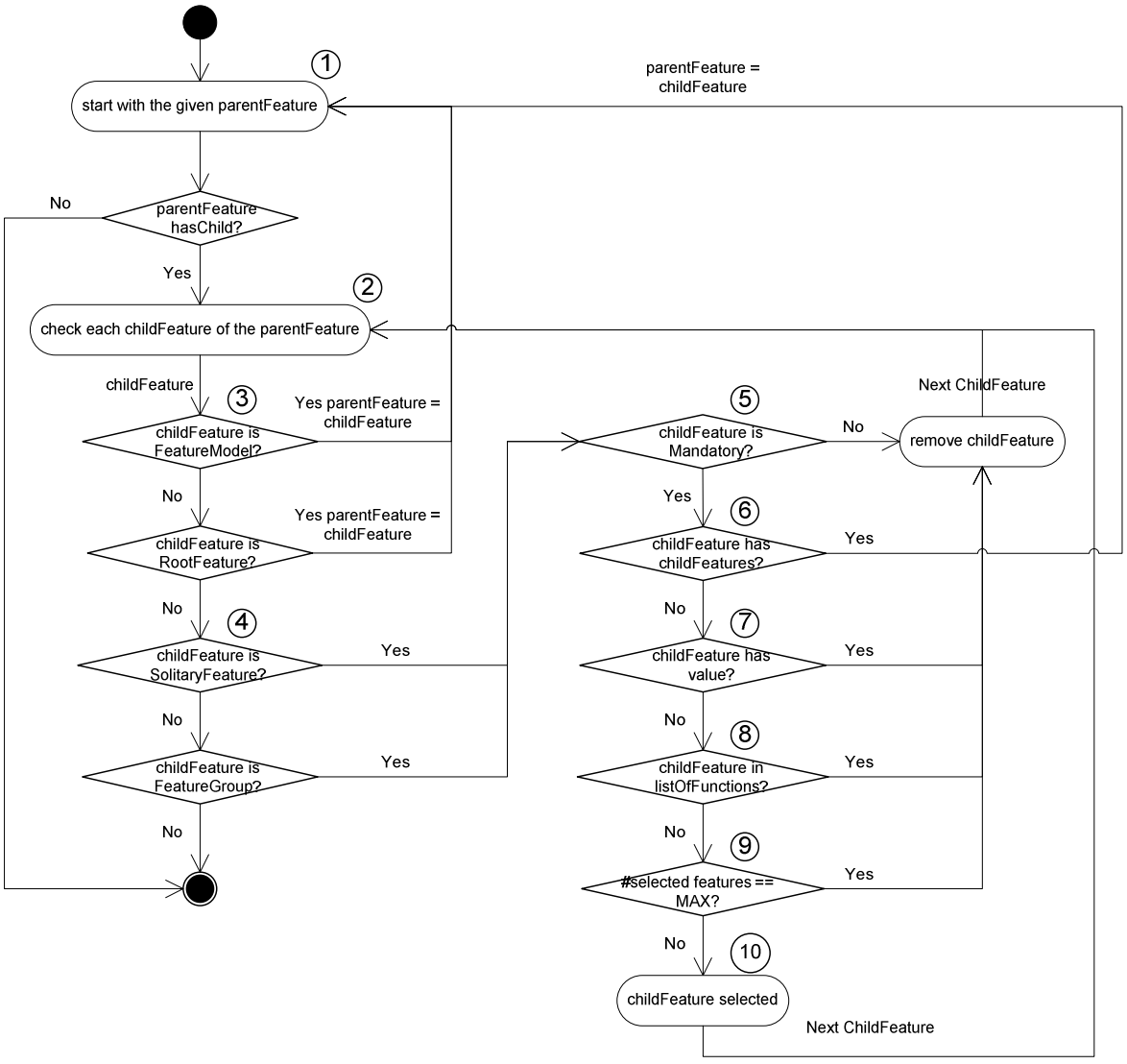
   De geselecteerde feature mag nog geen waarde hebben. Zodra een feature een waarde heeft, heeft het geen nut om deze (nog eens) aan de burger te vragen.

3. ***Feature mag geen waarde kunnen krijgen door een functie die in de lijst van functies staat***

   De geselecteerde feature mag niet door een functie een waarde kunnen krijgen die in de lijst van functies staan. Als dit wel het geval is moet er eerst geprobeerd worden om via deze functie deze waarde in te vullen (dit komt in sectie 6.6 aan de orde). Als dat niet lukt dan pas moet deze feature aan de burger worden gevraagd. Zo wordt er geprobeerd de burger zo min mogelijk gegevens in te laten vullen.

Verder houdt het selectiealgoritme *ObligatoryFeatureSelect* bij hoeveel features er zijn geselecteerd die naar een webpagina worden getransformeerd. Dit aantal mag niet het maximale aantal (die van te voren is opgegeven) overschrijden. Dit moet ervoor zorgen dat een beperkt aantal features op één pagina wordt gezet. Zo wordt een deel van het oriëntatieprobleem (zie sectie 2.2.1) opgelost.

Dit selectiealgoritme selecteert in principe die features die verplicht zijn en die alleen maar door de eindgebruiker zelf ingevuld kunnen worden. De werking van het selectiealgoritme is in Figuur 6.4 als UML activiteitdiagram weergegeven.



Figuur 6.4: UML activiteitdiagram van het selectiealgoritme ObligatoryFeatureSelect

Het selectiealgoritme krijgt het feature model en de bijbehorende lijst van functies mee. Het hele feature model wordt doorlopen. Het selectiealgoritme begint met de *root* element van het feature model die meteen een *parent* element is.

1. Wanneer een *parent* feature *children* heeft, dan gaat het algoritme verder met stap 2. Anders is het einde van het algoritme. De resulterende feature model wordt teruggegeven.

2. Elke child feature moet gecontroleerd worden. Hier wordt een *child* geselecteerd en het algoritme gaat verder met stap 3.

3. Wanneer een *child* feature van het type *FeatureModel* of *RootFeature* is wordt deze feature als *parent* feature aan het selectiealgoritme teruggegeven en worden de *child* features van deze feature gecontroleerd (terug naar stap 1). Als een *child* feature niet van deze types is, gaat het algoritme verder met stap 4.

4. Wanneer een *child* feature van het type *SolitaryFeature* of een *FeatureGroup* is gaat het algoritme verder met processtap 5 waar deze feature verder wordt gecontroleerd, anders stopt het algoritme. Het resulterende feature model wordt teruggegeven.

5. Wanneer een *child* feature niet voldoet aan de eis die bij punt 1 staat, wordt deze feature met de bijbehorende *child* features verwijderd uit het feature model en het algoritme gaat verder met stap 2 voor de volgende *child* feature. Anders gaat het algoritme verder met stap 6.

6. Wanneer een *child* feature *child* features heeft gaat het algoritme verder met stap 1 met deze *child* feature als *parent* feature en worden de *child* features van deze feature gecontroleerd. Anders gaat het algoritme verder met stap 7.

7. Wanneer een *child* feature niet voldoet aan de eis die bij punt 2 staat, wordt deze feature verwijderd uit het feature model en gaat het algoritme verder met stap 2 voor de volgende *child* feature. Anders gaat het algoritme verder met stap 8.

8. Wanneer een *child* feature niet voldoet aan de eis die bij punt 3 staat, wordt deze feature verwijderd uit het feature model en gaat het algoritme verder met stap 2 voor de volgende *child* feature. Anders gaat het algoritme verder met stap 9.

9. De teller (die het aantal tot nu toe geselecteerde features telt) wordt met één opgehoogd. Wanneer deze teller groter is dan de van te voren gestelde maximale aantal wordt deze feature verwijderd uit het feature model en gaat het algoritme verder met stap 2 voor de volgende *child* feature. Anders gaat het algoritme verder met stap 10.

10. Deze *child* feature voldoet aan alle eisen en wordt geselecteerd door deze in het feature model te laten staan. Het algoritme gaat verder met stap 2 voor de volgende *child* feature, indien aanwezig.

6.3.1.2 De werking van het selectiealgoritme FunctionFeatureSelect

Voordat het selectiealgoritme *FunctionFeatureSelect* begint met het selecteren van features, wordt eerst een functie uit de lijst van functies gekozen. Een functie wordt gekozen op basis van het aantal features die geen waarde hebben in het feature model en die mogelijkerwijze door deze functie wel een waarde kunnen krijgen. De functie die het meeste aantal features kan ophalen wordt gekozen.

Het selectiealgoritme *FunctionFeatureSelect* moet de features selecteren die aan alle condities voldoen die hieronder zijn aangegeven:

1. ***Feature is een input waarde van de geselecteerde functie***

   De geselecteerde feature moet een invoer zijn voor de geselecteerde functie. Zo worden de mogelijke invoer features voor deze functie verzameld zodat de functie in de volgende processtap aangeroepen kan worden.

2. ***Feature moet geen waarde hebben***

   De geselecteerde feature (die een invoer is voor de geselecteerde functie) moet geen ingevulde waarde hebben. Zodra deze al een waarde heeft, dan heeft het geen nut om deze nog een keer te vragen.

Als een feature niet aan de bovenstaande condities voldoet, dan wordt er ook gekeken of deze feature aan de condities van het *ObligatoryFeatureSelect* selectiealgoritme (die in sectie 6.3.1.1 zijn weergegeven) voldoet. Wanneer een functie is aangeroepen, dan wordt deze functie verwijderd uit de *lijst van functies*. Wanneer een functie is aangeroepen, dan kan het zijn dat dit geen resultaat heeft opgeleverd. In dat geval moeten de features (die niet door de functie zijn opgehaald) aan de burger gevraagd worden waardoor hier ook de condities van het *ObligatoryFeatureSelect* algoritme wordt gecontroleerd.

In Code 6.2 is als voorbeeld een deel van de lijst van functies van het feature model "*verhuizing doorgeven*" weergegeven.

```
1.    <?xml version="1.0" encoding="UTF-8"?>
2.    <functions>
3.       <function name="getPersonDetails" type="MidOfficeCall">
4.          <inputsoutputs>
5.             <input>
6.                <feature pathname="verhuizing/movement/applicant/person/aNumber"/>
7.             </input>
8.             <output>
9.                <feature pathname=" verhuizing/movement/applicant/person/name/lastName"/>
10.               <feature pathname=" verhuizing/movement/applicant/person/id"/>
11.            </output>
12.         </inputsoutputs>
13.         <inputsoutputs>
14.            <input>
15.               <feature pathname=" verhuizing/movement/applicant/person/bsn"/>
16.            </input>
17.            <output>
18.               <feature pathname=" verhuizing/movement/applicant/person/name/lastName"/>
19.               <feature pathname=" verhuizing/movement/applicant/person/id"/>
20.            </output>
21.         </inputsoutputs>
22.      </function>
23.      <function name="deduceSalutationFromGender" type="specializeFunction">
24.         <inputsoutputs>
25.            <input>
26.                  <feature pathname=" verhuizing/movement/applicant/person/gender"/>
27.            </input>
28.            <output>
29.               <feature pathname=" verhuizing/movement/applicant/person/salutation"/>
30.            </output>
31.         </inputsoutputs>
32.      </function>
33.   </functions>
```

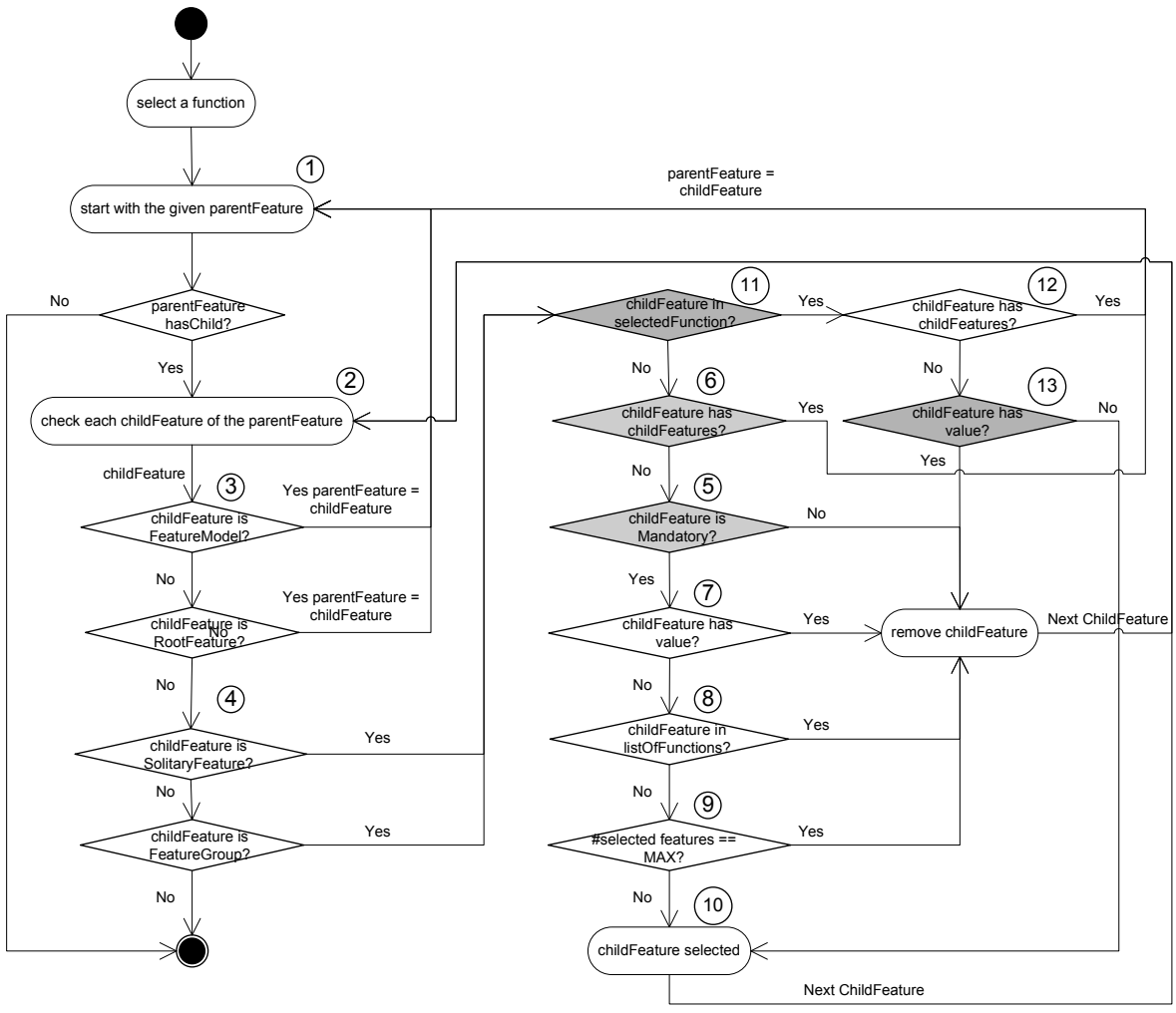
Code 6.2: een deel van de lijst van functies van het feature model "verhuizing doorgeven"

In de lijst van functies van een feature model staan alleen die functies die voor het desbetreffende feature model van toepassing zijn. Een lijst van functies heeft een element *functions* waarin de *child* elementen de naam *function* hebben. Deze *function* elementen hebben attribuut *name* welke de naam van de desbetreffende functie bevat en attribuut *type* welke aangeeft of deze een eMAXX Mid Office functie of een zelfgedefinieerde functie is. Elke *function* element heeft één of meerdere *inputsoutputs* elementen. Deze *inputsoutputs* element hebben twee *child* elementen, namelijk: *input* en *output* elementen.

In het *input* element zijn één of meerdere *feature* elementen weergegeven. Elke *feature* element heeft een attribuut *pathname* waarin de padnaam van een feature staat. Dit geldt ook voor *output* element. De *feature* elementen in het *input* element geven aan welke features nodig zijn om deze functie uit te voeren.

De *feature* elementen in het *output* element geven aan welke features een waarde kunnen krijgen nadat deze functie wordt uitgevoerd. Het kan dus zijn dat een functie wordt uitgevoerd, maar dat niet alle features, die in het *output* element staan, een waarde krijgen.

De werking van het selectiealgoritme is in Figuur 6.5 als UML activiteitdiagram weergegeven.



Figuur 6.5: UML activiteitdiagram van het selectiealgoritme FunctionFeatureSelect

Dit selectiealgoritme is bijna hetzelfde als de *ObligatoryFeatureSelect* selectiealgoritme (zie sectie 6.3.1.1) omdat diezelfde condities ook hier geldig zijn. Alleen dit selectiealgoritme krijgt twee condities erbij. Deze condities zijn met een donkere achtergrondkleur in Figuur 6.5 weergegeven. De condities die een lichtere achtergrondkleur hebben zijn de condities van het vorige selectiealgoritme. In dit selectiealgoritme zijn de plaatsen van deze twee condities gewisseld, omdat een invoer van een functie een optionele feature kan zijn of *parents* heeft die optioneel zijn. Wanneer in stap 4 de optionele features verwijdert worden, dan kunnen deze features niet worden geselecteerd. Daarom zijn stappen 4 en 5 verwisseld, zodat er alsnog naar de *child* features van een optionele feature gekeken kan worden om features te selecteren die een invoer zijn voor een functie.

De gekozen functie samen met het feature model worden als parameter meegegeven aan de functie "*selectFeatures*" die de features selecteert. Nu wordt het hele feature model doorlopen om features te selecteren.

De stappen 1 t/m 4 het vorige algoritme worden doorlopen. Na stap 4 gaat het algoritme niet verder met stap 5 maar verder met stap 11 waar de eerste conditie van dit selectiealgoritme wordt gecontroleerd.

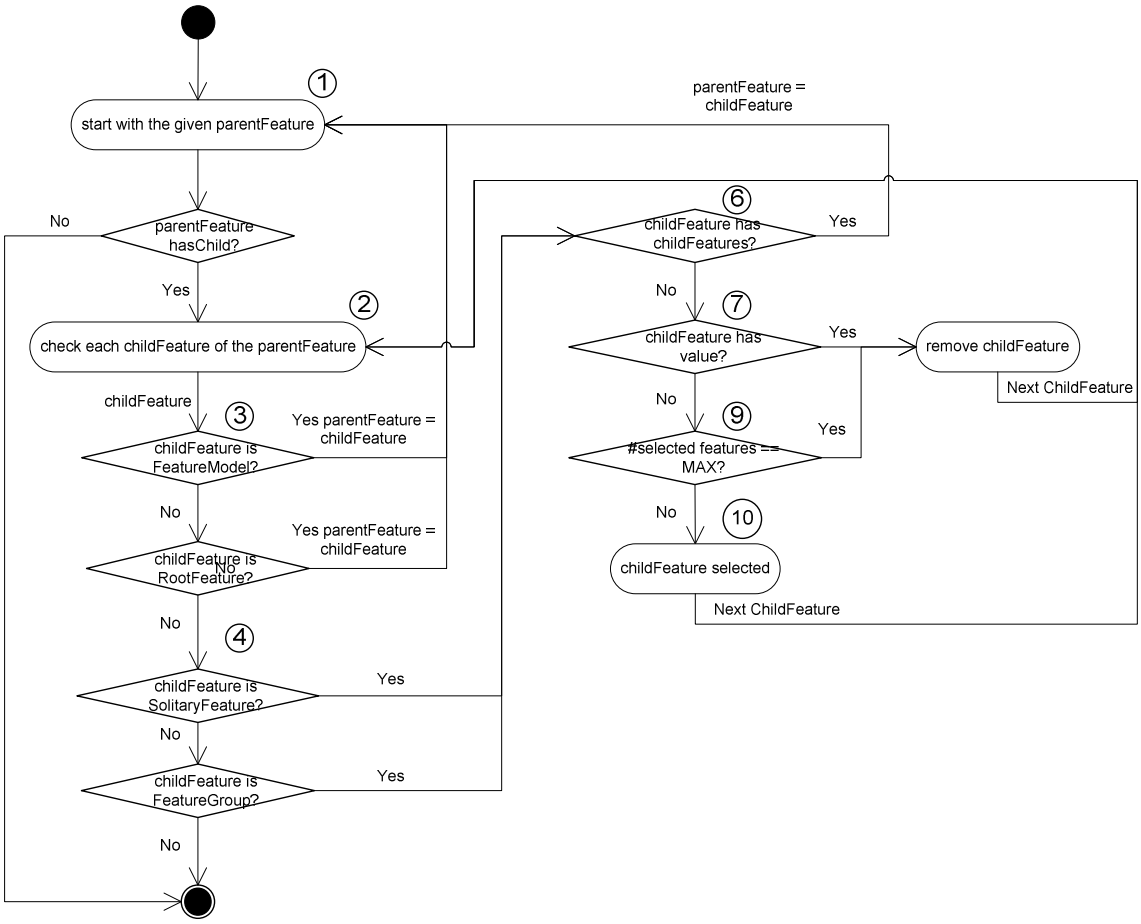
11. Wanneer een *child* feature voorkomt in de input element van de geselecteerde functie gaat het algoritme naar stap 12. Anders gaat het algoritme verder met stap 6 (let op stap 5 en 6 van het vorige selectiealgoritme zijn hier omgedraaid).

12. Deze stap is bijna hetzelfde als stap 6 waarbij er gekeken wordt of *child* feature *child* features heeft. Zodra dit het geval is, gaat het algoritme verder met stap 1. Anders gaat het algoritme verder met stap 13.

13. Wanneer een *child* feature geen waarde heeft gaat het selectieproces verder met stap 9 waar deze feature wordt geselecteerd. Anders wordt deze feature verwijderd uit het feature model en gaat het algoritme verder met stap 2 voor de volgende *child* feature, indien aanwezig.

6.3.1.3 De werking van het selectiealgoritme OptionalFeatureSelect

Het selectiealgoritme *OptionalFeatureSelect* moet ervoor zorgen dat de overgebleven optionele features die geen waarde hebben aan de eindgebruiker worden gevraagd. Ook hier wordt een counter bijgehouden die niet alle overgebleven features gaat selecteren. De werking van het selectiealgoritme is in Figuur 6.6 (zie volgende pagina) als UML activiteitdiagram weergegeven.



Figuur 6.6: UML activiteitdiagram van het selectiealgoritme OptionalFeatureSelect

Het selectiealgoritme krijgt alleen het feature model mee. Het hele feature model wordt doorlopen. Dit selectiealgoritme is bijna hetzelfde als het selectiealgoritme *ObligatoryFeatureSelect* (zie sectie 6.3.1.1). Alleen in dit selectiealgoritme worden de stappen 5 en 8 afwezig. De stap 5 is niet nodig omdat deze kijkt of het een verplichte feature is. Aangezien er alleen maar optionele features zijn gebleven, is deze stap niet nodig. De stap 8 is ook niet nodig omdat deze kijkt of er features zijn die een input zijn voor een functie. Aangezien er geen functies meer over zijn om aan te roepen, is deze stap overbodig. De rest van de stappen worden op dezelfde manier gebruikt in dit selectiealgoritme.

6.3.2 Implementatie van de processtap

De aanroep van het selectiealgoritme ObligatoryFeatureSelect is in Code 6.3 weergegeven. Het resultaat wordt gebruikt door de processtap: *generate web page of selected obligatory feature* die in sectie 6.4 wordt besproken. De implementatie code van dit selectiealgoritme is te vinden in bijlage C.

```
1.    DOMResult domResultSelectedFM = (DOMResult) session.getAttribute("selectedFM");
2.    Document listOfFunctions= (Document) session.getAttribute("listOfFunctions");
3.    DOMResult domResultObligatoryFeatures =
4.    ObligatoryFeatureSelect.returnFM(tempDomResultSelectedFM, listOfFunctions);
```

Code 6.3: implementatie voor de aanroep van het selectiealgoritme ObligatoryFeatureSelect

6.4 Processtap: generate web page of selected obligatory features

Deze processtap is bijna gelijk aan de processtappen *generate web page of selected function features* en *generate web page of selected optional features*. In deze processtap wordt gebruikt gemaakt van de transformatie T_1 die in sectie 4.4 is beschreven. Deze transformatie transformeert een gegeven feature model naar een webpagina. Deze drie processtappen verschillen op de volgende punten met elkaar:

1. Het feature model die de transformatie T_1 als invoer meekrijgt

Elke processtap heeft een eigen selectiealgoritme. Het resultaat van dit selectiealgoritme wordt als invoer meegegeven aan de transformatie T_1. Bij elke processtap wordt dus een webpagina gegenereerd van features die specifieke eigenschap hebben.

2. De transformatieregel TR_1 van de transformatie T_1

De gegenereerde webpagina's van deze processtappen moeten naar een andere JSP pagina verwijzen. Hierdoor is de transformatieregel TR_1, waarin de link naar de volgende JSP wordt gecreëerd, van deze transformatie voor elke processtap gewijzigd zodat deze naar de juiste pagina's verwijzen.

3. De verwijzing naar de volgende JSP pagina wanneer het feature model leeg is

Voordat de transformatie wordt uitgevoerd, wordt gekeken naar het feature model (die de invoer is voor de transformatie T_1) of deze features bevat die naar een webpagina getransformeerd kunnen worden. Wanneer dit het niet geval is, dan wordt er naar de volgende JSP pagina verwezen. Ieder proces verwijst naar een ander JSP pagina.

In dit hoofdstuk zal alleen de processtap: *generate web page of selected obligatory features* worden besproken. Aangezien deze processtap niet veel verschilt met de processtap die in sectie 4.4 is bespoken, zal in deze sectie alleen het verschil tussen deze processtappen worden besproken. In sectie 6.4.1 wordt de implementatie van deze processtap besproken.

Het verschil tussen deze processtappen zijn dat in deze processtap het feature model wordt gecontroleerd voordat deze naar een webpagina wordt getransformeerd. Bij de controle wordt er gekeken of er features zijn die naar een webpagina getransformeerd kunnen worden. Deze controle wordt in de vorige processtap niet uitgevoerd. De controle functie krijgt een feature model. In dit feature model wordt er naar een feature gezocht die nog geen waarde heeft. Zodra dit het geval is, dan geeft deze functie een *true* terug. Anders wordt er een *false* opgeleverd. Wanneer deze controle een true geeft, dan wordt een webpagina gegenereerd en anders gaat het aanvraagproces verder met de volgende processtap: *generate report of selected product* die in sectie 6.7 wordt beschreven.

6.4.1 Implementatie van de processtap

De implementatie van de aanroep van de controle functie en van de transformatie van deze processtap is Code 6.4 weergegeven. In deze processtap wordt het feature model verkregen door het selectiealgoritme ObligatoryFeatureSelect. In de vorige processtap wordt het feature model uit het file systeem opgehaald.

```
1.    boolean notEmpty =
2.    CheckFeaturesSelect.returnCheckFeature(domResultObligatoryFeatures);
3.    if (notEmpty) {
4.
5.    File fTransformationRules =
6.    new File("..\\webapps\\Generator3\\xsl\\T1TransformationRules.xsl");
7.
8.    StreamSource XSLwebpage = new StreamSource(fTransformationRules);
9.
10.   DOMSource FM1= new DOMSource(domResultObligatoryFeatures.getNode());
11.   StringWriter swWebpageOfObligatoryFeatures = new StringWriter();
12.
13.   TransformerFactory factory = TransformerFactory.newInstance();
14.   Transformer T1 = factory.newTransformer(XSLwebpage);
15.   T1.transform(FM1,new StreamResult(swWebpageOfObligatoryFeatures));
16.
17.   out.write(swWebpageOfObligatoryFeatures.toString() );
18.   }
19.   else
20.   {
21.      %>
22.      <script type="text/javascript">
23.      location.href = 'GenerateWebPageOfFunctionFeatures.jsp';
24.      </script>
25.      <%
26.   }
```

Code 6.4: implementatie van de processtap

6.5 Processtap: specialize selected fm by means of input values

Deze processtap komt meerdere keren in deze generator voor. Ze verschillen alleen qua verwijzing naar de volgende processtap. Deze processtappen zitten in verschillende JSP pagina's waarin de verwijzing naar de juiste JSP is aangegeven. Daar waar deze processtap voorkomt, wordt het feature model gespecialiseerd met behulp van de ingevulde gegevens op een webpagina.

Deze processtap is bijna hetzelfde als de processtap: *specialize selected feature model by means of input values* die in sectie 4.6 is weergegeven. In deze processtap wordt gebruikt gemaakt van de transformatie T_2 die in sectie 4.6 is beschreven. Deze transformatie specialiseert het feature model met behulp van de ingevulde gegevens op een webpagina. In sectie 6.5.1 worden de verschillen tussen deze twee processtappen weergegeven. In sectie 6.5.2 wordt de implementatie van deze processtap weergegeven.

6.5.1 Verschillen tussen deze twee processtappen

De verschillen tussen deze processtappen zijn:

- ***De transformatie T_2 krijgt het dataoutput document als parameter mee***.

  In deze processtap worden de ingevulde waarden op een webpagina in een XML document opgeslagen die niet naar een file wordt geschreven. Dit XML document wordt vervolgens als parameter meegegeven aan de transformatie. Tijdens de transformatie wordt deze parameter ingelezen in een template i.p.v. een bestand zoals in sectie 4.6.

- ***Verwijzing naar de volgende JSP pagina***

  De processtap, die in sectie 4.6 is beschreven, is geïmplementeerd in de laatste JSP pagina (*JSP Send Report*) waardoor er geen verwijzing is naar een andere JSP pagina. In deze processtap wordt terug naar de JSP pagina (*JSP Generate Obligatory Features*) verwezen, waarin de processtap: *select obligatory features* (die in sectie 6.3 is beschreven) wordt uitgevoerd.

6.5.2 Implementatie van de processtap

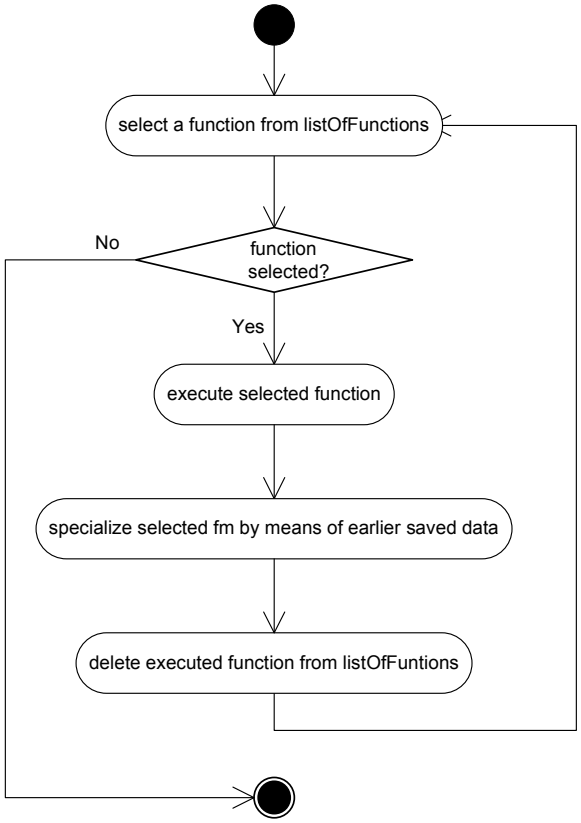
De implementatie van de aanroep van de transformatie T_2 is in Code 6.5 weergegeven. Het feature model wordt in deze processtap niet uit een file gelezen zoals in sectie 4.6, maar uit de sessie.

```
1.    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
2.    DocumentBuilder builder = factory.newDocumentBuilder();
3.    Document ouputFormDataDoc =
4.    builder.parse(new InputSource(new StringReader(stringFormData)));
5.
6.    File fTransformationRules = new
7.    File("..\\webapps\\Generator3\\xsl\\T2TransformationRules.xsl");
8.    DOMResult domResultSelectedFM = (DOMResult) session.getAttribute( "selectedFM" );
9.    DOMSource FM1 = new DOMSource(domResultSelectedFM.getNode());
10.   StreamSource XSLspecialize = new StreamSource(fTransformationRules);
11.   DOMResult FM2 = new DOMResult();
12.   TransformerFactory factory = TransformerFactory.newInstance();
13.   Transformer T2 = factory.newTransformer(XSLspecialize);
14.   T2.setParameter("FormDataOutput", ouputFormDataDoc);
15.   T2.transform(domSourceSelectedFM,resultTFormDataSpecializeFM);
16.   session.setAttribute("selectedFM", resultTFormDataSpecializeFM);
17.   %>
18.   <script type="text/javascript">
19.      location.href = 'GenerateWebPageOfObligatoryFeatures.jsp';
20.   </script>
```

Code 6.5: implementatie van de processtap

6.6 Processtap: execute all possible functions

Deze processtap maakt gebruik van de functie *ExecuteFunctions* om functies uit te voeren en het feature model te specialiseren aan de hand van deze functies. Het feature model en de bijbehorende lijst van functies worden als parameter meegegeven aan deze functie. Deze functie wordt in de JSP pagina (*JSP Specialize Function Features*) aangeroepen. Nadat deze functie is aangeroepen en is uitgevoerd gaat het proces terug naar de processtap: *select function features*. Het UML activiteitdiagram van de functie *ExecuteFunctions* is in Figuur 6.7 weergegeven.



Figuur 6.7: UML activiteitdiagram van de functie ExecuteFunctions

Deze functie selecteert eerst een functie uit de lijst van functies. Deze gaat de lijst langs en kijkt voor ieder functie naar de features in de input element en of deze een waarde hebben in het feature model. Zodra er een functie wordt gevonden waarbij alle input elementen een waarde hebben in het feature model, dan wordt deze functie geselecteerd en wordt niet verder gezocht naar een andere functie. Wanneer er geen functie wordt gevonden die uitgevoerd kan worden, wordt er een *null* gegeven.

Vervolgens gaat deze functie kijken of er een functie is geselecteerd. Als dit het geval is dan gaat de *ExecuteFunctions* functie de geselecteerde functie uitvoeren. Het resultaat van de uitgevoerde functie wordt dan meteen weggeschreven naar het feature model met behulp van de transformatie T_4 die in sectie 5.4 is besproken. Alleen hier wordt het resultaat van de uitgevoerde functie niet weggeschreven naar de file systeem maar wordt meteen gebruikt. Het gebruikte feature model voor de transformatie wordt niet uit het file systeem opgehaald, maar deze is als parameter meegegeven aan de functie *ExecuteFunctions*.

Nadat het feature model is gespecialiseerd, wordt het uitgevoerde functie uit het lijst van functies verwijdert zodat deze niet nog een keer wordt uitgevoerd. Vervolgens gaat de functie *ExecuteFunctions* opnieuw beginnen met het selecteren van een functie uit de lijst van functies om de resterende functies uit te voeren die uitgevoerd kunnen worden.

Wanneer er geen functie meer is die uitgevoerd kan worden, dan stopt de *ExecuteFunctions* functie. Vervolgens gaat het proces terug naar de processtap: *select function features* die in JSP pagina (*JSP Generate Function Features*) wordt uitgevoerd.

In Code 6.6 is de implementatie deze processtap weergegeven, waarbij de functie *ExecuteFunctions* wordt aangeroepen.

```
1.    ExecuteFunctions.ExecuteFunctions(selectedFM, listOfFunctions);
2.    %>
3.    <script type="text/javascript">
4.    location.href = 'GenerateFunctionFeatures.jsp';
5.    </script>
6.    <%
```

Code 6.6: implementatie van de processtap

6.7 Processtap: generate report of selected product

Deze processtap is bijna hetzelfde processtap die in sectie 4.7 is beschreven. Ook hier wordt de transformatie T_3 uitgevoerd. Het verschil van deze processtap met de vorige processtap is dat de feature model voor deze transformatie niet uit de file systeem wordt opgehaald, maar uit de sessie zoals in sectie 6.2 voor de transformatie T_0 is beschreven. Hierdoor zal deze processtap niet verder worden behandeld.

7 Evaluatie/testen van de model gedreven e-formulier generator

In dit hoofdstuk wordt de model gedreven e-formulier generator met workflow geëvalueerd/getest omdat deze generator het meest complexe is van de ontwikkelde generatoren. Daarbij worden twee aspecten geëvalueerd/getest namelijk:

1. ***Effectiviteit: de ontworpen generator kan gebruikt worden voor het automatisch genereren van webpagina's van een willekeurig opgestelde e-formulier***

   Het hoofddoel van het evalueren/testen van dit aspect is een indicatie krijgen of deze generator inderdaad effectief is voor een willekeurig opgestelde e-formulier.

2. ***Efficiëntie: werken met generatoren kost minder tijd dan het uitprogrammeren van een e-formulier***

   Het hoofddoel van het evalueren/testen van dit aspect is een indicatie krijgen of het inderdaad minder tijd kost om met generatoren te werken dan het uitprogrammeren van een e-formulier.

In sectie 7.1 worden de testmethoden besproken. In sectie 7.2 worden de resultaten van de huidige ontwikkeltechniek van de e-formulieren besproken. In sectie 7.3worden de resultaten van de nieuwe ontwikkeltechniek van de e-formulieren besproken. In sectie 7.4 worden de resultaten van de ontwikkeltechnieken met elkaar vergeleken

7.1 Testmethoden

Om de effectiviteit te testen, worden de twee e-formulieren ingevuld door een eindgebruiker. Nadat deze ingevuld is door een eindgebruiker, worden het aantal verkeerd ingevulde waarden geteld. Het invullen van deze e-formulieren gaat met behulp van scenario's die in sectie 7.1.1 zijn weergegeven. Het is normaal de bedoeling dat deze test toegepast wordt op de huidige en de nieuwe ontwikkeltechniek van de e-formulieren. De e-formulieren van de huidige ontwikkeltechniek zijn al commercieel in gebruik en zijn destijds al getest en werken prima, waardoor alleen de nieuwe ontwikkeltechniek wordt getest.

Om de efficiëntie te testen, worden twee e-formulieren opgesteld door systeemontwerpers met behulp van scenario's die in sectie 7.1.2 zijn weergegeven., waarbij de benodigde tijden voor het opstellen van de e-formulieren worden gemeten. Het is normaal de bedoeling dat deze test toegepast wordt op de huidige en de nieuwe ontwikkeltechniek van de e-formulieren waarbij er e-formulieren opgesteld worden door systeemontwerpers die evenveel kennis hebben over beide ontwikkeltechnieken. Aangezien ik de enige ben die bekend is met de nieuwe ontwikkeltechniek worden de e-formulieren met deze ontwikkeltechniek door mij opgesteld. Gezien mijn kennis over de huidige ontwikkeltechniek minder is dan de nieuwe ontwikkeltechniek, worden de e-formulieren met de huidige ontwikkeltechniek door een systeemontwerper van eMAXX opgesteld. Daarbij zal de webpagina's het zelfde uiterlijk en hetzelfde functionaliteit moeten hebben als de gegenereerde webpagina's door de generator. Gezien de tijd korte tijd die er is is dit niet haalbaar en worden de benodigde tijd voor het opstellen van e-formulieren met de huidige ontwikkeltechniek uitgerekend met een formule die in sectie 7.1.3 wordt beschreven.

7.1.1 Scenario's voor het invullen van de e-formulieren

Bij het invullen van de e-formulieren moet bij de volgende features de volgende waarden ingevuld worden (als die er staat) gebruikmakend van de volgende twee scenario's:

- ***Scenario voor het invullen van het e-formulier voor uitreksel aanvraag***

• excerptRequestMessage		
o excerptInfo		
▪ referenceNumber	33333333	
▪ date	26-11-2007	
▪ description	uittreksel aanvraag	
▪ remark	oficieel te gebruiken	
▪ Product		
• id	0	
• code	0	
• naam	geboorteakte	
• alias		
o code	0	
o naam	geboorteakte	
▪ international	nee	
▪ language		
• isoCode	31	
• name	Nederlands	
▪ certificate		
• birth		
o familyMember	refPerson	Persoonsgegevens familie lid
o declarerInfo	refPerson	Persoonsgegeven aanvrager
o addressInfo	refAddress	Adresgegevens aanvrager
declarerInfo/familyMember		
• person		
o name		
▪ initials	T	
▪ firstNames	Tina	
▪ middleName	Sandra	
▪ lastName	Mol	
o id	70	
o citizenNumber	950011502	
o anumber	2301313497	
o bsn	950011502	
o gender	V	
o salutation	Mevr	
• address		
o id	70	
o description	woning	
o street		
▪ id	2666	
▪ name	de Anjen	
o houseNumber		
▪ number	23	
▪ letter	p	
▪ remark	c	
▪ description	huurwoning	
o zipCode	8918LC	

- ***Scenario voor het invullen van het e-formulier voor kapvergunning aanvraag***

• fellingPermitMessage		
o applicant		
▪ person	refperson	Persoonsgegevens aanvrager
▪ address	refaddress	Adresgegevens aanvrager
o site		
▪ ownerschip		
• Own ground	eigen grond	
▪ location		
• number	1	
• streetName	melkweg	
• houseNumber	11	
• houseLetter	l	

```
        • houseRemark                 a
        • zipCode                     1111 pp
        • city                        Enschede
    o permit
      ▪ referenceNumber               333333333
      ▪ object                        kapvergunning
      ▪ broodPresent                  nee
      ▪ animalsPresent                nee
      ▪ Derden geïnformeerd           ja
      ▪ discussionsHeld               mondeling
      ▪ reason                        valt bijna om
      ▪ Relatie met bouwaanvraag?     nee
      ▪ relationWithBuildingpermit    nee
      ▪ details
        • location                    1
          o locationNumber            1
          o treeSpecies               Eikeboom
          o number                    1
          o circumference             100
          o frontGarden               nee
          o backGarden                ja
          o sideGarden                nee
          o position                  rechts achter
          o reason                    valt bijna om
applicant
• person
    o name
      ▪ initials                      T
      ▪ firstNames                    Tina
      ▪ middleName                    Sandra
      ▪ lastName                      Mol
    o id                              70
    o citizenNumber                   950011502
    o anumber                         2301313497
    o bsn                             950011502
    o gender                          V
    o salutation                      Mevr
• address
    o id                              70
    o description                     woning
    o street
      ▪ id                            2666
      ▪ name                          de Anjen
    o houseNumber
      ▪ number                        23
      ▪ letter                        p
      ▪ remark                        c
      ▪ description                   huurwoning
      ▪ zipCode                       8918LC
```

7.1.2 Scenario’s voor het ontwikkelen van de e-formulieren

De features die opgesteld worden hebben ieder een waarde dat aangeeft om welke feature het gaat. Ieder feature heeft ook een type. Er zijn volgende typen: *sf* (*SolitaryFeature*), *rf* (*RootFeature*), *fg* (*FeatureGroup*), *gf* (*GroupedFeature*) en *refX* (*Reference*F*eature* waarbij *X* de gerefereerde feature model is). Bij het type van de feature wordt ook de cardinaliteit gegeven wanneer die *opt* (*optional*) is. De features kunnen attribuut hebben. Wanneer dit het geval is, dan wordt het type van het attribuut opgegeven. Elke feature (behalve de *RootFeatures* en *ReferenceFeatures*) heeft verder een beschrijving die de feature beschrijft. De referentie features zijn al eerder opgesteld en aangezien deze één keer opgesteld hoeft te worden, worden de tijden voor het realiseren van referentie features niet opgenomen voor het realiseren van de feature modellen.

De systeemontwerper die deze e-formulieren met de nieuwe ontwikkeltechniek gaat opstellen zal gebruik maken van de volgende twee scenario’s:

- ***Scenario voor het ontwikkelen van e-formulier voor uittreksel aanvraag***

  De volgende features moeten opgesteld worden in het feature model voor het e-formulier "*uittreksel aanvraag*":

Feature waarde:	**soort feature**:	**type attribuut:**	**beschrijving:**
• excerptRequestMessage	rf		
o excerptInfo	sf		Uittreksel info
▪ referenceNumber	sf	string	Referentienummer
▪ date	sf	string	Huidige datum
▪ description	sf	string	Beschrijving aanvraag
▪ remark	sf	string	extra beschrijving aanvraag
▪ product	sf		Product
• id	sf	int	product id
• code	sf	string	product code
• name	sf	string	product naam
• alias	sf		Product alias
o code	sf	string	alias code
o name	sf	string	alias naam
▪ international	sf opt	boolean	internationale uittreksel?
▪ language	sf		Taal van het uittreksel
• isoCode	sf	string	Taal iso code
• name	sf	string	Taal
▪ certificate	fg opt		Uittreksel
• birth	gf		Geboorte akte
o familyMember	refPerson		relatie tot aanvrager
• death	gf		Overlijdensakte
o dateOfDeath	sf	string	Datum van overlijden
o placeOfDeath	sf	string	Plaats van overlijden
o relation	sf	string	relatie tot aanvrager
o person	refPerson		Persoon overleden
• divorce	gf		Echtscheidingsakte
o dateOfMarriage	sf	string	Datum vorig huwelijk
o dateOfDivorce	sf	string	Datum echtscheiding
o placeOfMarriage	sf opt	string	Plaats vorige huwelik
o partner	refPerson		Partner
• marriage	gf		Huwelijks akte
o dateOfMarriage	sf	string	Datum huwelijk
o placeOfMarriage	sf opt	string	Plaats huwelijk
o partner	refPerson opt		Partner
• marriageToPartnership	gf		Omzetting huwelijk in partnerschap
o dateOfConversion	sf	string	Datum omzetting huwelijk in partnerschap
o dateOfMarriage	sf opt	string	Datum huwelijk
o placeOfMarriage	sf opt	string	Plaats huwelijk
o partner	refPerson opt		Partner
• partnership	gf		partnerschap
o dateOfPartnership	sf	string	Datum partnerschap
o placeOfPartnership	sf opt	string	Plaats partnerschap
o partner	refPerson opt		Partner
• partnershipTermination	gf		Beeindiging partnerschap
o dateOfPartnership	sf	string	Datum partnerschap
o dateOfPartnershipTermination	sf	string	Datum beeindiging partnerschap
o placeOfPartnership	sf opt	string	Plaats partnerschap
o partner	refPerson opt		Partner
• partnershipToMarriage	gf		Omzetting partnerschap in huwelijk
o dateOfConversion	sf	string	Datum omzetting partnerschap in huwelijk
o dateOfPartnership	sf opt	string	Datum partnerschap
o placeOfPartnership	sf opt	string	Plaats partnerschap
o partner	refPerson opt		Partner
o declarerInfo	refPerson		persoonsgegeven aanvrager
o addressInfo	refAddress		adresgegevens aanvrager
o *child*Info	refPerson opt		persoonsgegevens kind
o *child*AddressInfo	refAddress opt		adresgegevens kind

Naast dit opgestelde feature model wordt ook de bijbehorende lijst van functies opgesteld. Er wordt alleen gebruikt gemaakt van de functie "*getPersonDetails*" die (als het kan) de waarden van de referentie features ophaalt. In deze lijst van functies worden dan de path van de referentiefeatures en van de features van de referentiefeatures opgegeven.

- ***Scenario voor het ontwikkelen van e-formulier voor kapvergunning aanvraag***

  De volgende features moeten opgesteld worden in het feature model voor het e-formulier "*kapvergunning aanvraag"*:

Feature waarde:	**soort feature:**	**type attribuut:**	**beschrijving:**
• fellingPermitMessage	rf		
o applicant	sf		Aanvrager
▪ person	refperson		
▪ address	refaddress		
o representative	sf opt		Gemachtigde
▪ person	refperson		
▪ address	refaddress		
o lessor	sf opt		Eigenaar
▪ person	refperson		
▪ address	refaddress		
o site	sf		Terrein
▪ ownership	fg		Eingendom
• Own ground	gf	Own ground	eigen grond
• Ground rent	gf	Ground rent	erf pacht
• Rent	gf	Rent	huur
▪ location	sf opt		Lokale aanduiding
• number	sf	string	Nummer
• StreetName	sf	string	Straatnaam
• houseNumber	sf	int	Huisnummer
• houseLetter	sf	string	Huisletter
• houseRemark	sf	string	Toevoeging
• zipCode	sf	string	Postcode
• city	sf	string	Woonplaats
▪ registry	sf opt		Kadastrale aanduiding
• municipalityCode	sf	string	Gemeente code
• section	sf	string	Sectie
• number	sf	string	Nummer
• letter	sf opt	string	Letter
• index	sf opt	string	Index
o permit	sf		Kapvergunning voor
▪ referenceNumber	sf	string	Referentienummer
▪ object	sf	string	Object
▪ broodPresent	sf	boolean	Broedsels in een boom
▪ animalsPresent	sf	boolean	Wonende dieren in een boom
▪ discussionsHeld	sf	boolean	Derden geïnformeerd
▪ information	sf	string	Informatie informeren derden
▪ reason	sf	opt string	Reden
▪ relationWithBuildingpermit	sf opt	boolean	Relatie met bouwaanvraag?
▪ plantNewTree	sf opt	boolean	Is er sprake van herbeplanting?
▪ details	sf		
• location	sf opt		Locatie
o locationNumber	sf	string	Locatienummer
o treeSpecies	sf	string	Boomsoort
o number	sf	int	Aantal
o circumference	sf	int	Omtrek in centimeters
o frontGarden	sf	boolean	Voortuin
o backGarden	sf	boolean	Achtertuin
o sideGarden	sf	boolean	Zijtuin
o position	sf opt	string	Plaats
o reason	sf	string	Reden
▪ various	sf opt	string	Variatie

Ook hier wordt de lijst van functies opgezet waarbij alleen de functie "*getPersonDetails*" wordt gebruikt zoals in het vorige scenario is aangegeven.

7.1.3 Uitrekenen van de benodigde tijden voor het realiseren van e-formulieren met huidige ontwikkeltechniek

Het vergelijken van de tijden voor het opstellen van de e-formulieren met verschillende ontwikkeltechnieken is erg moeilijk. Dit heeft te maken met het feit dat de opgestelde e-formulieren met de huidige ontwikkeltechniek geavanceerder zijn en er beter uitzien dan de e-formulieren die met de nieuwe ontwikkeltechniek worden opgesteld. Ook worden sommige velden van de webpagina van een e-formulier met de huidige ontwikkeltechniek gevalideerd. In de nieuwe ontwikkeltechniek is dit nog niet het geval. Om een goede vergelijking te kunnen maken is een systeemontwerper van eMAXX het volgende gevraagd:

Hoeveel tijd kost het realiseren van één gegeven?

Hiervoor worden de volgende activiteiten uitgevoerd:

- **Gevraagde gegeven tonen op een webpagina**

  Aan een webpagina moet een veld toegevoegd worden om het gevraagde gegeven aan de burger te kunnen vragen.

- **Ingevulde gegeven opslaan in een object**

  Het ingevulde gegeven moet in een object worden opgeslagen om deze verder te kunnen gebruiken.

- **Gegeven ophalen uit een object en wegschrijven naar het rapport**

  Het object, waarin het ingevulde gegeven staat, moet weggeschreven worden naar het rapport.

Het antwoord van de systeemontwerper van eMAXX op de gestelde vraag is: gemiddeld kost het 6 minuten voor het realiseren van één gegeven om deze aan de burger te vragen warbij de bovenstaande activiteiten worden uitgevoerd.

Er kunnen ook gegevens opgehaald worden. Hiervoor moeten de volgende activiteiten worden uitgevoerd in de huidige e-formulieren om een soortgelijk resultaat te krijgen als de nieuwe e-formulieren.

- **Invoer voor de functie moet uit een object worden gelezen**

  Als een functie een invoer nodig heeft moet deze uit een object opgehaald worden en meegegeven worden aan deze functie.

- **Het benodigde gegeven uit het resultaat ophalen en naar het rapport schrijven**

  Het resultaat van de uitgevoerde functie moet naar het rapport worden weggeschreven. Deze waarden worden eerst in object opgeslagen en vervolgens in het rapport gestopt.

Aan de systeemontwerper van eMAXX zijn vervolgens de volgende twee vragen gevraagd:

1. ***Hoeveel tijd kost het realiseren om één invoergegeven aan een functie mee te geven om deze functie te kunnen aanroepen?***

2. ***Hoeveel tijd kost het realiseren van één gegeven (die het resultaat is van de uitgevoerde functie) op te slaan in een object en vervolgens in het rapport stoppen?***

Het antwoord op vraag 1 is dat het gemiddeld genomen 2 minuten duurt per invoer om deze op te halen en aan de functie mee te geven. Het antwoord op de vraag 2 is dat het gemiddeld genomen 2 minuten duurt om één gegeven van het resultaat in een object op te slaan en deze vervolgens in het rapport te stoppen.

Met deze antwoorden kan een formule opgesteld worden waarmee het uitgerekend kan worden hoeveel tijd het kost voor het realiseren van een e-formulier met de huige ontwikkeltechniek.

De formule voor het uitrekenen van de benodigde tijd voor het realiseren van een e-formulier is:

> **Het realiseren van webpagina's** = het aantal gegevens dat aan de burger gevraagd moeten worden x 6 min.
>
> Voor elke functie die uitgevoerd kan worden moet het volgende berekend worden:
>
> **Het realiseren van een functie** =
> het aantal gegevens die invoer zijn voor een functie x 2 min +
> het aantal gegevens die resulteren nadat een functie is uitgevoerd x 2 min.
>
> Algemeen zal dan voor een e-formulier het volgende gelden:
>
> *Realiseren van webpagina* + *realiseren van functies* (elke functie wordt afzonderlijk uitgerekend en wordt aan het eind al deze tijden opgeteld) = de benodigde tijd om het e-formulier op te stellen

7.2 Huidige ontwikkeltechniek van de e-formulieren

In deze sectie zullen de resultaten van de uitgevoerde testen met de huidige ontwikkeltechniek van de e-formulieren worden besproken. In sectie 7.2.1 worden de resultaten van het testen van de effectiviteit van deze ontwikkeltechniek besproken. In sectie 7.2.2 worden de resultaten van het testen van de efficiëntie van deze ontwikkeltechniek besproken.

7.2.1 Effectiviteit van de huidige ontwikkeltechniek

Zoals eerder aangegeven, de e-formulieren die opgesteld zijn met de huidige ontwikkeltechniek zijn al in gebruik en zijn destijds al getest en werken prima. De e-formulieren uittreksel aanvraag en kapvergunning aanvraag zijn al eerder opgesteld door eMAXX. Deze e-formulieren zijn bij de ontwikkeling ervan al getest door de systeemontwerpers van eMAXX waardoor deze niet nog eens zijn getest.

7.2.2 Efficiëntie van de huidige ontwikkeltechniek

De efficiëntie van de huidige ontwikkeltechniek wordt bepaald door de benodigde tijd voor het opstellen van de e-formulieren. In sectie7.1.3 is een formule opgesteld om de tijd voor het opstellen van een e-formulier uit te rekenen.

De benodigde tijd voor het opstellen van de e-formulieren met de huidige ontwikkeltechniek is als volgt uitgerekend:

De features van de referentie features die in deze twee e-formulieren voorkomen, kunnen opgehaald worden door de functie *getPersonDetails*. Aangezien de benodigde tijd voor het opstellen van de referentie features niet zijn meegeteld, worden in de huidige ontwikkeltechniek deze ook niet meegeteld. Wel is er een lijst van functies opgesteld. Om gelijksoortige vergelijking te kunnen maken worden de tijden voor het realiseren van de invoeren van de functies wel meegerekend maar voor de uitvoeren niet. Wanneer de uitvoeren ook meegerekend worden, dan zullen het opstellen van de referentie features ook meegerekend worden wat niet is gedaan.

Uittreksel aanvraag:

Het e-formulier *uittreksel aanvraag* heeft 32 gegevens die aan de eindgebruiker worden gevraagd. In dit e-formulier kan op 10 plekken de functie *getPersonDetails* aangeroepen worden. Voor deze functie kan een A-nummer of een BSN nummer aan de eindgebruiker gevraagd worden. De formule voor dit e-formulier is hieronder weergegeven.

32 gevraagde gegevens aan de burger x 6 min = 192 min
2 (voor een A-nummer of voor een BSN) x (10 x *getPersonDetails* x 2 min) = 40 min

Totaal = 192 + 40 = 232 min = **3 uur 52 min** is de benodigde tijd om dit e-formulier op te stellen.

Kapvergunning aanvraag:

Het e-formulier kapvergunning aanvraag heeft 31 gegevens die aan de eindgebruiker worden gevraagd. In dit e-formulier kan op 3 plekken de functie *getPersonDetails* aangeroepen worden. De formule voor dit e-formulier is hieronder weergegeven.

31 gevraagde gegevens aan de burger x 6 min = 186 min
2 x (voor een A-nummer of voor een BSN) x (3 x *getPersonDetails* x 2 min) = 12 min

Totaal = 186 + 12 = 198 min = **3 uur 18 min** is de benodigde tijd om dit e-formulier op te stellen.

In de onderstaande Tabel 7.1 zijn de kwalitatieve resultaten genoteerd bij het opstellen van de e-formulieren. De linkerkolom geeft aan om welke e-formulier het gaat. De rechterkolom geeft aan hoeveel tijd nodig is om het e-formulier op te stellen met de huidige ontwikkeltechniek.

	Benodigde tijd voor het opstellen van de e-formulieren met de huidige ontwikkeltechniek
Uittreksel aanvraag	3 uur 52 min
Kapvergunning aanvraag	3 uur 18 min

Tabel 7.1 kwalitatieve resultaten bij het opstellen van de e-formulieren met de huidige ontwikkeltechniek

De reden dat het opstellen van het e-formulier voor uittreksel aanvraag langer duurt dan van de kapverginning aanvraag komt doordat er in uittrekselaanvraag op meerdere plekken een functie aangeroepen kan worden. Hiervoor moeten extra verhandelingen verricht worden waardoor het langer duurt om dit e-formulier op te stellen.

7.3 Nieuwe ontwikkeltechniek van de e-formulieren

In deze sectie zullen de resultaten van de uitgevoerde testen met de nieuwe ontwikkeltechniek van de e-formulieren worden besproken. In sectie 7.3.1 worden de resultaten van het testen van de effectiviteit van deze ontwikkeltechniek besproken. In sectie 7.3.2 worden de resultaten van het testen van de efficiëntie van deze ontwikkeltechniek besproken.

7.3.1 Effectiviteit van de nieuwe ontwikkeltechniek

De e-formulieren uittreksel aanvraag en kap vergunning zijn ingevuld door een eindgebruiker met behulp van de generator. Nadat alle gegevens zijn ingevuld door de eindgebruiker zijn de ingevulde waarden gecontroleerd. De resultaten zijn in Tabel 7.2 weergegeven. De linkerkolom geeft aan om welke e-formulier het gaat. De andere kolommen geven het aantal fouten, die opgedeeld zijn in typen, weer bij het invullen van de e-formulieren.

	# niet ingevulde verplichte features	**# niet ingevulde optionele features die wel gevuld moesten worden gezien het scenario**	**#features die verkeerd zijn ingevuld**	**# niet ingevulde features waarvan de waarden wel opgeslagen zijn**
Uittreksel aanvraag	9	0	9	7
Kapvergunning aanvraag	0	0	19	7

Tabel 7.2: kwalitatieve resultaten bij het invullen van de e-formulieren met behulp van de generator

De resultaten zullen nader toegelicht worden. Allereerst zullen de fouten van het ingevulde *uittreksel aanvraag* e-formulier worden besproken. Vervolgens zullen de fouten van het ingevulde *kapvergunning aanvraag* e-formulier worden besproken.

- ***Uittreksel aanvraag***

    - **Het aantal niet ingevulde verplichte features**

      In het ingevulde e-formulier zijn er 9 niet ingevulde verplichte features. Dit heeft betrekking tot het invullen van de persoonsgegevens van de familierelatie wanneer je een geboorte wilt doorgeven. De reden dat deze features niet zijn ingevuld is de cardinaliteit van de *parent* feature *certificate*. Wanneer een van de producten in certificate geselecteerd wordt behoord de cardinaliteit van de feature certificate te veranderen van <0,1> naar <1,1>. De ontwikkelde generator verandert de cardinaliteiten niet nadat de features een waarde hebben gekregen. Daardoor zijn de features van familierelatie niet herkend als verplichte features en is deze niet ingevuld.

    - **Het aantal niet ingevulde optionele features die wel gevuld moesten worden gezien het scenario**

      Er zijn geen optionele features die wel ingevuld moest worden volgens het scenario maar die geen waarde hebben. In de laatste lus werd alle optionele features gevraagd aan de eindgebruiker.

    - **Het aantal features die verkeerd zijn ingevuld**

      Er zijn 9 features die verkeerd zijn ingevuld waarbij 8 daarvan zijn ingevuld bij *chilInfo* en 1 ervan zijn ingevuld bij *childAddressInfo*. Volgens het scenario moet *childInfo* niet gevuld worden. Deze worden wel degelijk ingevuld door de generator wanneer de opgehaalde gegevens van de aanvrager weggeschreven worden naar het feature model. De generator schrijft bij ieder persoon deze gegevens weg. De generator behoort de verschillende persoonsgegevens te onderscheiden maar deze is nog niet geïmplementeerd in de generator. In de opgehaalde gegevens is er een element met naam number. Het feature huisnummer van huisnummergegevens heeft ook hetzelfde naam. De generator geeft hierdoor deze feature de waarde van dit element.

    - **Het aantal niet ingevulde features waarvan de waarden wel opgeslagen zijn**

      Er zijn 7 niet ingevulde features waarvan de waarden opgeslagen zijn en opgehaald zijn door de eMAXX Mid Office functie. De opgehaalde gegevens hebben een andere elementnaam dan het opgestelde feature model waardoor deze gegevens niet naar het feature model werden geschreven.

- ***Kapvergunning aanvraag***

    - **Het aantal niet ingevulde verplichte features**

      In het e-formulier worden alle verplichte features ingevuld en zijn er geen fouten van dit type. Dit gaat goed omdat in het feature model geen optionele keuze mogelijkheden voorkomen waarna de cardinaliteit ervan veranderd moeten worden om de features die aan deze keuzes behoren verplicht te maken.

    - **Het aantal niet ingevulde optionele features die wel gevuld moesten worden gezien het scenario**

      Ook hier worden de optionele features goed afgehandeld die in de laatste lus worden gevraagd.

    - **Het aantal features die verkeerd zijn ingevuld**

      Er zijn 19 aantal features die een verkeerde waarde hebben waarbij 18 hiervan zijn de persoonsgegevens van de gemachtigde en de eigenaar. Volgens het scenario moesten deze features niet ingevuld worden. Deze features krijgen een waarde omdat de generator de opgehaalde persoonsgegevens van de aanvrager wegschrijft bij alle personen. De andere is het feature huisnummer die een verkeerde waarde krijgt omdat er in de opgehaalde gegevens een element de zelfde naam heeft.

    - **Het aantal niet ingevulde features waarvan de waarden wel opgeslagen zijn**

      Het aantal niet ingevulde features waarvan de waarden wel opgeslagen zijn is 7. Dit zijn weer dezelfde features die bij het *uittreksel aanvraag* e-formulier ook niet ingevuld werden. De reden is hetzelfde als bij *uittreksel aanvraag* e-formulier het.

7.3.2 Efficiëntie van de nieuwe ontwikkeltechniek

De efficiëntie van de nieuwe ontwikkeltechniek wordt bepaald door de benodigde tijd voor het opstellen van de e-formulieren. Voor het opstellen van een e-formulier moeten het feature model en de bijbehorende lijst van functies opgesteld worden. De resultaten van deze test is in Tabel 7.3 weergegeven. De linkerkolom geeft aan om welke e-formulier het gaat. In de tweede kolom worden de benodigde tijden voor het opstellen van de feature modellen van de e-formulieren weergegeven. In de derde kolom worden de benodigde tijden voor het opstellen van de lijsten van functies van de e-formulieren weergegeven. In de laatste kolom worden de totale tijden voor het opstellen van de e-formulieren weergegeven.

	Feature model	**Lijst van functies**	**Totaal**
Uittreksel aanvraag	2 uur	50 min	2 uur 50 min
Kapvergunning aanvraag	2 uur	15 min	2 uur 15 min

Tabel 7.3: kwalitatieve resultaten bij het opstellen van e-formulieren met de nieuwe ontwikkeltechniek

De reden dat het opstellen van lijsten van functies voor uittreksel aanvraag langer duurt dan kapverginning aanvraag komt doordat bij uittreksel aanvraag op meerdere plekken een funcie aangeroepen kan worden. Elke functie die aangeroepen kan worden moet in de lijst worden opgenomen.

7.4 Vergelijkingen

In deze sectie zullen de effectiviteit en de efficiëntie van de twee ontwikkeltechnieken worden besproken aan de hand van de resultaten. In sectie 7.4.1 zullen de effectiviteit van de ontwikkeltechnieken met elkaar worden vergeleken. In sectie 7.4.2 zullen de efficiëntie van de ontwikkeltechnieken met elkaar worden vergeleken.

7.4.1 Vergelijkingen effectiviteit van de ontwikkeltechnieken

Zoals eerder is aangegeven zijn de e-formulieren met de huidige ontwikkeltechniek al eerder opgesteld en getest. Deze e-formulieren werken prima en zijn hierdoor deze niet nog eens getest. De e-formulieren zijn eerst opgesteld voor deze generator en vervolgens zijn deze e-formulieren ingevuld met behulp van de generator. Eerder in Tabel 7.2 zijn de resultaten hiervan weergegeven. Het was al bekend dat de huidige ontwikkeltechniek effectief was. De nieuwe ontwikkeltechniek is nog niet effectief. Deze heeft nog aantal problemen. Gezien de korte tijd die er was is de generator van de nieuwe ontwikkeltechniek niet goed getest. Er zitten nog aantal bugs in die eruit gehaald moeten worden. Bij het ontwikkelen van e-formulieren met huidige ontwikkeltechniek worden ook getest en behoort de generator ook getest te worden.

7.4.2 Vergelijkingen efficiëntie van de ontwikkeltechnieken

In Tabel 7.1 zijn de benodigde tijden voor het opstellen van de e-formulieren met de huidige ontwikkeltechniek weergegeven. In Tabel 7.3 zijn de benodigde tijden voor het opstellen van de e-formulieren met de nieuwe ontwikkeltechniek weergegeven. De resultaten laten zien dat het opstellen van e-formulieren met de nieuwe ontwikkeltechniek veel sneller gaat dan de huidige ontwikkeltechniek. Dit komt doordat de huidige ontwikkeltechniek niet alleen de velden voor de benodigde gegevens op de webpagina's worden geïmplementeerd, maar ook ervoor moeten zorgen dat deze gegevens op de juiste wijze worden ingelezen en weggeschreven worden naar het rapport. Bij het opstellen van e-formulieren met de nieuwe ontwikkeltechniek worden alleen de featurs van de benodigde gegevens gecreëerd. Er hoeft verder geen acties uitgevoerd te worden om gegevens op te halen en deze weg te schrijven naar het rapport omdat deze automatisch gaat.

8 Conclusies en aanbevelingen

In dit hoofdstuk worden de conclusies en aanbevelingen van dit onderzoek weergegeven. In sectie 8.1 wordt de conclusie weergegeven en in sectie 8.2 worden de aanbevelingen weergegeven.

8.1 Conclusies

In dit afstudeeronderzoek is onderzocht hoe webpagina's van e-formulieren automatisch gegenereerd kunnen worden gebruikmakend van modellen. De indruk bestaat namelijk dat het werken met generatoren tijd en geld wordt bespaard dan het handmatig implementeren van deze webpagina's.

In dit onderzoek zijn drie generatoren ontwikkeld die van elkaar verschillen in complexiteit en ieder een ander probleem oplost. Deze generatoren genereren webpagina's van e-formulieren aan de hand van datamodellen. De model gedreven e-formulier generator zonder interactie genereert één webpagina waarop alle benodigde gegevens aan de eindgebruiker wordt gevraagd. De model gedreven e-formulier generator met ondersteuning van eMAXX Mid Office roept een eMAXX Mid Office functie aan waarbij een interactie met de eMAXX Mid Office plaatsvindt waarmee er opgeslagen gegevens worden opgehaald en vervolgens wordt één webpagina gegenereerd waarop de resterende gegevens worden gevraagd. De model gedreven e-formulier generator met workflow voert meerdere keren een interactie met eMAXX Mid Office (indien nodig) en genereert meerdere webpagina's (indien nodig) waarbij ervoor probeert te zorgen dat de eindgebruiker zo min mogelijk gegevens hoeft in te vullen. Aangezien de laatste generator het meest complexe en het meest interessantste generator is, is deze generator gebruikt om de nieuwe ontwikkeltechniek te getest/geëvalueerd op effectiviteit en efficiëntie. De huidige ontwikkeltechniek is ook getest/geëvalueerd op effectiviteit en op efficiëntie om ze met elkaar te kunnen vergelijken.

Gezien de resultaten qua effectiviteit en de redenen waarom de nieuwe ontwikkeltechniek nog niet helemaal goed werkt, zal de generator uitvoerig getest moeten worden waarbij de bugs eruit worden gehaald. Dit is meer een implementatie probleem dan de opgestelde nieuwe ontwikkeltechniek.

Gezien de resultaten qua efficiëntie is het inderdaad tijd besparend, wanneer de e-formulieren met de nieuwe ontwikkeltechniek worden opgesteld dan deze handmatig te implementeren met huidige ontwikkeltechniek. Wel moet er geattendeerd worden dat de gegenereerde webpagina's standaard uitzien. Wanneer er verschillende klanten verschillende behoeften hebben qua lay-out, dan kan het zijn dat het implementeren van deze verschillen tussen klanten extra tijd gaat kosten. Het kan zijn dat deze extra tijd ervoor zorgt dat er meer tijd gaat kosten om deze e-formulieren op te stellen en wijzigingen aanbrengen aan de generator in de nieuwe ontwikkeltechniek dan handmatig implementeren van deze e-formulieren inde huidige ontwikkeltechniek. Er moet goed gekeken worden naar de behoeften van klanten of deze niet al te veel van elkaar verschillen waardoor het werken met de generator tijd besparend kan zijn.

De generator is nog niet rijp genoeg om meteen commercieel te gebruiken. Eerst zal de generator uitvoerig getest worden waarbij de bugs eruit gehaald worden. Vervolgens moet er nog een en al aan het uiterlijk van de gegenereerde webpagina's wat gebeuren, om deze te verfraaien. Het is dan goed als er een onderzoek wordt gedaan naar het instellen van het uiterlijk van de gegenereerde webpagina's. Daarbij moet er gekeken worden hoe dit uiterlijk makkelijk veranderbaar is voor verschillende klanten.

Dit onderzoek toont aan dat aan de hand van datamodellen automatisch webpagina's van e-formulieren gegenereerd kunnen worden. Werken met deze generator kan ervoor zorgen dat er geld en tijd worden gespaard voor het opstellen van e-formulieren.

Ik vind dat dit onderzoek een start zal maken voor het verder ontwikkelen van de model gedreven e-formulier generator. Dit onderzoek geeft een nieuwe kijk in het genereren van webpagina's van e-formulieren. Zoals eerder aangegeven zijn de huidige generatoren meer van statische aard. De ontwikkelde generator in dit onderzoek is dynamisch waarbij er interacties worden uitgevoerd met systemen of eindgebruikers. In de toekomst zullen er meer van dit soort generatoren worden ontwikkeld.

In dit onderzoek zijn een aantal deelvragen beantwoord die hieronder kort worden besproken.

Wat is het datamodel?

Het invullen van een e-formulier voor een aanvraag kan van burger tot burger verschillen. Het op te stellen datamodel moet hiermee rekening kunnen houden. Feature modeling kan gebruikt worden om de gemeenschappelijke en variabele delen van het datamodel weer te geven. De feature modeling techniek voldoet verder aan de eisen die in sectie 1.3 zijn opgesteld. In overleg met eMAXX is er voor deze modelleringtechniek gekozen.

Hoe wordt het datamodel gerepresenteerd?

Het datamodel wordt opgesteld met behulp van het programma *XFeature*. Dit programma slaat het opgestelde datamodel in een XML bestand. Het datamodel wordt in syntax gerepresenteerd.

Hoe wordt het datamodel getransformeerd naar webpagina's?

Voor het ontwikkelen van de generator is er gebruik gemaakt van MDE ontwikkeltechniek omdat deze ontwikkeltechniek belooft dat de inspanning voor het ontwikkelen en onderhouden van systemen gereduceerd kan worden door te werken met modellen in plaats van code. De transformaties die in dit onderzoek voorkomen maken gebruikt van het transformatie overzicht, die in sectie 3.1.3 is weergegeven. De transformatieregels zijn in XSL geschreven die XML bestanden omzet naar een ander formaat. Aangezien de opgestelde datamodellen in een XML bestand worden opgeslagen, is er voor dit gekozen om datamodellen te transformeren naar webpagina's.

Welke eMAXX Mid Office functionaliteiten zijn er te gebruiken?

De functie *getPersonDetails* is een functie van eMAXX Mid Office die in dit onderzoek is gebruikt. Verder zijn er functies zoals *lookUpPerson* en *lookUpAddress* die gebruikt kunnen worden maar die niet zijn gebruikt. In dit onderzoek is er meer gekeken of er gebruikt gemaakt kon worden van de functie van eMAXX Mid Office. Het is ook niet de bedoeling om alle functies uit te testen. Deze kunnen later onderzocht worden in een ander onderzoek.

Wanneer wordt er een functie gebruikt?

Er wordt een functie gebruikt wanneer er gegevens in het datamodel door een functie ingevuld kunnen worden; Bijvoorbeeld de functie *getPersonDetails* wordt gebruikt om persoons- en adresgegevens in het datamodel in te vullen.

Hoe wordt de volgorde van de functies bepaald?

Er wordt eerst een lijst opgesteld van mogelijke functies voor het desbetreffende datamodel. Vervolgens wordt de functie geselecteerd die uitgevoerd kan worden en die de meeste gegevens in het datamodel kan invullen. De uitgevoerde functie wordt uit het lijst verwijderd en wordt de volgende functie uitgevoerd.

Hoe wordt bepaald welke gegevens met elkaar samenhangen?

Aan de hand van selectiealgoritme wordt er bepaald welke gegevens met elkaar samenhangen. De ontwikkelde selectiealgoritmen bepalen niet welke gegevens met elkaar samenhangen zoals die in sectie 2.2.1 is besproken. Wel worden gegevens (features) geselecteerd op basis van hun eigenschappen. Hiervoor zijn er drie selectiealgoritmen ontwikkeld waarbij ieder gegevens (features) selecteert die andere eigenschappen hebben. In de volgende sectie zal er een aanbeveling worden gegeven met betrekking tot het bepalen van samenhangende gegevens.

Hoe worden deze samenhangende gegevens op dezelfde webpagina gezet?

Het selectiealgoritme dat de samenhangende gegevens (features) selecteert haalt de niet samenhangende gegevens uit het datamodel. Het resulterende datamodel bevat alleen gegevens die samenhangend zijn. Aan de hand van dit datamodel wordt een webpagina gegenereerd. Zo komen de samenhangende gegevens op dezelfde webpagina te staan.

8.2 Aanbevelingen

Voordat deze generator echt in productie wordt opgenomen, zal nog een en al aan de generator veranderd moeten worden. Hieronder worden aanbevelingen gegeven zodat deze generator in productie kan worden opgenomen.

- ***Automatisch genereren van webpagina waarin de producten staan***

  In de huidige werkwijze van de generator wordt een webpagina (die handmatig is geïmplementeerd) ingelezen waarin de producten staan die geselecteerd kunnen worden. Deze webpagina kan door de generator worden gegenereerd door gebruik te maken van *lookUpProducts* functie van de eMAXX Mid office. Deze functie haalt een lijst van producten op. Aan de hand van deze lijst kan de generator een webpagina genereren waarop deze producten staan en die geselecteerd kunnen worden door de eindgebruiker.

- ***Automatisch genereren van lijst van functies***

  De lijst van functies die in de ontwikkelde generator wordt gebruikt wordt handmatig opgesteld. De lijst van functies van een product kan automatisch gegenereerd worden. De invoer en de uitvoer van een functie staat op de server. Deze kunnen ingelezen worden en vergeleken worden met de bijbehorende feature model. Daarin wordt gekeken of er functies zijn die een deel van het feature model kunnen invullen. Zodra dit het geval is, dan wordt deze functie in de lijst van functies gezet. Als dit het niet geval is, dan wordt er naar de volgende functie gekeken. Dit geldt nu nog voor de eMAXX Mid Office functies. Als er andere functies zijn die gebruikt kunnen worden, dan kan van deze functies ook een soortgelijke opzet als die van de functies van de eMAXX Mid Office worden gedaan.

- ***Selectiealgoritme voor samenhangende gegevens***

  In vorige sectie is al aangegeven dat in de ontwikkelde generator de samenhangende gegevens niet geselecteerd worden zoals in sectie 2.2.1 is beschreven. Hiervoor zal een beter selectiealgoritme geschreven/onderzocht moeten worden die gebruikmakend van de structuur van het e-formulier samenhangende gegevens gaat selecteren.

- ***Gebruikte transformatie voor genereren webpagina kan algemener***

  Er worden meerdere keren gebruikt gemaakt van de transformatie T1 waarbij vanuit het feature model een webpagina wordt gegenereerd. Wanneer de webpagina's naar een andere JSP pagina moeten verwijzen, dan wordt er voor deze transformatie een andere XSL bestand gebruikt waarin de transformatieregel TR_1 net iets anders is. In deze transformatieregel staat waarnaar deze webpagina wordt verwezen. Dit verwijzen naar de volgende JSP pagina kan in de huidige JSP pagina waarin de webpagina wordt gegenereerd worden aangegeven. De transformatieregel TR_1 kan dan weggelaten worden en kan dit XSL bestand gebruikt worden voor elke processtap die een webpagina gaat genereren. Zo wordt er ruimte bespaard op de webserver. Wanneer er een wijziging wordt aangebracht aan dit XSL bestand, dan hoeft dit niet bij iedere XSL bestand, die webpagina's genereren, worden uitgevoerd.

- ***Uiterlijk van de webpagina's verfraaien***

  In dit onderzoek is er geen aandacht gegeven aan het uiterlijk van de gegenereerde webpagina. Het uiterlijk van deze webpagina's kunnen verfraaid worden door bijvoorbeeld gebruik te maken van CSS (Cascading Style Sheets). CSS is een techniek voor de stijl (vormgeving) van webpagina's. De informatie over de vormgeving wordt toegevoegd aan de HTML-code van het document [wik07p].

- ***Gebruik maken van ingevulde gegevens om webpagina's adaptiever te maken***

  In het onderzoek [Kui06] is aangegeven dat de webpagina's adaptiever gemaakt kan worden door gebruik te maken van gegevens van de eindgebruiker. Zo kan bijvoorbeeld de lettergrootte van een webpagina veranderd worden wanneer de leeftijd van de eindgebruiker bekend is. Bij oudere mensen kan de letters worden vergroot omdat ze wat moeilijker kunnen zien dan een persoon die jong is. Er moet een onderzoek komen hoe dit geïntegreerd kan worden in deze generator om adaptieve e-formulieren te genereren.

- ***Valideren van de ingevulde velden op een webpagina***

  In de webpagina's van de huidige e-formulieren worden sommige velden gevalideerd. Deze velden worden gevalideerd op de ingevulde waarde of deze correct zijn ingevuld zoals postcode bestaat uit vier cijfers en twee letters. De gegenereerde webpagina's hebben dit soort validatie niet. Er moet een onderzoek komen hoe de velden van de gegenereerde webpagina's gevalideerd kunnen worden. Een optie kan zijn dat de validatiewaarde bij de feature in het feature model wordt meegegeven en gebruikmakend van een nieuwe transformatieregel javascripts in de webpagina's genereren zodat de velden gevalideerd kunnen. Een ander optie is een lijst van de validatiewaarden opstellen en deze aan de transformatie meegeven om javascript in de webpagina te zetten. Nogmaals dit moet worden onderzocht wat de mogelijkheden zijn en wat de beste optie is.

Dankwoord

Ik wil mijn begeleiders Bedir Tekinerdogan en Mehmet Aksit van de Universiteit Twente, alsmede mijn begeleiders Anton Boerma en Richard Scholten van eMAXX B.V. bedanken voor de begeleiding tijdens mijn afstudeeropdracht.

Referenties

[Béz06] Jean Bézivin, Fabian Büttner, Martin Gogolla, Frédéric Jouault, Ivan Kurtev, Arne Lindow. Model Transformations? Transformation Models! **MoDELS2006**, 2006.

[Cla04] Clark, T., Evans, A., Sammut, P., Willans, J.: Applied Metamodelling - A Foundation for Language Driven Development. version 0.1, 2004.

[Cza98] This is a chapter from K. Czarnecki. Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models. Ph.D. thesis, Technische Universität Ilmenau, Germany, 1998.

[Cza04] K. Czarnecki, S. Helsen, U. Eisenecker. Staged Configuration Using Feature Models. In R. L. Nord, editor, Software Product Lines: Third International Conference, SPLC 2004, Boston, MA, USA, August 30-September 2, 2004. Proceedings, volume 3154 of Lecture Notes in Computer Science, pages 266-283, Springer-Varlag, 2004.

[Cza05] K. Czarnecki and C. H. P. Kim. Cardinality-based feature modelling and constraints: a progress report. In International Workshop on Software Factories, San Diego, California, Oct 2005. Paper available at http://softwarefactories.com/workshops/OOPSLA-2005/

[Cza06] K. Czarnecki, C. H. P. Kim and K. T. Kalleberg. Feature Models are Views on Ontologies. To appear in Software Product Line Conference (SPLC), Baltimore, USA, August 21-24, 2006, IEEE CS, 2006.

[Deu07] A. van Deursen, E. Visser, and J.Warmer. Model-driven software evolution: A research agenda. In D. Tamzalit, editor, CSMR Workshop on Model-Driven Software Evolution (MoDSE'07), pages 41–49, Amsterdam, The Netherlands, March 2007.

[DiC97] A.DiCaterino, K. Larsen, M. Tang and W. Wang. An Introduction to Workflow Management Sysytems, Center for Technology in Government, November 1997..

[Dig06] Digid: Home http://www.digid.nl/ (december 2006)

[Dig07] Vraag en antwoord/ Nummergebruik, wat is A-nummer? http://www.digid.nl/overheid/vraag-en-antwoord/vraag-en-antwoord/certificaat/ (mei 2007)

[Dle07] Gemeente Enschede (2007), Het Digitaal Loket. https://www.loket.enschede.nl/griffieren/p1/schermen.asp?schermid=1 (maart 2007)

[Ema06a] Statische Prototype van het doorgeven van verhuizing, X:\Persoonlijk\Pieternel\Documenten\Prototypes\Online Statisch\index.html (december 2006)

[Ema06b] Dynamische Prototype van het doorgeven van verhuizing, X:\Persoonlijk\Pieternel Kuiper\Documenten\Prototypes\Online Adaptatie\index.html (december 2006)

[Emo06] M.Veenhuis. eMAXX Mid Office Architectuur Blauwdruk versie 3.2, Enschede, 30 januari 2006.

[Fea04] M. Antkiewicz and K. Czarnecki, "FeaturePlugIn: Feature Modeling Plug-In for Eclipse", OOPSLA'04 Eclipse Technology eXchange (ETX) Workshop, 2004. http://www.swen.uwaterloo.ca/~kczarnec/etx04.pdf (oktober 2007)

[Gde07] Gemeente Deventer (2007), Ja, ik doe aangifte van mijn verhuizing binnen Deventer.

https://forms.deventer.nl/_layouts/FormServer.aspx?DefaultItemOpen=1&XsnLocation=/FormServerTemplates/Aanvraag%verhuizing%binnen%Gemeente.xsn (maart 2007)

[Gho07] Gemeente Hoogeveen (2007), Verhuizen binnen de gemeente Formulier: Zo geeft u een adreswijziging door . http://www.hoogeveen.nl/Formulier.aspx?ProductID=8676&FormID=60 (maart 2007)

[Kan90] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Nov. 1990.

[Kui06] P.M. Kuiper. Adaptieve gemeentelijke e-formulieren. ..Universiteit Twente, Enschede, Sep. 2006.

[Mes07] Mesbah & A. van Deursen. Migrating Multi-page Web Applications to Single-page Ajax Interfaces. In Proc.11th Euro. Conference on Software Maintenance and Reengineering (CSMR). 2007.

[Pur06] Pure::variants Eclipse Plug-in User's Guide Version 2.0 for pure::variants 2.2, 2006. http://www.pure-systems.com/fileadmin/downloads/pure-variants/doc/pv-user-manual.pdf (oktober 2007)

[Ros06] S. Roser and B. Bauer. An approach to automatically generated model transformations using ontology engineering space. In Proceedings of Workshop on Semantic Web Enabled Software Engineering (SWESE), Athens, GA, U.S.A., 2006.

[SEA06] XSD (XML Schema Definition) http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci831325,00.html (december 2006)

[Sei] http://www.sei.cmu.edu/domain-engineering/domain_model.html

[W3C06] http://www.w3.org/XML/Schema

[Wan05] W. Wang. Evaluation of UML Model Transformation Tools. Technical University of Vienna, Business Informatics Group, Master Thesis, 2005.

[Wik06a] XML (Extensible Markup Language) http://nl.wikipedia.org/wiki/Extensible_Markup_Language (december 2006)

[Wik06b] XSLT (Extensible Stylesheet Language Transformations) http://nl.wikipedia.org/wiki/Extensible_Stylesheet_Language_Transformations (december 2006)

[Wik06c] SOAP (Simple Object Access Protocol) http://nl.wikipedia.org/wiki/Simple_Object_Access_Protocol (december 2006)

[Wik07a] BSN (Burgerservicenummer) http://nl.wikipedia.org/wiki/BSN (mei 2007)

[Wik07b] Sofinummer (sociaal-fiscaal nummer) http://nl.wikipedia.org/wiki/Sofinummer (mei 2007)

[Wik07c] HTTP (Hypertext Transfer Protocol) http://nl.wikipedia.org/wiki/Http (mei 2007)

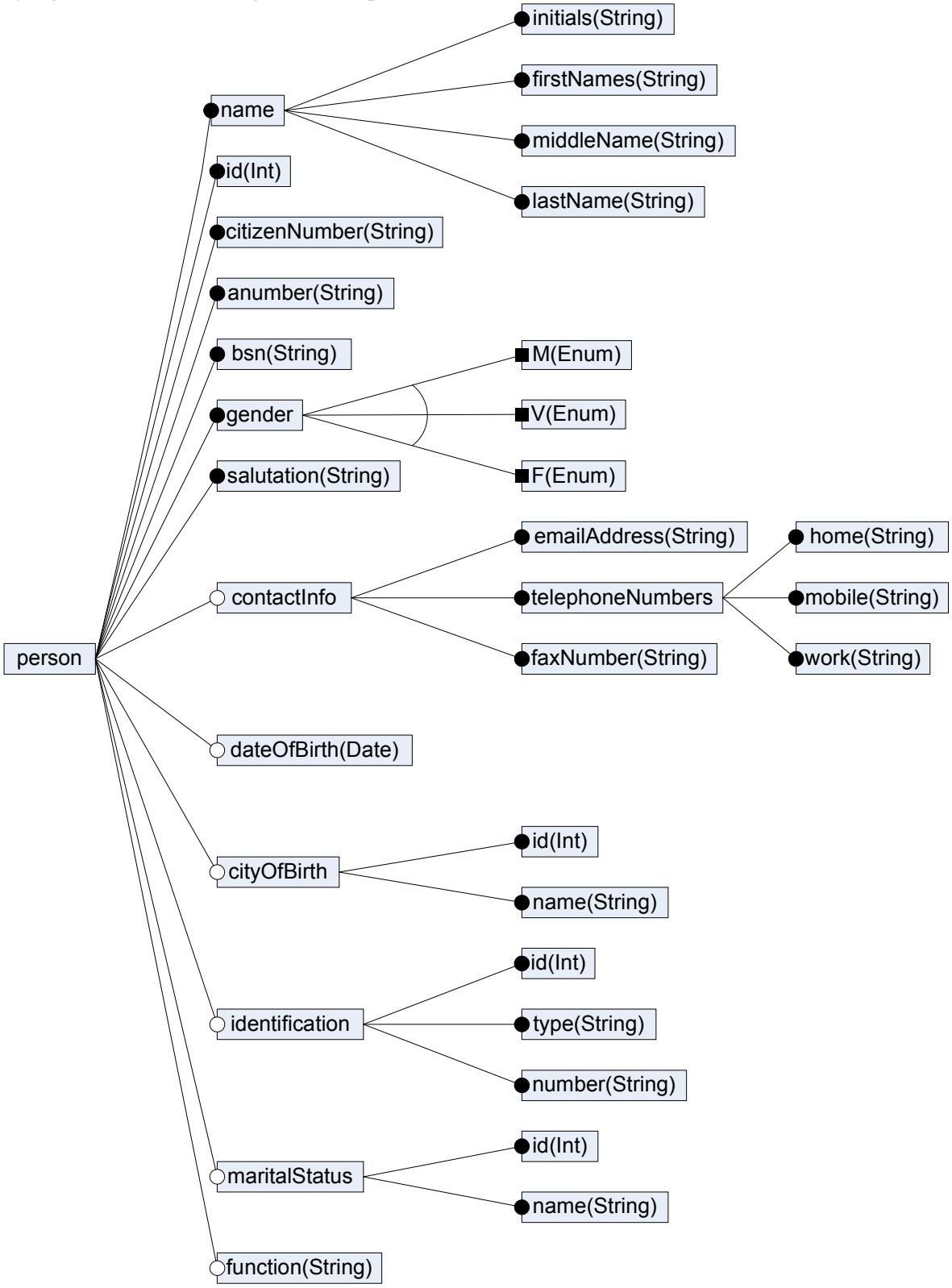
[Wik07d] SMTP (Simple Mail Transfer Protocol) http://nl.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol (mei 2007)

[Wik07e] HTTPS (HTTP Secure) http://nl.wikipedia.org/wiki/HTTPS (mei 2007)

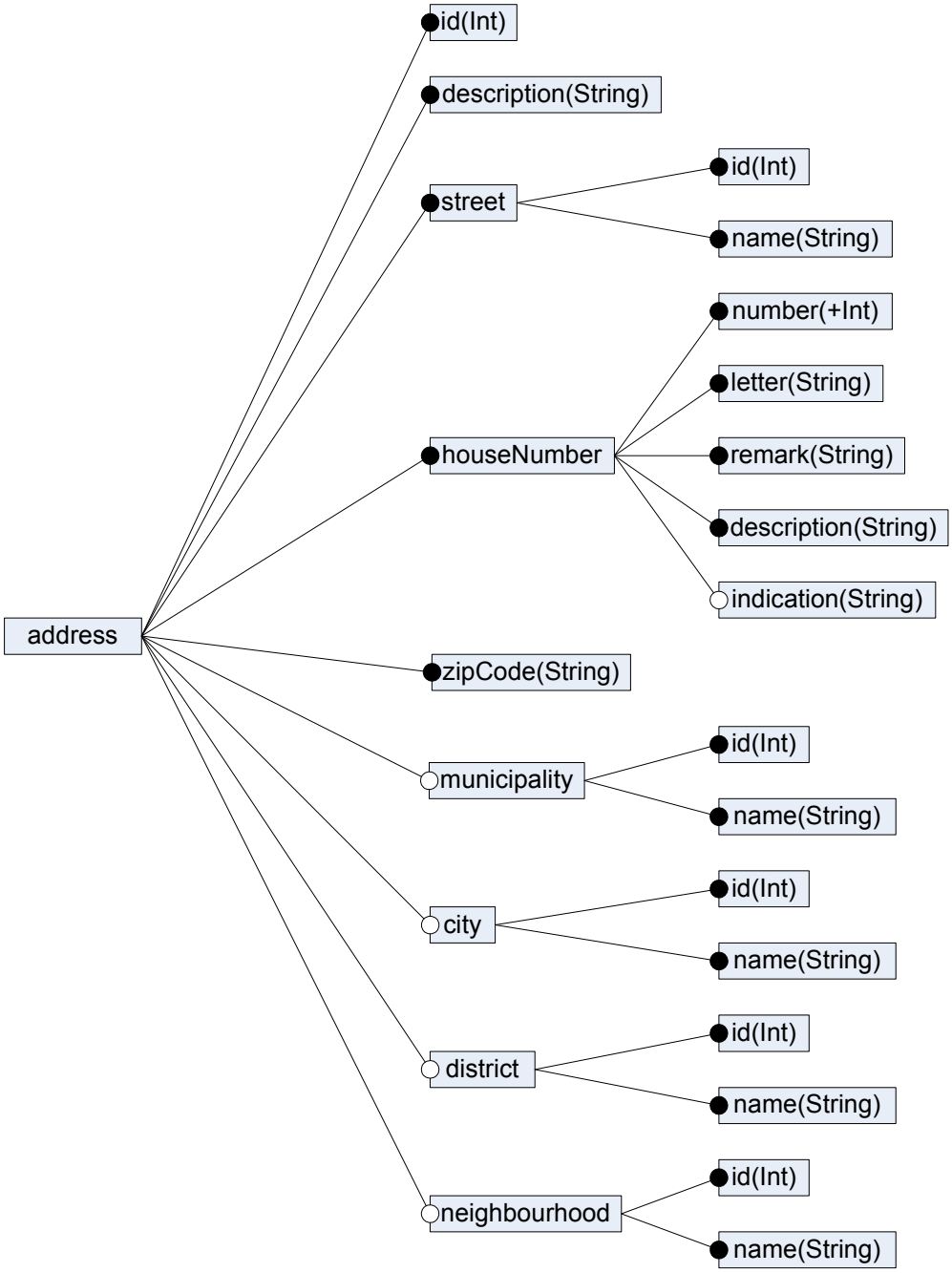
[Wik07f] FTP (File Transfer Protocol)
http://nl.wikipedia.org/wiki/File_Transfer_Protocol (mei 2007)
[Wik07g] JSP (JavaServer Pages)
http://nl.wikipedia.org/wiki/JSP (mei 2007)
[Wik07h] GBA (Gemeentelijke Basisadministratie Persoonsgegevens)
http://nl.wikipedia.org/wiki/Gemeentelijke_Basisadministratie_Persoonsgegevens (mei 2007)
[Wik07i] MOF (Meta-Object Facility)
http://en.wikipedia.org/wiki/Meta-Object_Facility (augustus 2007)
[Wik07j] Eclipse
http://nl.wikipedia.org/wiki/Eclipse_(software) (augustus 2007)
[Wik07k] OMG (Object Management Group)
http://nl.wikipedia.org/wiki/Object_Management_Group (september 2007)
[Wik07l] HTML (HyperText Markup Language)
http://nl.wikipedia.org/wiki/Html (september 2007)
[Wik07m] SOA (Service-Oriented Architecture)
http://nl.wikipedia.org/wiki/Service-ori%C3%ABntatie (september 2007)
[Wik07n] SPL (Software Product Lines)
http://en.wikipedia.org/wiki/Software_product_lines (september 2007)
[Wik07o] XSL (Extensible Stylesheet Language)
http://nl.wikipedia.org/wiki/XSL (september 2007)
[Wik07p] CSS (Cascading Style Sheets)
http://nl.wikipedia.org/wiki/Cascading_Style_Sheets (november 2007)
[Xfe05] A. Pasetti and O. Rohlik. Technical note on a concept for the xfeature tool. Technical report, P and P Software GmbH / ETH-Zurich, 2005. http://www.pnp-software.com/XFeature/pdf/XFeatureToolConcept.pdf (oktober 2007)

Bijlagen

Bijlage A: feature diagrammen *person* en *address*



Figuur 8.1: feature diagram person



Figuur 8.2: feature diagram address

Bijlage B: inhoud XML bestand van het feature model *doorgeven verhuizing*

```
1.    <?xml version="1.0" encoding="UTF-8"?>
2.    <fm:FeatureModel fm:value="verhuizing" xmlns:fm ="http://www.pnp-
      software.com/XFeature/fmm xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.pnp-software.com/XFeature/fmm C:\Tools\eclipse-SDK-
      3.2.1\plugins\com.pnp.xfeature_0.9.8\src\FMP_configuration_files\FMP.xfmm">
3.        <fm:RootFeature fm:value="movement">
4.            <fm:SolitaryFeature fm:value="applicant">
5.                <fm:Annotation fm:value="Annotation">
6.                    <fm:Description fm:value="Aanvrager"/>
7.                </fm:Annotation>
8.                <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
9.                <fm:SolitaryFeature fm:value="oldAddress">
10.                   <fm:Annotation fm:value="Annotation">
11.                       <fm:Description fm:value="Huidige adres"/>
12.                   </fm:Annotation>
13.                   <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
14.                   <fm:SolitaryReference fm:value="refAddress">
15.                       <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
16.                   </fm:SolitaryReference>
17.               </fm:SolitaryFeature>
18.               <fm:SolitaryReference fm:value="refPerson">
19.                   <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
20.               </fm:SolitaryReference>
21.           </fm:SolitaryFeature>
22.           <fm:SolitaryFeature fm:value="referenceNumber">
23.               <fm:Annotation fm:value="Annotation">
24.                   <fm:Description fm:value="Referentienummer"/>
25.               </fm:Annotation>
26.               <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
27.               <fm:Attribute fm:value="Attribute">
28.                   <fm:String fm:value="String"/>
29.               </fm:Attribute>
30.           </fm:SolitaryFeature>
31.           <fm:SolitaryFeature fm:value="dateOfMovement">
32.               <fm:Annotation fm:value="Annotation">
33.                   <fm:Description fm:value="Verhuisdatum "/>
34.               </fm:Annotation>
35.               <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
36.               <fm:Attribute fm:value="Attribute">
37.                   <fm:String fm:value="String"/>
38.               </fm:Attribute>
39.           </fm:SolitaryFeature>
40.           <fm:SolitaryFeature fm:value="typeOfMovement">
41.               <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
42.               <fm:FeatureGroup fm:value="FeatureGroup">
43.                   <fm:Annotation fm:value="Annotation">
44.                       <fm:Description fm:value=" Ik bewoon het nieuwe adres:"/>
45.                   </fm:Annotation>
46.                   <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
47.                   <fm:GroupedFeature fm:value="independant">
48.                       <fm:Attribute fm:value="Attribute">
49.                           <fm:String fm:value="String">
50.                               <fm:StringProperties fm:value="StringProperties">
51.                                   <fm:StringValue fm:value="Zelfstandig"/>
52.                               </fm:StringProperties>
53.                           </fm:String>
54.                       </fm:Attribute>
55.                   </fm:GroupedFeature>
56.                   <fm:GroupedFeature fm:value="lodging">
57.                       <fm:Attribute fm:value="Attribute">
58.                           <fm:String fm:value="String">
59.                               <fm:StringProperties fm:value="StringProperties">
60.                                   <fm:StringValue fm:value="Kamerverhuur"/>
61.                               </fm:StringProperties>
62.                           </fm:String>
63.                       </fm:Attribute>
64.                   </fm:GroupedFeature>
65.                   <fm:GroupedFeature fm:value="resident">
```

```
66.                         <fm:Attribute fm:value="Attribute">
67.                             <fm:String fm:value="String">
68.                                 <fm:StringProperties fm:value="StringProperties">
69.                                     <fm:StringValue fm:value="Inwonend"/>
70.                                 </fm:StringProperties>
71.                             </fm:String>
72.                         </fm:Attribute>
73.                         <fm:SolitaryFeature fm:value="assentingInhabitant">
74.                             <fm:Annotation fm:value="Annotation">
75.                                 <fm:Description fm:value="Hoofdbewoner"/>
76.                             </fm:Annotation>
77.                             <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
78.                             <fm:SolitaryReference fm:value="refPerson">
79.                                 <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
80.                             </fm:SolitaryReference>
81.                         </fm:SolitaryFeature>
82.                     </fm:GroupedFeature>
83.                 </fm:FeatureGroup>
84.             </fm:SolitaryFeature>
85.             <fm:SolitaryFeature fm:value="fellowApplicants">
86.                 <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
87.                 <fm:SolitaryFeature fm:value="fellowApplicant">
88.                     <fm:Annotation fm:value="Annotation">
89.                         <fm:Description fm:value="Medeverhuizer"/>
90.                     </fm:Annotation>
91.                     <fm:Cardinality fm:cardMax="*" fm:cardMin="0"/>
92.                     <fm:SolitaryReference fm:value="refPerson">
93.                         <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
94.                     </fm:SolitaryReference>
95.                 </fm:SolitaryFeature>
96.             </fm:SolitaryFeature>
97.             <fm:SolitaryFeature fm:value="previouwInhabitant">
98.                 <fm:Annotation fm:value="Annotation">
99.                     <fm:Description fm:value="Gegevens vorige inwoner"/>
100.                </fm:Annotation>
101.                <fm:Cardinality fm:cardMax="1" fm:cardMin="0"/>
102.                <fm:SolitaryReference fm:value="refPerson">
103.                    <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
104.                </fm:SolitaryReference>
105.                <fm:SolitaryReference fm:value="refAddress">
106.                    <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
107.                </fm:SolitaryReference>
108.            </fm:SolitaryFeature>
109.            <fm:SolitaryFeature fm:value="newAddress">
110.                <fm:Annotation fm:value="Annotation">
111.                    <fm:Description fm:value="Nieuwe adres"/>
112.                </fm:Annotation>
113.                <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
114.                <fm:SolitaryReference fm:value="refAddress">
115.                    <fm:Cardinality fm:cardMax="1" fm:cardMin="1"/>
116.                </fm:SolitaryReference>
117.            </fm:SolitaryFeature>
118.        </fm:RootFeature>
119.    </fm:FeatureModel>
```

Bijlage C: Java code van het selectiealgoritme ObligatoryFeatureSelect

```
1.    package com.emaxx.eformgenerator;
2.    import java.io.IOException;
3.    import javax.xml.parsers.ParserConfigurationException;
4.    import javax.xml.transform.dom.DOMResult;
5.    import javax.xml.xpath.XPathExpressionException;
6.    import org.w3c.dom.Document;
7.    import org.w3c.dom.Node;
8.    import org.xml.sax.SAXException;
9.
10.   public class ObligatoryFeatureSelect
11.   {
12.      private static int counter ;
13.      private static int MaxFeatures;
14.      private static int counterFeatures;
15.      private static Node resultNode;
16.      public static DOMResult returnFM(DOMResult FM, Document CallList)throws
17.      ParserConfigurationException, SAXException, IOException, XPathExpressionException
18.      {
19.         counter = 0;
20.         counterFeatures = 0;
21.         MaxFeatures = 10;
22.         //Document dList = (Document) FM.getNode();
23.         DOMResult TempFM = new DOMResult();
24.         TempFM.setNode(FM.getNode().cloneNode(true));
25.         searchFM(TempFM.getNode(), CallList.getDocumentElement());
26.         System.out.print(counter);
27.         return TempFM;
28.      }
29.
30.      //Dit selecteerd de features die naar de webpagina worden gegenereerd.
31.      //De gene die niet gegenereerd zullen worden, worden hier verwijderd.
32.
33.      public static void searchFM(Node doc, Node list)
34.      {
35.         if (doc.hasChildNodes())
36.         {
37.            for (int i=doc.getChildNodes().getLength() -1; i>=0 ; i--)
38.            {
39.               if( "RootFeature".equals(doc.getChildNodes().item(i).getLocalName()))
40.               {
41.                  searchFM(doc.getChildNodes().item(i), list);
42.               }
43.               else
      if(doc.getChildNodes().item(i).getNodeName().equals("fm:FeatureModel"))
44.               {
45.                  searchFM(doc.getChildNodes().item(i), list);
46.               }
47.               else if
48.               (doc.getChildNodes().item(i).getNodeName().equals("fm:SolitaryFeature"))
49.               {
50.                  boolean checkMandatoryF =
51.                  checkMandatoryOfFeature(doc.getChildNodes().item(i));
52.                  boolean testChangeSF = checkAttributeOfSF(doc.getChildNodes().item(i));
53.                  boolean testChildFeature =
54.                  checkNodeForChildFeatures(doc.getChildNodes().item(i));
55.                  String sPathName = featurePathName(doc.getChildNodes().item(i));
56.                  if((!(checkMandatoryF)))
57.                  {
58.                     doc.removeChild(doc.getChildNodes().item(i));
59.                  }
60.                  else if (testChildFeature)
61.                  {
62.                  searchFM(doc.getChildNodes().item(i), list);
63.                  }
64.                  else if (testChangeSF)
65.                  {
66.                     doc.removeChild(doc.getChildNodes().item(i));
67.                  }
```

```
68.                  else
69.                  {
70.                     boolean checkFeatureInCallList = checkNodeInList( sPathName,list);
71.                     if (checkFeatureInCallList)
72.                     {
73.                        doc.removeChild(doc.getChildNodes().item(i));
74.                     }
75.                     else
76.                     {
77.                        counterFeatures = counterFeatures + 1;
78.                        if (counterFeatures > MaxFeatures )
79.                        {
80.                           doc.removeChild(doc.getChildNodes().item(i));
81.                        }
82.                     }
83.                  }
84.               }
85.               else if
      (doc.getChildNodes().item(i).getNodeName().equals("fm:FeatureGroup"))
86.               {
87.                  boolean checkMandatory =
88.                  checkMandatoryOfFeature(doc.getChildNodes().item(i));
89.                  if(!(checkMandatory))
90.                  {
91.                     doc.removeChild(doc.getChildNodes().item(i));
92.                  }
93.                  else
94.                  {
95.                     int GroupedFeatureCounter = 0;
96.                     for(int k = doc.getChildNodes().item(i).getChildNodes().getLength()-1;
97.                     k >= 0 ; k--)
98.                     {
99.                        if (doc.getChildNodes().item(i).getChildNodes().item(k).
100.                       getNodeName().equals("fm:GroupedFeature"))
101.                       {
102.                          GroupedFeatureCounter = GroupedFeatureCounter + 1;
103.                          boolean checkChildFeature =
104.                          checkNodeForChildFeatures(doc.getChildNodes().item(i).
105.                          getChildNodes().item(k));
106.                          if (checkChildFeature)
107.                          {
108.                             searchFM(doc.getChildNodes().item(i).getChildNodes().item(k),
109.                             list);
110.                          }
111.                       }
112.                    }
113.                    if (GroupedFeatureCounter > 1)
114.                    {
115.                       counterFeatures = counterFeatures + 1;
116.                       if (counterFeatures > MaxFeatures )
117.                       {
118.                          doc.removeChild(doc.getChildNodes().item(i));
119.                       }
120.                    }
121.                    else
122.                    {
123.                       doc.removeChild(doc.getChildNodes().item(i));
124.                    }
125.                 }
126.              }
127.
128.              else
129.              {
130.                 // do nothing
131.              }
132.           }
133.        }
134.
135.     }
136.
137.
```

```
138.
139.     // Dit functie kijkt of de checkNode fm:Cardinality als kind heeft.
140.     // Zo ja, dan wordt naar de waarde van de attribute cardMin gekeken.
141.     // Zodra cardMin ==1 dan geeft dit functie boolean true terug en anders boolean
      false
142.        terug.
143.
144.     public static boolean checkMandatoryOfFeature(Node node)
145.     {
146.        boolean checkCardinality;
147.        checkCardinality = true;
148.        for(int j= node.getChildNodes().getLength() - 1; j >= 0 ; j--)
149.        {
150.           if (node.getChildNodes().item(j).getNodeName().equals("fm:Cardinality"))
151.           {
152.              if (node.getChildNodes().item(j).getAttributes().
153.              getNamedItem("fm:cardMin").getNodeValue().equals("1"))
154.              {
155.                 checkCardinality = true;
156.              }
157.              else
158.              {
159.                 checkCardinality = false;
160.              }
161.           }
162.           else {}
163.        }
164.        return checkCardinality;
165.     }
166.
167.     //Dit kijkt of de waarde van een gegeven node is veranderd.
168.     //Zodra de waarde is veranderd, dan geeft dit functie true anders false op.
169.
170.     public static boolean checkAttributeOfSF(Node node)
171.     {
172.        boolean checkChangeAttribute = false;
173.        for(int j= node.getChildNodes().getLength()-1; j >= 0 ; j--)
174.        {
175.           if (node.getChildNodes().item(j).getNodeName().equals("fm:Attribute"))
176.           {
177.              Node childAttribute = node.getChildNodes().item(j).getFirstChild();
178.              if(("").equals(childAttribute.getAttributes().
179.                 getNamedItem("fm:value").getNodeValue()) ||
180.                 (" ").equals(childAttribute.getAttributes().
181.                 getNamedItem("fm:value").getNodeValue()) ||
182.                 ("String").equals(childAttribute.getAttributes().
183.                 getNamedItem("fm:value").getNodeValue()) ||
184.                 ("Integer").equals(childAttribute.getAttributes().
185.                 getNamedItem("fm:value").getNodeValue()) ||
186.                 ("Float").equals(childAttribute.getAttributes().
187.                 getNamedItem("fm:value").getNodeValue())                          )
188.              {
189.                 checkChangeAttribute = false;
190.              }
191.              else
192.              {
193.                 checkChangeAttribute = true;
194.                 return true;
195.              }
196.           }
197.        }
198.        return checkChangeAttribute;
199.     }
200.
201.     //Dit kijkt of de kinderen van de opgegeven node een feature is.
202.     //Zodra de node een feature als kind heeft, geeft dit true, anders false op.
203.
204.     public static boolean checkNodeForChildFeatures(Node node)
205.     {
206.        boolean ChildFeature = false;
207.        for(int j= node.getChildNodes().getLength()-1; j>=0; j--)
```

```
208.        {
209.           if (node.getChildNodes().item(j).getNodeName().equals("fm:SolitaryFeature") ||
210.              node.getChildNodes().item(j).getNodeName().equals("fm:RootFeature") ||
211.              node.getChildNodes().item(j).getNodeName().equals("fm:FeatureGroup") ||
212.              node.getChildNodes().item(j).getNodeName().equals("fm:FeatureModel") )
213.           {
214.              ChildFeature = true;
215.
216.           }
217.        }
218.        return ChildFeature;
219.     }
220.
221.     //Dit kijkt of de opgegeven string voorkomt in de attribuut van de feature die in de
222.     output element van de opgegeven node staat. Zodra dit string voorkomt, geeft dit
       true
223.     anders false op.
224.
225.     public static boolean checkNodeInList(String string, Node node)
226.     {
227.        boolean result = false;
228.        if (node.hasChildNodes())
229.         {
230.            for(int i = node.getChildNodes().getLength() -1; i >= 0; i--)
231.            {
232.               String st2 = node.getChildNodes().item(i).getNodeName();
233.               if (node.getChildNodes().item(i).getNodeType() == Node.ELEMENT_NODE)
234.               {
235.                  if ("output".equals(node.getChildNodes().item(i).getNodeName()))
236.                  {
237.                     for (int j = node.getChildNodes().item(i).getChildNodes().
238.                          getLength()-1; j >= 0; j--)
239.                     {
240.                          if ( node.getChildNodes().item(i)!= null &&
241.                                node.getChildNodes().item(i).getChildNodes().item(j) !=
242.                                null &&  node.getChildNodes().item(i).getChildNodes().
243.                                item(j).getAttributes() != null &&
244.                                node.getChildNodes().item(i).getChildNodes().item(j).
245.                                getAttributes().getNamedItem("pathname") != null &&
246.                                node.getChildNodes().item(i).getChildNodes().item(j).
247.                                getAttributes().getNamedItem("pathname").getNodeValue() !=
248.                                null)
249.                           {
250.                           String st = (node.getChildNodes().item(i).getChildNodes().
251.                           item(j).getAttributes().getNamedItem("pathname").
252.                           getNodeValue());
253.                           if (string.equals(node.getChildNodes().item(i).
254.                                 getChildNodes().item(j).getAttributes().
255.                                 getNamedItem("pathname").getNodeValue()))
256.                           {
257.                              result = true;
258.                              return result;
259.                           }
260.                           }
261.                     }
262.
263.                  }
264.                  else
265.                  {
266.                        result = checkNodeInList(string, node.getChildNodes().item(i));
267.                        if(result)
268.                        {
269.                           return true;
270.                        }
271.                    }
272.               }
273.
274.            }
275.         }
276.        return result;
277.     }
```

```
278.     //dit geeft een string terug wat de hele pathnaam tot aan de root parent weer.
279.
280.     public static String featurePathName(Node node)
281.     {
282.         StringBuffer result = new StringBuffer();
283.          Node temp = node;
284.          while (temp != null)
285.          {
286.              if ( temp.getAttributes() != null &&
287.                    temp.getAttributes().getNamedItem("fm:value") != null &&
288.                    temp.getAttributes().getNamedItem("fm:value").getNodeValue() != null)
289.              {
290.                 result.insert(0,temp.getAttributes().getNamedItem("fm:value").
291.                 getNodeValue());
292.              }
293.              if (temp.getParentNode() != null)
294.              {
295.                 result.insert(0, "/");
296.              }
297.             temp = temp.getParentNode();
298.          }
299.          String resultString = result.toString();
300.          resultString = resultString.substring(1);
301.          return resultString;
302.     }
303.  }
```