Modelleren van CMS based WebApplicaties

Master Thesis

Versie: 1.2

Scriptienummer: 594

23 juli 2008

M.J.B. (Thijs) Kupers

Radboud Universiteit Nijmegen

Begeleider: Patrick van Bommel

Master Computer Science

Colofon

Auteur

Thijs Kupers (0521450)
Radboud Universiteit Nijmegen
Master Computer Science, Information Systems
thijs@thijskupers.nl

Contactgegevens GX

GX Creative Online Development B.V.
Wijchenseweg 111
6538 SW Nijmegen
024-38 88 261
www.gx.nl
info@gx.nl

Supervisors

Jurriaan Souer (GX) – jurriaans@gx.nl
Patrick van Bommel (Radboud Universiteit) – p.vanbommel@cs.ru.nl

Abstract

Webtechnologie speelt een steeds belangrijke rol in de huidige informatiemaatschappij. Door het gebruik van Content Management Systemen (CMS) kunnen websites eenvoudig en snel worden aangepast. Deze systemen worden steeds geavanceerder en daardoor ook complexer. Dit paper beschrijft een techniek voor het modelleren van CMS-based WebApplicaties. Hierbij is de focus gelegd op het modelleren van formulieren. Aan de hand van een framework zijn verschillende bestaande modelleertalen met elkaar vergeleken. Op basis hiervan is het WebForm Diagram ontwikkeld. Tevens is er een koppeling gelegd tussen het diagram en een CMS, waardoor deze automatisch geconfigureerd kan worden.

Doordat het diagram verschillende abstractieniveaus ondersteunt, kan het worden toegepast in verschillende fasen van het ontwikkelproces. Daarnaast verbetert het de communicatie, wat ten goede komt aan de kwaliteit van de implementatie en de efficiëntie waarmee deze implementatie kan worden uitgevoerd. Het maakt een formulier inzichtelijker en biedt vanwege zijn koppeling met het CMS een goede documentatie mogelijkheid.

Inhoudsopgave

H1 Inleiding

Het internet krijgt een steeds grotere rol in het dagelijks leven. Meer en meer nieuwe websites ontstaan en meer en meer actuele gebeurtenissen worden op een interactieve manier aan elkaar gekoppeld (Lee en Shirani 2004). Met de groei van het web, zijn er ook applicaties ontstaan die het beheer van al deze informatie vereenvoudigen. WYSIWYG-editors (What You See Is What You Get) zorgen ervoor dat mensen zonder kennis van HTML een eigen website kunnen maken. Maar websites worden steeds veeleisender en willen meer en actuelere content bieden. Dit kwam met de intrede van het CMS: Een content management systeem, waarbij een webpagina wordt gescheiden van content en lay-out. Hierbij kan de content eenvoudig worden toegevoegd en worden gewijzigd, wat optimale flexibiliteit biedt (Lee en Shirani 2004).

1.1 Probleemstelling

In het verleden is er onderzoek gedaan naar ontwerp methodieken voor webapplicaties (Schwabe and Rossi 1998), (Ceri, Fraternali and Bongio 2000), (Brambilla, et al. 2006), (Koch, Software Engineering for Adaptive Hypermedia Systems: Reference Model, Modeling Techniques and Development Process 2000), (Koch and Kraus, The expressive Power of UML-based Web Engineering 2002), (Pastor, Fons and Pelechano 2003), (Melia and Gómez, Applying Transformations to Model Driven Development of Web Applications 2005) en (Meliá, Gómez and Koch 2005). Veel van deze methodes zijn afgeleid van al bestaande methodes en technieken. Echter, er is nog weinig onderzocht op het gebied van webengineering waarbij er gebruikt wordt gemaakt van een al bestaand Web-based Content Management System (WCMS) – Het meeste onderzoek gaat uit van web engineering van scratch (Weerd, et al. 2006).

GX WebEngineering Method (WEM) geeft een specificatie van het engineeringtraject van een CMS gebaseerde webapplicatie (Souer, et al. 2007). Hierbij wordt onderscheid gemaakt tussen een standaard traject en een complex traject. Standaard projecten zijn hoofdzakelijk gebaseerd op standaard functionaliteiten die in het CMS aanwezig zijn. Complexe projecten bevatten zowel nieuwe functionaliteiten als aanpassingen op de standaard functionaliteiten.

Het projectmatige gedeelte van WEM is grotendeels beschreven. Uit case studies blijkt dat de vertaalslag van functionaliteit naar standaard componenten en hun configuratie nog niet goed is uitgewerkt (Weerd, et al. 2006), (Souer, et al. 2007). De stap tussen requirements en Use Cases enerzijds en een geconfigureerd WCMS anderzijds is nog te groot. Dit geldt zowel voor de ontwikkelaars als voor de klant.

1.2 Onderzoeksvraag

Na aanleiding van bovengeschetste probleemstelling, is de volgende onderzoeksvraag gedefinieerd:

"Hoe zou een modelleertaal eruit moeten zien waarmee je op basis van requirements en use cases modules binnen een CMS kan specificeren?"

Door een taal te creëren waarin op een functioneel niveau de componenten kunnen worden gespecificeerd, wordt de vertaling tussen requirements en een werkend product makkelijker. Deze taal dient aan te sluiten bij de wensen van de gebruikers, zodat het de communicatie verbetert. Doordat de specificatie van de modelleertaal formeel vast ligt kunnen aan de hand van gemaakte modellen gedeelten van de WCMS gegenereerd worden. Daarnaast kan het proces ook de andere kant op werken: Door reverse engineering, kunnen gedeelten van het WCMS worden getransformeerd tot een model, welke het WCMS beschrijft. Hierdoor kan het model gebruikt worden als documentatie en verbetert het de communicatie.

Een WCMS kan bestaan uit erg veel onderdelen. Naast de functionaliteit om eenvoudig content op een website te plaatsen, komen er ook complexe zaken bij kijken zoals personalisatie, formulieren, verschillende soorten content, autorisatie, navigatie, uitbreidbaarheid middels plug-ins, multimedia enzovoorts. De scoop van dit onderzoek beperkt zich tot de functionaliteiten van webformulieren. Deze formulieren zijn de belangrijkste interactie met de bezoeker. Door het onderzoek hierop af te baken kan er een goed beeld worden gevormd van de mogelijkheden van het modelleren van dit deel van een WCMS. Als blijkt dat het modelleren van dit deel van de applicatie in de praktijk ook een toegevoegde waarde heeft, kan er ook worden overgegaan tot het modelleren van andere zaken van de applicatie.

Om het onderzoek beheersbaar te houden, is de onderzoeksvraag opgesplitst in verscheidene deelvragen:

- Welke eigenschappen moet de modelleertaal hebben?
    - Wie zijn de gebruikers?
        - Wat zijn hun doelen?
    - Welke informatie en functionaliteiten van een CMS moet er gemodelleerd kunnen worden?
- Hoe is een modelleertaal gedefinieerd?
    - Hoe vindt de transformatie van het model naar de achterliggende code plaats?
- Welke bestaande modelleertalen worden in hetzelfde vakgebied ingezet?
    - Welke aspecten hiervan zijn bruikbaar?
    - Hoe kunnen bestaande modelleertalen met elkaar worden vergeleken?
        - Hoe kunnen wensen van gebruikers in deze vergelijking worden meegenomen?

De volgende sectie beschrijft de relatie tussen een informatie systeem en een CMS-based WebApplicatie. De rest van deze thesis is als volgt opgebouwd. Hoofdstuk 2 beschrijft Model Driven Engineering, een vorm van softwareontwikkeling, waarbij modellen centraal staan. Vervolgens wordt in hoofdstuk 3 het probleemdomein, dat van CMS-based WebApplicaties, beschreven. De gebruikersanalyse staat in hoofdstuk 4, waarna bestaande modelleertalen met elkaar worden vergeleken in hoofdstuk 5. In hoofdstuk 6 wordt vervolgens de definitie gegeven van de modelleertaal: het WebForm Diagram. Vervolgens wordt in hoofdstuk 7 de implementatie hiervan beschreven. Deze wordt in hoofdstuk 8 geëvalueerd. Tot slot wordt in hoofdstuk 9 de conclusie van dit onderzoek getrokken. In hoofdstuk 10 zijn de bronnen vermeld, welke gebruikt zijn in dit onderzoek.

1.3 CMS-based WebApplicaties

Door de explosieve groei van het web, groeit ook de hoeveelheid beschikbare informatie (Fielden 2002). Het wordt daarom steeds belangrijker om deze informatie eenvoudig te kunnen beheren. Dit leidde tot steeds geavanceerdere informatie systemen, met elk hun eigen doel. Door de komst van snellere internetverbindingen, nieuwere webtechnologieën en de groeiende vraag naar eenvoudig te beheren websites, zijn Web Content Management Systemen (WCMS) ontstaan. In Figuur 1 wordt de relatie tussen informatie systemen en WCMS'en geschetst (Souer, et al. 2007). De verschillende typen informatie systemen worden hieronder verder beschreven

Information System (Informatiesysteem)

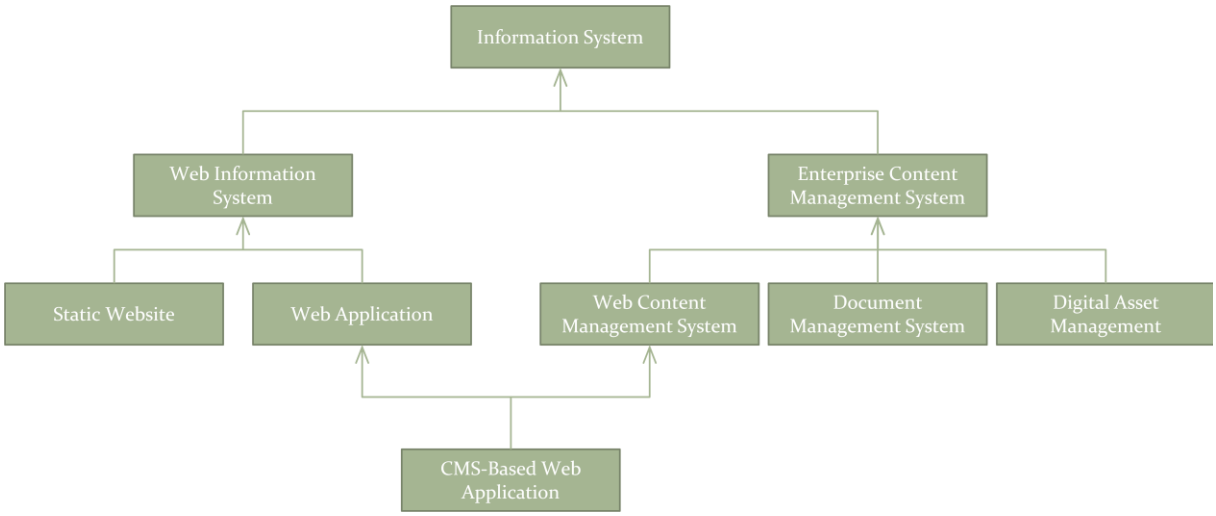
Een informatiesysteem kan op drie verschillende manieren gezien worden. Ten eerste als een technisch systeem geïmplementeerd door computers en communicatieapparatuur. Ten tweede als een sociaal systeem, zoals een organisatie, in relatie tot de aanwezige informatie, en ten derde als een conceptueel systeem, wat een abstractie is van één van bovengenoemde (Hirschheim, Klein and Lyytinen 1995).

Web Information System (Webinformatiesysteem)

Een webinformatiesysteem is een speciaal type informatiesysteem welke gebruik maakt van webtechnologie. Dit gebruik kan breed worden opgevat, dit kan slechts zijn voor de presentatie van informatie, maar ook interactieve content en beheer (Souer, et al. 2007).

Static Website (Statische website)

Een statische website is een informatie systeem, toegankelijk via het web, waarbij de gebruiker kan navigeren tussen de verschillende informatie. Dit wordt door De Troyer ook wel Kiosk website genoemd. Genoemd naar de digitale informatiekiosken in musea, waar je door verschillende informatie kan navigeren (Troyer en Leune 1998).



Figuur 1 De positionering van CMS-based WebApplicaties ten aanzien van Informatie Systemen (Souer, et al. 2007)

Web Application (Webapplicatie)

Een webapplicatie is een informatiesysteem die toegang biedt tot informatie en interactieve functionaliteit via het web, waarbij ook de status van deze data gewijzigd kan worden (Gnaho 2001).

Enterprise Content Management System (Enterprise Content Management Systeem)

Een Content Management Systeem (CMS) biedt de mogelijkheid voor het creëren, archiveren, zoeken, beheren en publiceren van informatie in een flexibele en geïntegreerde omgeving (Burzagli, et al. 2004).

Digital Asset Management

Digital Asset Management (DAM) is het beheer van multimediale bestanden zoals, beeld, geluid, animaties, muziek. Onder beheren valt onder andere het delen, back-uppen, ranken, groeperen, archiveren, onderhouden en exporteren (Austerberry 2004).

Document Management System (Document Management Systeem)

Een document management systeem (DMS) beheert en bewaard documenten. Documenten kunnen op deze manier door de gehele organisatie gevonden en gebruikt worden. Deze documenten hebben eigenschappen in plaats van een locatie, zoals daar in een bestandsysteem sprake van is. Denk hierbij aan versiebeheer, verschillende statussen (concept, definitief) en archivering, auteursinformatie, enzovoorts (Dourish, et al. 2000).

Web Content Management System (Web Content Management Systeem)

Web Content Management omvat de activiteiten bij de creatie van digitale content richting een webpubliek. Een Web Content Management Systeem bestaat uit de software tools die benodigd zijn om Web Content Management uit te voeren (McKeever 2003).

CMS-Based Web Application (CMS gebaseerde Webapplicatie)

Een CMS-based WebApplicatie kan gezien worden als een webapplicatie voor het beheren en controleren van content. Dit omvat een scheiding van content, structuur en grafische design (Souer, et al. 2007).

H2 Model Driven Engineering

Het doel van dit onderzoek is om een modelleertaal te ontwerpen voor het modelleren van een CMS. Hierbij wordt een model geschetst van de gewenste situatie en hieruit wordt het CMS automatische geconfigureerd. Deze werkwijze komt overeen met Model Driven Engineering (MDE) (Schmidt 2006).

MDE is het systematisch gebruiken van modellen als primaire ontwerpmethode in het ontwerpproces. De modellen worden gebruikt bij het implementatie proces. Sommige modellen bevatten weinig detail waarbij bijbehorende code met de hand moet worden geschreven. Andere modellen bevatten alle details en kunnen worden gebruikt om code uit te genereren.

2.1 Historisch perspectief

De afgelopen tientallen jaren zijn er verschillende abstractie lagen gecreëerd zodat software ontwikkeld kan worden met aandacht voor het ontwerp, in plaats van het onderliggende systeem, zoals CPU, geheugen en netwerk. Het programmeren met machinecode was niet meer nodig met het ontstaan van programmeertalen als assembly en Fortran. Ook het ontstaan van besturingssystemen zorgde ervoor dat er niet direct met de hardware gecommuniceerd hoefde te worden. Dit bracht het ontwikkelen van software naar een hoger abstractieniveau. Echter, deze oplossingen zijn nog steeds computer georiënteerd. Ze abstraheren slechts in de *solution space* in plaats van de *problem space*. De *problem space* is het domein waarbinnen de applicatie moet functioneren, zoals Telecom, luchtvaart, verzekeringen, etc.

In de jaren tachtig ontstond computer-aided software engineering (CASE), waarbij de focus lag op het ontwerpen door middel van grafische representaties. Een probleem was wel dat deze CASE-tools slecht aansloten op de onderliggende platforms en complexe code genereerde. Hierdoor was het moeilijk om te ontwerpen, debuggen en ontwikkelen met CASE-tools en de daarmee gemaakte applicaties.

De verschillende talen en platformen kwamen ook op een steeds hoger abstractie niveau te staan. Hogere programmeertalen als C++, Java en C# doen hun intrede. Ontwikkelplatformen zoals J2EE, .NET en CORBA bevatten duizenden klassen en methoden, met vele afhankelijkheden, wat veel mogelijkheden biedt, maar ook de complexiteit vergroot. Al deze complexe zaken, zorgen ervoor dat ontwikkelaars veel aandacht moeten besteden aan details van het programmeerwerk, in plaats van zich te concentreren op abstractere architectuur.

Een mogelijke oplossing om deze complexiteit, en het gebrek om concepten van het domein binnen een dergelijke programmeertaal goed uit te kunnen drukken is Model Driven Engineering. Hierbij wordt een domein specifieke modelleertaal gebruikt als basis voor de onderliggende code (Schmidt 2006).

2.2 Model Driven Engineering

Binnen Model Driven Engineering (MDE) worden modellen gebruikt als primaire ontwikkelmethode. MDE is een combinatie van een Domain Specific Modeling Language en een transformatie mechanisme. De domein specifieke modelleertaal, geeft de structuur, gedrag of requirements van een bepaald domein weer. Het transformatiemechanisme transformeert het model naar de verschillende implementaties, zoals broncode, xml-bestanden, invoer van een simulator of een andere weergave van het model (Deursen, Klint and Visser 2000).

2.2.1 Domain Specific Modeling Language

Een domain specific modeling language (DSML) is een grafische executeerbare specificatie taal dat, door middel van verschillende notaties en abstracties, gespecialiseerd is om concepten uit een bepaald domein te modelleren. Een DSML wordt beschreven door een metamodel, welke de relaties tussen concepten uit het domein beschrijft. Ook de semantiek en de verschillende beperkingsregels uit het domein worden hierdoor beschreven.

Een voordeel van een domein specifieke taal is dat een oplossing biedt beschreven met concepten uit het probleemgebied. Hierdoor is het model ook leesbaar voor domeinexperts, zodat hun kennis beter kan worden ingezet bij de validatie en het wijzigen van het model. Daarnaast beschrijven de modellen de kern van het probleem, kan het gebruikt worden als documentatiemiddel en kan eenvoudig worden hergebruikt. Dit alles verbetert de productiviteit, betrouwbaarheid, onderhoudbaarheid en de communicatie.

Doordat de taal specifiek is voor een bepaald domein, heeft het een kleine leercurve en biedt het ondersteuning aan een grote groep gebruikers. Bovendien biedt het validatie en optimalisatie mogelijkheden in een vroeg stadium van een project. Hoe eerder in een project een fout wordt ontdekt, hoe goedkoper het is om deze fout te herstellen.

Een nadeel van een specifiek domeingerichte modelleertaal is het feit dat deze speciaal voor het domein moet worden ontwikkeld, en dat hier natuurlijk ook allerlei kosten aan verbonden zijn. Hierdoor zijn domeinspecifieke talen vaak maar beperkt beschikbaar. Daarnaast is het ook moeilijk om een onderscheid te maken tussen domein specifieke en algemene programmeertaal constructen. Tevens kan gegenereerde code minder efficiënt zijn dan handgeschreven code (Deursen, Klint en Visser 2000).

2.2.2 Transformatie Mechanisme

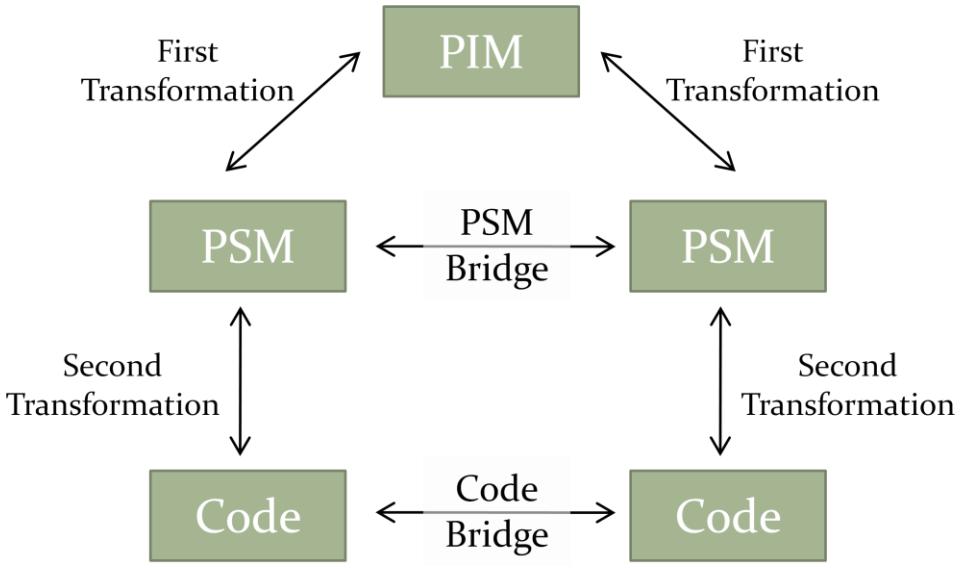
Een transformatie mechanisme transformeert een model naar broncode, een xml-bestand, of eventueel een ander model. Door deze directe koppeling is de kans op fouten aan de implementatie kant een stuk kleiner. Een vorm van MDE is Model Driven Architecture (MDA). Dit is een door de Object Management Group (OMG) vastgelegde standaard (OMG, MDA Guide Version 1.0.1 2003).

Het belangrijkste punt binnen MDA zijn de modellen binnen het software ontwikkelproces. Het ontwikkelproces bestaat uit het modelleren van het uiteindelijke systeem. Net als bij traditionele software ontwikkelmethoden, bestaat het MDA ontwikkelmodel uit verschillende fases. Nadat de

requirements en de analyse is uitgevoerd, wordt er een platform onafhankelijk model (Platform Independent Model, PIM) gemaakt. Dit model beschrijft het domein onafhankelijk van het implementatieplatform. Vervolgens transformeert dit tot een platform specifiek model (Platform Specific Model, PSM). Vervolgens staat de PSM aan de basis voor de uiteindelijke code. Uit een PIM kunnen voor verschillende platformen verschillende PSM's worden gegenereerd, zie ook Figuur 2.

Een voorbeeld van dit proces is een specificatie van een Entity-Relationship diagram. Hier wordt een bepaald domein beschreven in termen van entiteiten, attributen en relaties. Dit kan gezien worden als een PIM, want het is onafhankelijk van de implementatie beschreven. Vervolgens is dit model te transformeren naar een PSM, een database model, met tabellen, kolommen en verwijzende sleutels specifiek voor bijvoorbeeld Oracle. Hieruit is vervolgens SQL code te genereren die de database kan maken. De PIM had ook getransformeerd kunnen worden naar een PSM welke de verschillende klassen binnen Java beschrijft. Ook hier kan code uit gegenereerd worden.

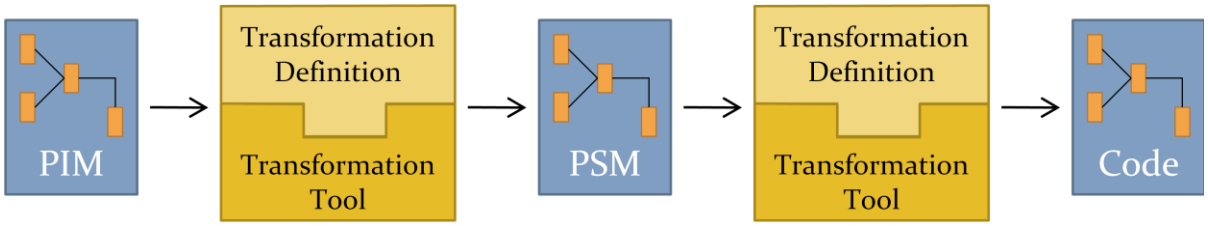
Het transformatie mechanisme bestaat uit twee zaken: een transformatie tool en een transformatie definitie, zie ook Figuur 3. De transformatie tool verzorgt de transformatie van een PIM naar een PSM. Deze tool gebruikt een definitie welke beschrijft hoe het model getransformeerd moet worden, de transformatie definitie. Een voorbeeld hiervan is de transformatie van UML naar C#. Deze definitie bevat verschillende regels, bijvoorbeeld hoe een Klasse in UML moet worden omgezet naar C# code, en hoe een relatie tussen x en y in UML transformeert naar een variabele definitie in klasse X van het type Y, in C#. De transformatie tool gebruikt deze regels om de transformatie uit te voeren. Hieruit volgt ook de definitie van een transformatie.



Figuur 2 Verschillende transformaties binnen MDA. Door middel van bridges is het ook mogelijk om tussen verschillende PSM's te transformeren (Kleppe, Warmer and Bast 2003).

Een transformatie is het automatisch genereren van een doelmodel vanuit een bronmodel, volgens een transformatie definitie. Een transformatie definitie is een verzameling regels die gezamenlijk beschrijven hoe een model in de brontaal kan worden omgezet naar een model in de doeltaal. Een transformatie regel is een beschrijving hoe een constructie in de brontaal kan worden vertaald naar een constructie in de doeltaal.

De bron- en doelmodel kunnen zowel in verschillende talen als dezelfde taal gespecificeerd zijn. Zo kunnen er ook transformaties zijn van UML naar UML of van Java naar Java. Dit kan bijvoorbeeld gebeuren wanneer er refactoring binnen het model plaatsvindt (Kleppe, Warmer and Bast 2003).



Figuur 3 De transformatie vindt plaats middels een transformatie tool en een transformatie definitie. Door deze de scheiden is hergebruik makkelijker (Kleppe, Warmer and Bast 2003).

2.3 Conclusie

De werkwijze van Model Driven Engineering komt overeen met de doelstelling van dit onderzoek: Door middel van een model wordt een module binnen het WCMS geconfigureerd. Uit onderzoek met betrekking tot MDE blijkt dat dit vele voordelen biedt. De genoemde nadelen worden door dit onderzoek ook beperkt: Zo richt dit onderzoek zich op de ontwikkeling van een domeingerichte modelleertaal, waardoor het nadeel van het gebrek aan een domein specifieke taal wegvalt.

H3 Domeinmodel

Dit hoofdstuk beschrijft het domein van een Webbased Content Management Systeem. Dit geeft antwoord op de deelvraag welke informatie er gemodelleerd moet worden. De verschillende eigenschappen en hun onderlinge verbanden uit het probleemgebied worden hierbij in kaart gebracht. Dit gebeurt middels een domeinmodel. Het domeinmodel beschrijft de elementen van het domein en hun onderlinge relaties. Hierbij zitten ook generalisaties bij, die verschillende entiteiten op een hoger niveau met elkaar verbinden (Kang, et al. 1990).

3.1 Beschrijving van het domein

Het domeinmodel is beschreven door middel van zijn entiteiten. Het domeinmodel is opgesplitst in 3 afbeeldingen en is weergegeven in Figuur 4, Figuur 5 en Figuur 6. Hieronder staan de beschrijvingen van de verschillende entiteiten uit het domeinmodel.

Form (formulier)

Synoniemen: webform, e-form

Omschrijving: Een formulier is een middel om gestructureerd informatie van de bezoeker te vergaren. Dit gebeurt meestal doordat de bezoeker antwoord moet geven op open of meerkeuze vragen (Watters and Zhang 2003).

Attributen: Name (String): De naam van het formulier.

Step (stap)

Synoniemen: -

Omschrijving: Een stap is een afzonderlijk deel van het invullen van het formulier. Een formulier bestaat uit één of meer stappen.

Attributen: Name (String): De naam van de stap.

Subform (subformulier)

Synoniemen: -

Omschrijving: Een subformulier dient als template voor een (gedeelte van) een formulier. Hiermee zijn veel gebruikte formulierdelen te hergebruiken.

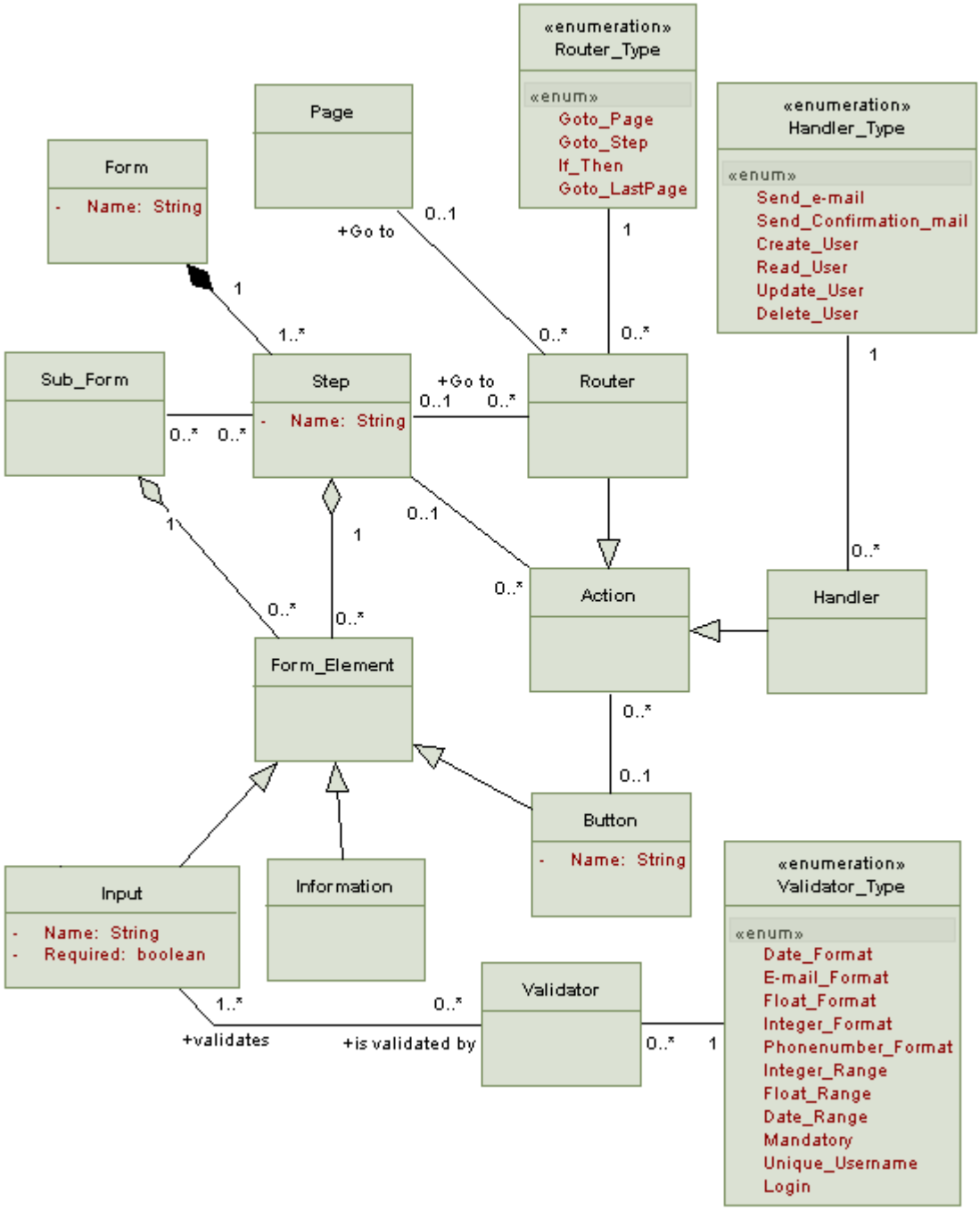
Attributen: Name (String): De naam van het subformulier

Form Element (formulier element)

Synoniemen: onderdeel

Omschrijving: Een formulier element is een afzonderlijk element van een stap/subformulier. Dit is onder te verdelen in drie categorieën: invoervelden, knoppen en informatie (Watters and Zhang 2003).

Attributen: -



Figuur 4 Het Domeinmodel: Dit deel geeft de samenhang tussen een formulier, stappen, velden, validators, handlers en routers weer.

Input (invoer)

Synoniemen: invoerveld

Omschrijving: Invoer geeft de bezoeker de mogelijkheid om informatie in te voeren. Er zijn drie verschillende typen: tekstinvoer; bestandsupload en een (keuze-) lijst (Watters and Zhang 2003).

Attributen: Name (String): De naam van het invoerveld. Deze kan dienen als label van het invoerveld.
Required (Boolean): Een indicator om aan te geven dat dit veld verplicht moet worden ingevuld.

Text Input Field (tekstinvoer)

Synoniemen: tekstveld

Omschrijving: De tekstinvoer geeft de bezoeker de mogelijkheid om tekst in te voeren.

Attributen: Default Value (String): Een vooraf ingevulde waarde.

Upload File (bestandsupload)

Synoniemen: -

Omschrijving: De bestandsupload geeft de bezoeker de mogelijkheid om een bestand mee te sturen met het formulier.

Attributen: -

List (lijst)

Synoniemen: keuzelijst

Omschrijving: Een lijst geeft de bezoeker de mogelijkheid om één of meerdere items te kiezen uit een set van items.

Attributen: -

Predefinied List (voorgedefinieerde lijst)

Synoniemen: -

Omschrijving: Een voorgedefinieerde lijst dient als template voor een lijst. Hiermee zijn lijsten die vaak voorkomen, eenvoudig te hergebruiken. Bijvoorbeeld:

- Man, Vrouw (geslacht)
- Zeer mee eens, mee eens, neutraal, mee oneens, zeer mee oneens (enquête)

Attributen: -

Query List (query lijst)

Synoniemen: -

Omschrijving: Een voorgedefinieerde lijst waarbij de items aan de hand van een databasequery tijdens executie uit de database worden gehaald.

Attributen: -

Button (knop)

Synoniemen: -

Omschrijving: Met een knop kan de bezoeker een actie uitvoeren op het formulier.

Attributen: Name (String): De naam van de knop. Deze kan dienen als label van de knop.

Information (informatie)

Synoniemen: informatieblok; informatie-element

Omschrijving: Een informatie element kan worden gebruikt om informatie te tonen aan de bezoeker. Bijvoorbeeld een tekst of een afbeelding (Watters and Zhang 2003).

Attributen: -

Text (tekst)

Synoniemen: -

Omschrijving: De tekst wordt gebruikt om een stuk tekst te tonen aan de bezoeker. Dit geeft dus geen invoermogelijkheden aan de bezoeker (Watters and Zhang 2003).

Attributen: Name (String): De te tonen tekst.

Image (afbeelding)

Synoniemen: plaatje

Omschrijving: Een afbeelding wordt gebruikt om een afbeelding te tonen aan de bezoeker. Dit geeft dus geen invoermogelijkheden aan de bezoeker (Watters and Zhang 2003).

Attributen: -

Action (actie)

Synoniemen: handeling

Omschrijving: Een actie voert een handeling uit op het formulier. Zo moeten er bijvoorbeeld acties worden uitgevoerd op het moment dat er op een knop wordt gedrukt, of nadat een stap is afgerond (Cooper, Reimann and Cronin 2007).

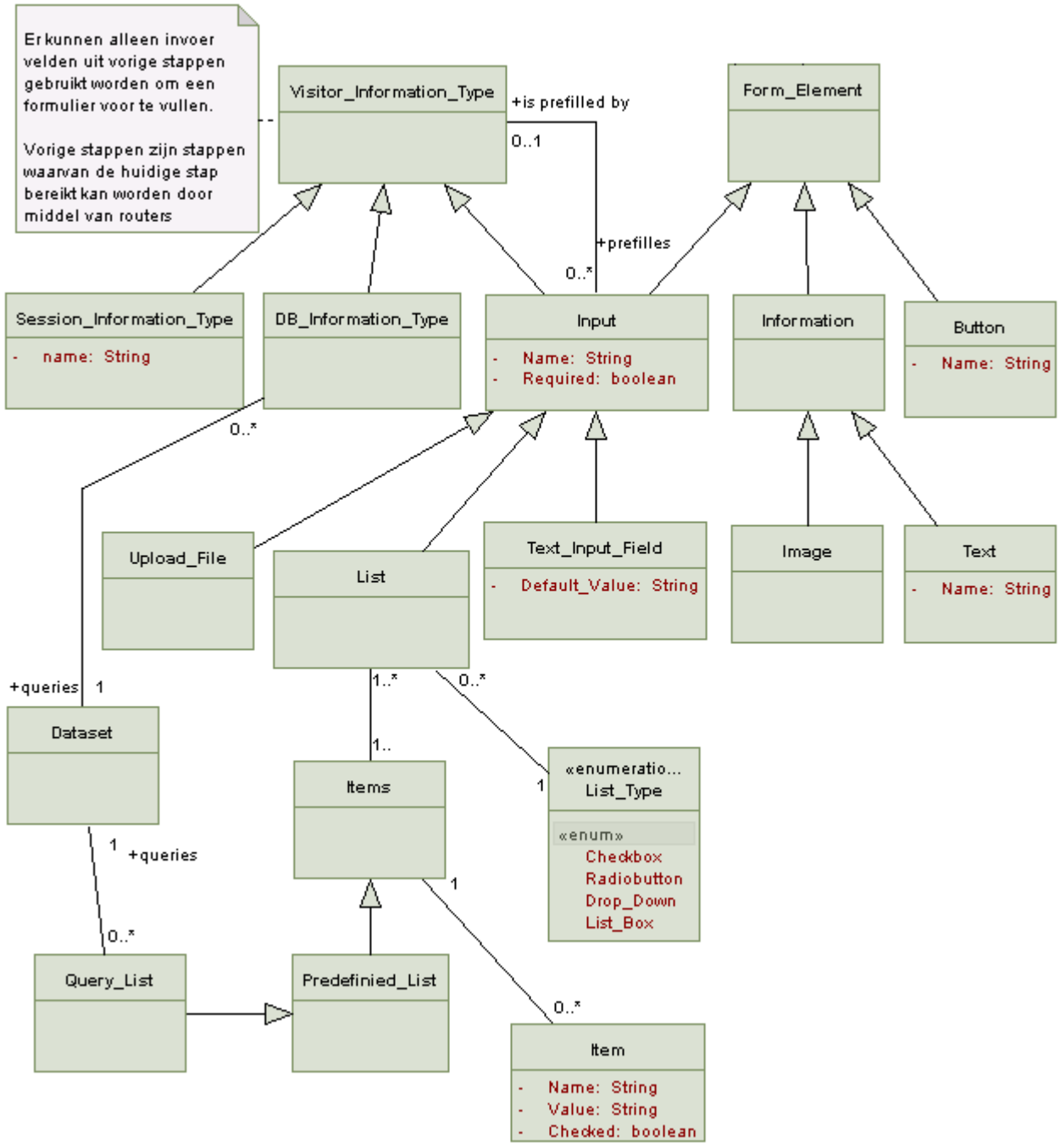
Attributen: -

Validator (validatieregel)

Synoniemen: -

Omschrijving: Een validatieregel controleert of de gebruikersinvoer voldoet aan bepaalde criteria. Dit kan zowel op een enkel inputveld zijn (bijvoorbeeld het e-mail adres voldoet aan de vorm xxx@domein.nl), alsmede een combinatie van velden (bijvoorbeeld controleer gebruikersnaam/wachtwoord) (Brabrand, et al. 2000).

Attributen: -



Figuur 5 Het Domeinmodel: Dit deel geeft de verschillende formulierelementen weer.

Handler

Synoniemen: -

Omschrijving: Een handler voert een bepaalde actie uit. Dit kan na elke stap plaatsvinden. Hierbij kan de handler gebruik maken van de gegevens die zijn ingevuld. Zo kunnen er gegevens uit de database worden gelezen, gegevens worden weggeschreven, een e-mail worden gestuurd, etc.

Attributen: -

Router

Synoniemen: -

Omschrijving: Een router bepaalt waar de bezoeker naar toe geleid wordt na het invullen van een formulier(stap). Dit kan een vervolg stap zijn of een hele andere pagina.

Attributen: -

Personalisatie

Personalisatie is het op maat aanbieden van content aan de bezoeker. Dit op basis van een gebruikersprofiel welke zowel door de bezoeker (bijvoorbeeld door middel van inloggen en het aangeven van voorkeuren) als door het systeem (onder andere door het afleiden van interesses uit bezochte webpagina's) gecreëerd kan worden (Mobasher 2007). Binnen een formulier kan dit gebruikersprofiel op twee verschillende manieren worden gebruikt.

Persoonlijke content: Het toespitsen van content richting de bezoeker aan de hand van zijn interesses en eventueel opgelegde beperkingen. Zo zou je bijvoorbeeld afgeschermde gedeeltes van een website kunnen hebben, die alleen toegankelijk zijn nadat de bezoeker is ingelogd. Ook kun je content wel of niet aanbieden afhankelijk van het gebruikersprofiel (Rossi, Schwabe and Guimarães 2001). Bij het boeken van een vliegticket zou een gepersonaliseerde vraag gesteld kunnen worden over de maaltijd tijdens de vlucht. Als er in een gebruikersprofiel staat dat de bezoeker vegetariër is, zou een vraag over een menu met kip of met vis niet relevant zijn. Dit concept wordt in het domeinmodel gerealiseerd door middel van de condition (conditie) entiteit.

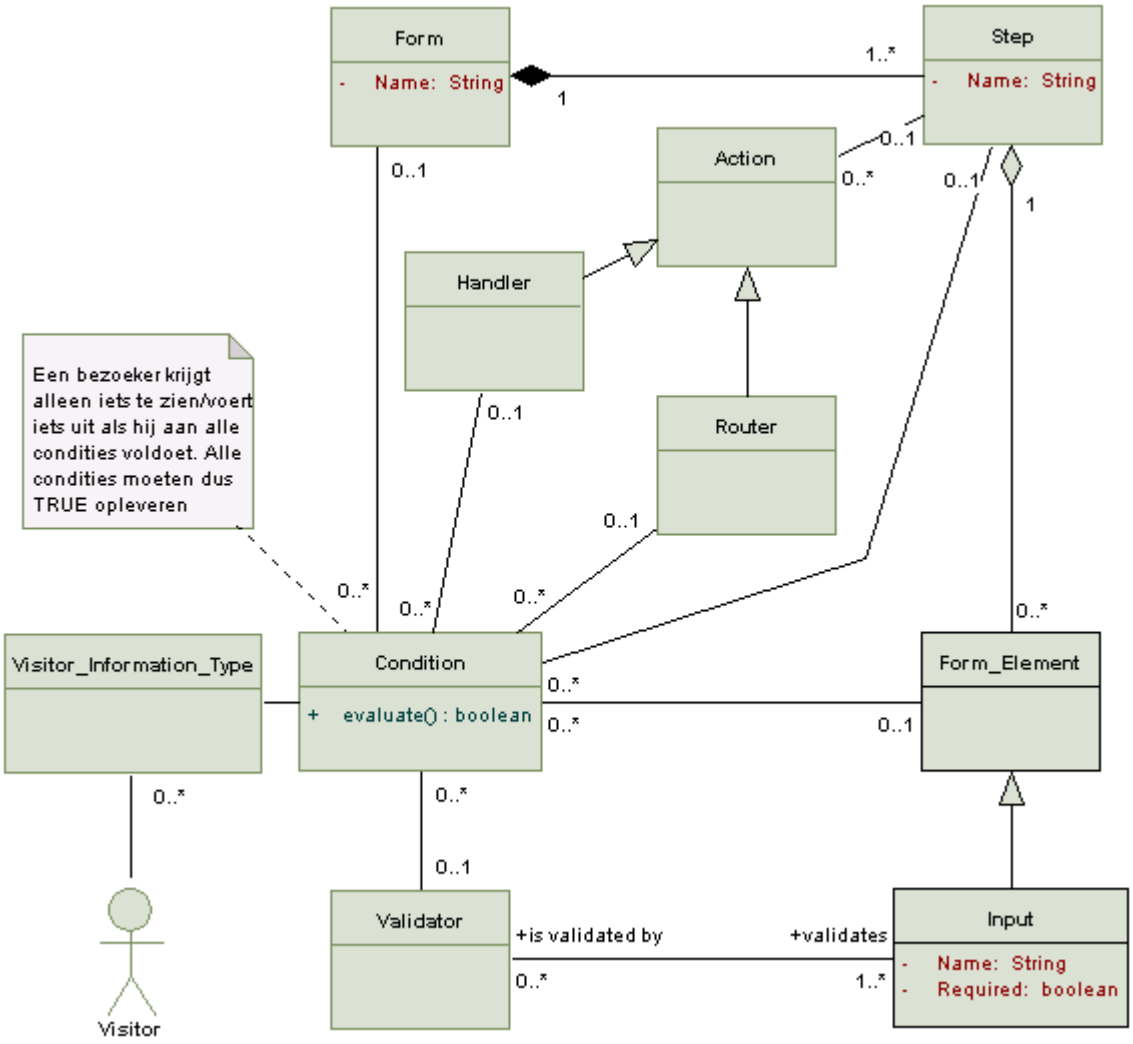
Invoervelden vooraf invullen: Het vooraf invullen van invoervelden met beschikbare informatie van de bezoeker. Dit vereenvoudigt het invulproces voor de bezoeker. Zo kan bijvoorbeeld zijn adres en woonplaats gegevens al zijn ingevuld. Deze informatie kan zowel uit de database, de actieve sessie of uit eerder ingevulde stappen gehaald worden. Dit concept wordt in het domeinmodel gerealiseerd door middel van de Visitor Information Type (bezoekersinformatie-type) entiteit.

Condition (conditie)

Synoniem: voorwaarde, preconditie

Omschrijving: Een beperkingsregel gebaseerd op informatie van de bezoeker. Bepaalde zaken mogen wel of niet worden gezien, afhankelijk van de instellingen en rechten van de bezoeker. Dit is een vorm van personalisatie.

Methoden: boolean evaluate(): Deze methode kijkt of de bezoeker aan de voorwaarde voldoet.



Figuur 6 Het Domeinmodel: Dit deel geeft de samenhang van een conditie weer.

Visitor Information Type (bezoekersinformatie-type)

Synoniemen: -

Omschrijving: De bezoekersinformatie bevat een type van informatie van de bezoeker. Bijvoorbeeld de gebruikersnaam in de sessie. Deze informatie kan als vooraf ingevoerde waarde voor een invoerveld dienen (personalisatie). De informatie kan uit de database komen (entiteit Database Information Type (database informatie-type)), uit de sessie (entiteit Session Information Type (sessie informatie-type)) of een waarde van een eerder ingevuld invoerveld.

Attributen: -

Database Information Type (database informatie-type)

Synoniemen: -

Omschrijving: De database informatie bevat een verwijzing naar een veld in de database. Deze kan gebruikt worden voor het vooraf invullen van een invoerveld. Zie ook *Visitor Information Type (bezoekersinformatie-type)*.

Attributen: -

Session Information Type (sessie informatie-type)

Synoniemen: -

Omschrijving: De Sessie informatie bevat een verwijzing naar gebruikersinformatie die is opgeslagen in de sessie. Deze kan gebruikt worden voor het vooraf invullen van een invoerveld. Zie ook *Visitor Information Type (bezoekersinformatie-type)*.

Attributen: -

3.2 Voorbeeld

Om de relatie van de geschetste begrippen wat te verduidelijken volgt hier een voorbeeld van een registratieproces op een website. Hierin komen verschillende elementen van een formulier voor.

Stap 1: De bezoeker vult zijn gegevens in. Deze worden gecontroleerd, bijvoorbeeld of het een unieke gebruikersnaam is (validatieregel). De gegevens worden in de database opgeslagen (handler) en de bezoeker krijgt een activeringsmail (handler). Vervolgens krijgt de bezoeker de volgende pagina te zien (router).

Stap 2: Deze stap bevat eigenlijk geen input van de bezoeker, maar slechts een tekst bericht dat er een activeringsmail is gestuurd naar de bezoeker.

Vervolgens klikt de bezoeker in de link in de activeringsmail en komt hij bij stap 3 terecht. Deze stap wordt dus niet bereikt door een router.

Stap 3: Ook deze stap bevat geen input van de bezoeker. Het toont een bericht dat de bezoeker succesvol is geregistreerd. Daarnaast wordt het account van de bezoeker in de database op *geactiveerd* gezet (handler). Hij krijgt vervolgens een inlogscherm waarin zijn gebruikersnaam (sessie informatie-type), Erik-Jan (de waarde van het sessie-informatie type: *gebruikersnaam*), al is ingevuld (personalisatie).

Papier versus webformulieren

Het zojuist beschreven registratieproces is een voorbeeld van een formulier op het web. Toch toont een webformulier veel gelijkenis met de papieren variant. Het grote verschil is dat de afhandeling van webformulieren wat afwijkt, doordat het gebruik kan maken van webfunctionaliteiten. Op papier wordt een handtekening gebruikt ter identificatie, op het web kan er bijvoorbeeld gebruik worden gemaakt van DigiD, e-mailverificatie (een bevestigingsmail) of een captcha (***c**ompletely **a**utomated **p**ublic **T**uringtest to tell **c**omputers and **h**umans **a**part*, een methode om te bepalen of er sprake is van een menselijke gebruiker (Ahn, Blum and Langford 2004)).

Het invullen van een formulier (zowel digitaal als op papier) is meer een proces te noemen dan een enkele handeling (Thompson and Torabi 2007). Bij complexere formulieren moeten vaak meerdere stappen worden doorlopen, al dan niet sequentieel (*Indien 'ja' ga naar stap 4*). Soms moet er informatie worden opgezocht of worden uitgerekend. Sommige informatie is van tevoren al ingevuld of voorgedrukt (bijvoorbeeld naam en rekeningnummer bij een acceptgiro).

De combinatie van stappen, handlers en routers verzorgt het proces van invullen van het formulier. Het doorlopen van deze stappen in dit proces heet de flow.

3.3 Conclusie

Dit hoofdstuk heeft inzicht gegeven op de vraag welke informatie en functionaliteiten er van een CMS gemodelleerd moeten worden. Aan de hand van een domeinmodel zijn deze functionaliteiten geïdentificeerd. Het domeinmodel en de beschrijvingen van de entiteiten geven weer welke zaken er in de modelleertaal terug moeten komen.

H4 Gebruikersanalyse

De gebruikersanalyse heeft verschillende doelen. De gebruikers moeten uiteindelijk met de modelleertaal gaan werken, dus het is van belang dat het aansluit bij de eisen en wensen van de gebruikers. Wat de gebruiker met de taal wil doen, zijn persoonlijke doel, komt voort uit het gebruikersonderzoek. Daarnaast is het van belang dat de taal overeenkomt met de denkwijze van de gebruiker. Deze twee zaken worden eerst globaal toegelicht in dit hoofdstuk, waarna een verslaglegging van de werkelijke analyse plaatsvindt. Het betreft hierbij gebruikers van GX WebManager. GX WebManager is een voorbeeld van een CMS-Based WebApplication.

4.1 GX WebManager

GX Webmanager (De huidige versie is versie 9) is een CMS-based webapplicatie, waarin mensen met een niet-technische achtergrond eenvoudig de content van de website kunnen beheren. Denk hierbij aan dagelijks vele nieuwe artikelen, foto's en filmpjes. Niet alleen het beheren van content, maar ook het beheren van andere zaken, zoals publicatie- en vervaldatum van een artikel, verschillende statussen van artikelen (bijvoorbeeld concept of gepubliceerd), workflow-, autorisatie- en interactiemanagement, FAQ, personalisatie, formulieren, multimediale content, caching en rapportage mogelijkheden.

Daarnaast kan er functionaliteit aan WebManager worden toegevoegd middels een WCB. Een WCB (WebManager Component Bundle) is een extern component welke gebruik maakt van de WebManager API en SDK. Doordat gebruikers hun eigen WCB's kunnen ontwikkelen, zijn ze niet afhankelijk van GX voor bedrijfsspecifieke functionaliteiten.

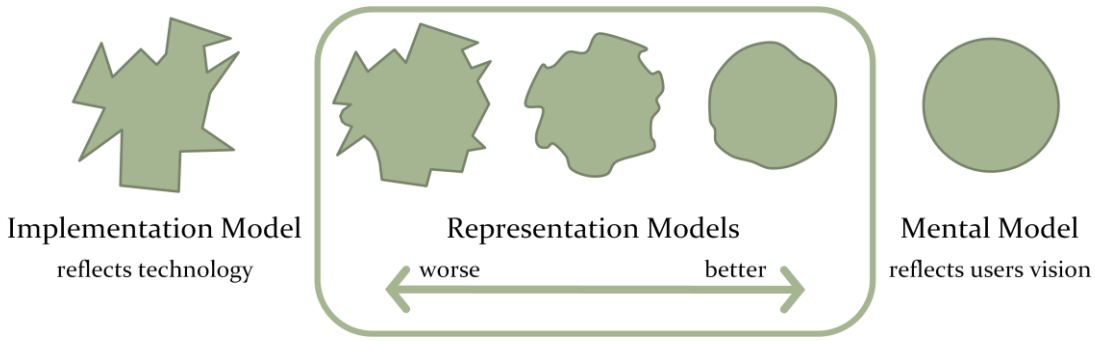
GX WebManager is in dit project gebruikt als CMS systeem. De abstracte syntax komt grotendeels overeen met de implementatie van de formulierenmodule van WebManager. Binnen WebManager is er een content import/export aanwezig, waarmee het mogelijk is om een formulier te exporteren of importeren met xml.

4.2 Mental Model

Het is belangrijk dat een systeem aansluit bij de denkwijze van de gebruiker, zijn mental model. De implementatie van het systeem staat hier vaak ver van af, maar door het kiezen van een goede representatie sluit het toch aan bij de werkwijze van de gebruiker. We beschrijven hieronder een drietal modellen: het Implementation Model, het Mental Model en het Representation Model.

Implementation Model

Een Implementation Model is de interne samenhang van componenten, relaties en hoe ze elkaar beïnvloeden binnen een systeem (Cooper, Reimann and Cronin 2007). Bijvoorbeeld mobiele communicatie. Er zijn verschillende zendmasten en een mobiele telefoon zoekt verbinding met de dichtstbijzijnde zendmast, door middel van een soort van radio signaal. Op het moment dat iemand in de buurt is van een andere zendmast, neemt die de verbinding over waardoor er gewoon kan worden gebeld.



Figuur 7 Het is belangrijk dat het Representation Model zo dicht mogelijk bij het Mental Model van de gebruiker ligt (Cooper, Reimann and Cronin 2007).

Bij een modelleertaal is het metamodel, de syntax en semantiek het Implementation Model.

Mental Model

Een Mental Model is de interne representatie van een systeem, zijn componenten, relaties en hoe ze elkaar beïnvloeden, zoals de gebruiker die heeft (Fein and Olson 1993), (Cooper, Reimann and Cronin 2007). Bijvoorbeeld bij het bellen met een mobiele telefoon: je kiest een nummer en je krijgt vervolgens verbinding met de persoon die je belt, net als met een gewone telefoon. Alleen, een mobiele telefoon werkt heel anders dan een vaste telefoon.

Hetzelfde kan ook gezegd worden voor bijvoorbeeld het maken van formulieren binnen een CMS. In je hoofd heb je een idee over de stappen, wat er op elke stap gebeurt, waar de validaties plaats vinden en hoe die stappen met elkaar verbonden zijn.

Representation Model

Het Representation Model is de representatie van het systeem die het heeft naar de buitenwereld. Deze representatie kan heel anders zijn dan de werkelijke implementatie (het implementation model). Bijvoorbeeld een besturingssysteem kan een netwerk fileserver er uit laten zien alsof het een lokale harde schijf is. Het model representeert niet de feitelijke implementatie van de schijf welke aan de andere kant van de wereld kan staan (Cooper, Reimann and Cronin 2007).

Bij een modelleertaal zijn de grafische symbolen die je gebruikt om de syntax en semantiek uit te drukken het Representation Model van de taal.

De relatie tussen de verschillende modellen is weergegeven in Figuur 7.

Hoe meer het Representation Model overeenkomt met het Mental Model van de gebruiker, hoe gemakkelijker de gebruiker het vindt om er mee te werken. Het systeem werkt intuïtiever omdat de gedachte die de gebruiker over het systeem heeft, zo kan hij met het systeem werken.

Bijvoorbeeld het specificeren van een kleur in een tekenprogramma. Dit zou geïmplementeerd kunnen zijn door het opgeven van 3 waardes (rood, groen en blauw) welke samen de kleur vormen. Dit staat echter dicht in de buurt van het Implementation Model. Beter is een interface

waarin je een kleur kan kiezen, en eventueel nog wat kan verfijnen (bijvoorbeeld iets donkerder, of iets feller). Dit staat dichter bij het Mental Model en is daarom makkelijker en intuïtiever voor de gebruiker.

Als het Implementation Model dicht tegen het Mental Model van de gebruiker aan staat, voorkom je onnodige complexiteit bij de gebruiker. Is dit niet het geval dan moet hij niet alleen nadenken over de complexe taak die hij moet uitvoeren, maar ook nog eens de vertaalslag naar het Implementation Model verrichten. Doordat deze vertaalslag niet hoeft te worden gemaakt kan de gebruiker zich volledig concentreren op zijn taak.

Door de representatie van de modelleertaal aan te laten sluiten bij het Mental Model van de gebruiker, zorg je er dus voor dat het intuïtief in gebruik is. Doordat gebruikers er gemakkelijk mee kunnen werken, vergroot dit de kans, dat het gebruikt gaat worden.

Door interviews te houden met de potentiële gebruikers wordt inzicht verkregen in hun mental model, zodat het te bouwen systeem hierop aansluit.

4.3 Analyse

De gebruikersanalyse is uitgevoerd door het houden van interviews onder een zestal gebruikers. Deze zijn allen werkzaam bij GX. Naast het feit dat ze zelf veelvuldig formulieren implementeren met WebManager, werken zij ook veel met klanten. Het doel van een CMS is dat de gebruiker, in dit geval de klant, zelf zijn content van zijn website kan beheren. Dit geldt dus ook voor formulieren. De ervaringen die de geïnterviewden hebben met klanten op dit gebied, komen ook naar voren in de interviews. Het doel van de interviews was om de volgende informatie te achterhalen:

- Wat zijn de huidige problemen bij het specificeren en implementeren van formulieren binnen GX WebManager. Het oplossen van deze problemen is voor een groot gedeelte het doel van de te ontwikkelen modelleertaal.
- Inzicht krijgen in het Mental Model van de gebruiker welke hij heeft bij de specificatie en implementatie van formulieren binnen GX WebManager.

4.3.1 Huidige Problemen

Gedurende de interviews kwamen er verschillende problemen aan bod. Ook uit eerdere case studies bleek dat de stap van functionaliteit in de vorm van use cases en requirements naar een geconfigureerd CMS Applicatie vaak te groot is (Weerd, et al. 2006) (Souer, et al. 2007).

Een eerste probleem is dat het specificeren van formulieren binnen WebManager complex is en niet intuïtief (Amelsfoort 2008), (Koenen 2008), (Ladage 2008). Dit geldt zowel voor de geïnterviewden als de ervaring die ze hebben met klanten. Zeker complexere formulieren, waarbij je verschillende stappen wel of niet doorloopt aan de hand van eerder gegeven antwoorden, zijn lastig om te beheren, omdat je snel het overzicht verliest. In verschillende projecten wordt er vaak al een modelletje getekend, waarin de relatie tussen de verschillende stappen duidelijk wordt.

Een ander probleem, is het feit dat een gemaakt formulier lastig overdraagbaar is aan iemand anders (Bovy 2008), (Mierlo 2008). Dit omdat er naast de bestaande informatie, zoals aanwezig in de Use Cases en de requirements, geen extra documentatie voorhanden is. Iemand anders moet zich vervolgens weer opnieuw gaan verdiepen in de verschillende stappen, handlers, routers en validators die worden gebruikt. Dit kost vaak onnodig veel tijd.

De verschillende problemen monden uit in verscheidene doelen voor de te maken modelleertaal.

Inzicht in de flow

Het inzicht geven in de flow zou een van de belangrijkste punten van de modelleertaal moeten zijn (Amelsfoort 2008), (Hoogerwerf 2008), (Koenen 2008), (Ladage 2008). Met de huidige manier van werken wordt er vaak al een modelletje getekend om de relaties tussen verschillende stappen duidelijk te maken. Deze sluit echter niet altijd goed aan op de ontwikkeling van de formulieren.

Verbeterde communicatie

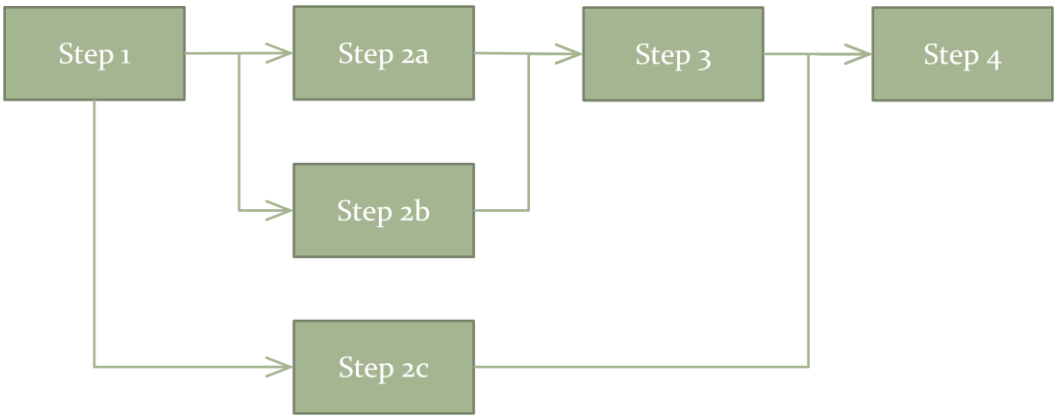
De modelleertaal zal communicatie ten goede komen. Dit werkt verschillende kanten op. Richting de klant is het makkelijker communiceren omdat hij een beter idee krijgt welke stappen er zijn en hoe ze met elkaar verbonden zijn. Dit zonder technische details, die voor de klant niet van belang zijn. Het is hiermee eenvoudiger voor de klant om te verifiëren of de juiste zaken geïmplementeerd worden. Daarnaast kan het model, eventueel met meer details, gebruikt worden tijdens de implementatie. Ook voor het testen van de werking van formulieren en het bieden van ondersteuning na afloop van een project kan de modelleertaal hulp bieden (Amelsfoort 2008), (Bovy 2008), (Hoogerwerf 2008), (Koenen 2008), (Ladage 2008), (Mierlo 2008).

Om deze communicatie op verschillende niveaus kunnen bieden, is het van belang de modelleertaal verschillende abstractieniveaus ondersteund.

4.3.2 Mental Model

Gebruikers vertelden tijdens het interview over interessante casussen, met betrekking tot formulieren, die ze zelf in het verleden hebben uitgevoerd. Door ze hiervan ook een schematisch overzicht te laten tekenen, wordt inzicht verkregen in hoe ze de verbanden tussen de verschillende stappen, routers, handlers en validators zien: hun Mental Model. Een overzicht van deze afbeeldingen is te zien in Bijlage I.

Wat opvalt, is dat iedereen een splitsing maakt tussen de weergave van de flow enerzijds en een specifieke stap anderzijds. Dit komt waarschijnlijk omdat je bij de specificatie van de flow je niet wilt bezighouden met de concrete invulling van een stap, maar daarvan wilt abstraheren.



Figuur 8 De meeste geïnterviewden tonen de flow als blokjes verbonden door pijlen, bijvoorbeeld zoals hier.

Het specificeren van een stap, wordt meestal niet echt beschreven. Vaak wordt verwezen naar de huidige implementatie binnen GX WebManager, waarbij de stappen ook apart van de flow worden gedefinieerd. Een geïnterviewde maakte wel onderscheid tussen verschillende invoervelden (Koenen 2008). In zijn model kon er een invoerveld van een bepaald type worden gekozen. Bijvoorbeeld tekstveld, getalveld, postcodeveld, datumveld, enzovoorts. Dit in tegenstelling tot de huidige implementatie van GX WebManager. Hier wordt een invoerveld gespecificeerd met daaraan gekoppeld een validator die controleert of de invoer een geldige postcode, getal of datum is. Deze representatie sluit echter niet aan bij het Mental Model, maar meer bij het Implementation Model.

Daarnaast specificeren de meeste geïnterviewden de flow middels blokjes die met behulp van pijlen met elkaar verbonden zijn, zie Figuur 8.

Opmerkelijk is dat het getekend model hier vaak ophoudt. Echter, navraag leert dat naast stappen en routers ook validators en handlers in dit plaatje gewenst zijn. Hoe dit zou moeten worden weergegeven, vindt men lastig uit te drukken.

Een tweetal tekent een validator als een soort van sluis voordat een router een stap verlaat, zie Figuur 9. Dit omdat een validator de voortgang van de flow blokt, als de ingevoerde data niet aan de eisen voldoet. Hoe je binnen deze notatie aangeeft welke validators gebruikt worden, is niet duidelijk.

Één geïnterviewde tekent de interactie als iets wat lijkt op een UML Activity Diagram of een Business Process Model (Bovy 2008). Deze persoon geeft wel aan kennis te hebben van Business Process Modeling. Door het op deze wijze modelleren van de interactie is er ruimte voor zowel routers, validators en handlers. Ook de volgorde waarin deze moeten worden uitgevoerd is duidelijk zichtbaar.

Opvallend was dat een andere geïnterviewde, die ook op de hoogte was van BPML, aangaf dat het verstandig was om bijvoorbeeld aan die standaard te houden (Ladage 2008). Dit omdat hier al

Figuur 9 Een voorbeeld van een visualisatie van een validator (Hoogerwerf 2008), (Ladage 2008).

bestaande tools voor zijn, en mensen bekend zijn met deze standaard. Echter, hij tekende het model zoals Figuur 8 en Figuur 9, wat aangeeft dat zijn Mental Model er niet direct uit ziet als BPML.

4.4 Conclusie

Het is belangrijk dat de te ontwikkelen modelleertaal aansluit bij de gebruikers. Dit wordt bereikt door de weergave (de implementatie) aan te laten sluiten bij het Mental Model van de gebruiker. Het belangrijkste doel van de taal is het inzichtelijk brengen van de flow en het verbeteren van de communicatie. Deze communicatie vindt plaats richting verschillende groepen gebruikers, waardoor een verschillend abstractieniveau gewenst is.

Verschillende abstractieniveaus komen ook terug wanneer gevraagd wordt om een model te tekenen zoals de gebruikers het voorstellen. Dit wordt door de meesten opgedeeld in twee delen. In het ene deel, wordt een detail beschrijving gewenst van een enkele stap. Het andere model beschrijft de flow, waarbij een abstracte weergave van de stap wordt gebruikt. De flow werd getekend als stappen verbonden door pijlen.

Een enkeling maakt een verwijzing naar Business Process Modelling Language (BPML) of UML Activity Diagrams. Deze sluiten echter niet precies aan bij het mental model van de gebruiker. Een mogelijke oplossing zou kunnen zijn om het formele model achter BPML of UML Activity Diagrams te gebruiken (Implementation Model) en daar een andere representatie aan te geven die beter aansluit bij het Mental Model van de gebruiker.

H5 Vergelijking bestaande modelleertalen

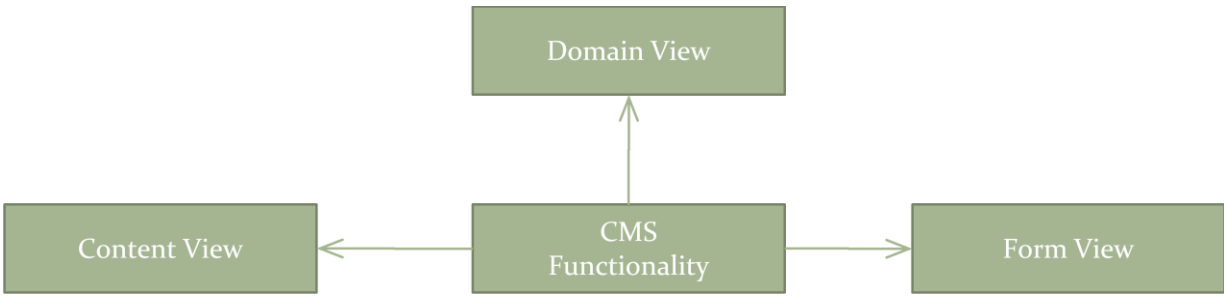
In dit hoofdstuk worden bestaande modelleermethoden met de gewenste situatie vergeleken. Dit geeft inzicht in bestaande modelleertalen en inzicht in welke concepten eventueel gebruikt kunnen worden. Dit gebeurt door verschillende eigenschappen puntsgewijs met elkaar te vergelijken aan de hand van een framework. Hieronder wordt eerst het Framework uitgelegd. Vervolgens worden er verschillende webontwerpmethoden beschreven, waarna vervolgens de vergelijking tussen de bestaande modelleertalen en de gewenste modelleertaal plaats vindt.

5.1 Het Framework

Voor de vergelijking tussen de verschillende modelleertalen, wordt er gebruik gemaakt van een aangepaste versie van het CREWS Framework (Rolland, Achour, et al. 1998). Een variant hierop is ook gebruikt voor de classificatie van ERP-modelleer technieken (Rolland and Prakash, Bridging the Gap Between Organisational Needs and ERP Functionality 2000), (Soffer, Golany and Dori 2003). Net als het CREWS Framework bestaat dit framework uit verschillende views. Elke view bestaat uit verschillende facetten met verschillende attributen. Deze facetten beschrijven verschillende eigenschappen van de modelleertalen die we willen vergelijken. In Figuur 10 zijn de verschillende views weergegeven: de Content View, Form View en Domain view. De content view bekijkt in welke mate de aanwezige kennis wordt meegenomen in het model. De form view bekijkt de structuur en de gebruikte notatie. De domain view vergelijkt de entiteiten uit het domein met de entiteiten in de modelleertaal.

De vergelijking vindt plaats door de attributen van de facetten uit te zetten tegen de verschillende modelleermethoden in een tabel. Hoe meer de modelleermethode aan de gewenste attributen voldoet hoe beter.

De verschillende views en hun facetten zijn hieronder verder beschreven.



Figuur 10 Het Framework gebruikt voor de vergelijking

5.1.1 Content View

De content view beschrijft in welke mate aanwezige kennis, die niet binnen het domein valt, wordt meegenomen. Dit aan de hand van drie facetten: abstractie, context en argumentatie.

Abstractie

Het abstractiefacet heeft één attribuut, welke de mogelijkheid van abstractie aangeeft van het model. Dit is een van de volgende waarden:

- *Flexible*: Verschillende maten van abstractie mogelijk binnen één model;
- *Fixed*: Wanneer een abstractieniveau is gekozen voor een model, kan deze niet meer veranderen;
- *Unique*: Er is maar één abstractie laag mogelijk, welke al vast ligt.

Context

De context bestaat uit drie booleaanse attributen: intern, interactie en contextueel. Het *intern* attribuut geeft aan of het interne systeem kan worden gemodelleerd. Het *interactie* attribuut geeft aan of interactie tussen het systeem en zijn omgeving kan worden uitgedrukt. Het *contextueel* attribuut geeft weer of het model de omgeving en zijn context kan weergeven.

Argumentatie

De argumentatie facet beslaat het attribuut *argument*. Het *argument* attribuut geeft aan of de taal in staat is om argumenten ten aanzien van keuzes te modelleren. Dit kan nuttige documentatie zijn, maar is niet verplicht.

In Tabel 1 zijn alle facetten en attributen nog eens opgesomd. Sommige waarden hebben een sterke voorkeur. Deze zijn hierin ook weergegeven. Ze zijn gebaseerd op gesprekken met verschillende soorten gebruikers.

Facet	Attribuut	Omschrijving	Gewenste Waarde
Abstractie	Abstractie: ENUM {Unique, Fixed, Flexible}	De mate van abstractie	Flexible
Context	Intern: BOOLEAN	Het model geeft de interne werking van het systeem weer	TRUE
	Interactie: BOOLEAN	Het model geeft de interactie tussen de omgeving en het systeem weer	TRUE
	Contextueel: BOOLEAN	Het model geeft de omgeving en zijn context weer	
Argumentatie	Argument: BOOLEAN	Het model heeft de mogelijkheid om argumenten t.a.v. keuzes te hebben	TRUE

Tabel 1 De verschillende attributen van de Content View

5.1.2 Form View

De form view geeft informatie over de notatie en structuur van de modelleertaal, door de facetten *notatie*, *structuur* en *perspectief*.

Notatie

Het notatie facet heeft één attribuut, *notatie*, welke aangeeft hoe formeel de modelleertaal is. Formeel of Semi-formeel hebben de voorkeur omdat hierbij gedeelten van het WCMS gegenereerd kunnen worden.

Structuur

Het structuur facet heeft drie attributen: element, intra-element relatie en inter-element relatie. Het *element* attribuut beschrijft de belangrijkste element typen in het model. Zo heeft UML onder andere een klasse, interface en een associatie (Rumbaugh, Booch and Jacobson 1999). De *intra-element relatie* beschrijft of het model geneste structuren toestaat. Een geneste structuur komt overeen met een flexibele abstractie laag (zie Content View). Het *inter-element relatie* attribuut beschrijft de beschikbare relaties tussen de elementen. Dit kan bestaan uit de volgende waarden:

- *Composition*: Geeft aan of er een samenstelling van elementen gedefinieerd kan worden. Hierbij hoort een onderdeel bij één enkel geheel. Bijvoorbeeld wielen bij een auto. Elk wiel hoort bij maar één auto en zonder wielen is de auto geen auto (Henderson-Sellers and Barbier 1999).
- *Generalization*: Geeft aan of er hiërarchische relaties gedefinieerd kunnen worden. Dit zijn relaties tussen een generiek element en een specifiek element. Voertuig is bijvoorbeeld een generalisatie van auto (Evans and Kent 1999).
- *Precedence*: Geeft aan of er een volgorde van elementen gedefinieerd kan worden, bijvoorbeeld een workflow (Baker, Baker and Campbell 2003).
- *AND*: Geeft aan of er een AND splitsing gedefinieerd kan worden. Een splitsing van een proces in meerder takken middels een AND geeft aan dat alle takken uitgevoerd moeten worden.
- *OR*: Geeft aan of er een OR splitsing gedefinieerd kan worden. Een splitsing van een proces in meerdere takken middels een OR geeft aan dat er enkele takken uitgevoerd kunnen worden.
- *XOR*: Geeft aan of er een XOR splitsing gedefinieerd kan worden. Een splitsing van een proces in meerder takken middels een XOR geeft aan dat er één tak moet worden uitgevoerd.

Perspectief

Het perspectief facet was oorspronkelijk niet aanwezig in het CREWS framework. Het heeft een enkel attribuut, perspectief, welke de verschillende modelleertalen indeelt in verscheidene klassen (Krogstie 1995). Dit wordt gedaan aan de hand van de eigenschappen van de modelleertaal. De verschillende klassen (dus ook de mogelijke waarden voor dit attribuut) zijn:

- *Structural*: Talen voor data modelleren.
- *Functional*: Talen waarbij de belangrijkste klasse een proces voorstelt.
- *Behavioral*: Talen waarbij de belangrijkste klassen state en transitie zijn.
- *Rule*: Regelgebaseerde talen.
- *Object*: Talen gebaseerd op object oriëntatie.
- *Communication*: Talen gebaseerd op taal/actie theorie van taalwetenschappen. Hierbij vinden acties plaats op basis van communicatie. Elke deelnemer aan een actie heeft zijn eigen doelen. Door middel van verschillende manieren van communicatie probeert hij deze doelen te bereiken.
- *Actor & Role*: Talen waarbij gebruikt wordt gemaakt van actoren en rollen.

In Tabel 2 worden alle facetten en attributen van de Form View nog eens weergegeven. Ook staat hierbij aangegeven welke waarde er gewenst zijn voor de bekeken modelleertalen.

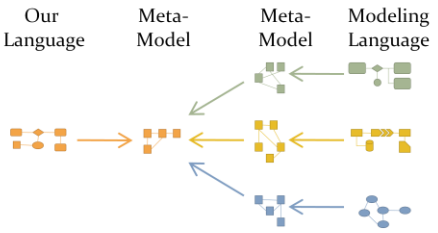
Facet	Attribuut	Omschrijving	Gewenste Waarde
Notatie	Notatie: ENUM {Formal, Semi-formal, Informal}	De mate van formaliteit	Formal, Semi-formal
Structuur	Element: SET	De belangrijkste element typen in het model	
	Intra-element relatie: SET (ENUM {Flat, Nested})	De mogelijkheid tot het maken van geneste structuren	Nested
	Inter-element relatie: SET (ENUM {Composition, Generalization, Precedence, AND, OR, XOR})	Mogelijke relaties tussen elementen	Precedence
Perspectief	Perspectief: ENUM {Structural, Functional, Behavioral, Rule, Object, Communication, Actor & Role}	Specificeert in welke klasse van modelleertalen de taal valt.	Functional

Tabel 2 De verschillende attributen van de Form View

5.1.3 Domain View

In de domain view wordt er een koppeling gelegd tussen te entiteiten uit het domein, en de verschillende modelleerconcepten uit de bekeken modelleertalen.

Het domeinmodel geeft de concepten weer van de te vormen modelleertaal. Een populatie van het domeinmodel is dus een model in de modelleertaal. Het domeinmodel is dus het metamodel van de modelleertaal.



Figuur 11 Entiteiten uit het metamodel van verschillende talen vormen het metamodel van de te vormen taal.

De te maken modelleertaal is afgeleid van bestaande modelleertalen. Op basis van de vergelijking die in dit hoofdstuk wordt gemaakt, worden er concepten uit verschillende modelleertalen gebruikt om de nieuwe modelleertaal te definiëren. Deze entiteiten kunnen dan op metaniveau aan elkaar gekoppeld worden, zie Figuur 11.

Het is van belang dat alle onderdelen uit het domein onderdeel vormen van de modelleertaal. Al deze entiteiten uit het domeinmodel moeten dus ook terugkomen in het metamodel. Er moet dus een koppeling komen tussen de entiteiten uit het metamodel van bestaande modelleertalen en het domeinmodel, zodat alles wordt afgedekt.

De Domain view beschrijft welke entiteiten uit een modelleertaal overeenkomen met entiteiten uit het domeinmodel. Dit wordt beschreven aan de hand van het *entiteit* facet.

Stel, de modelleertaal wordt een samentrekking van n modelleertalen. Formeel geldt dan:

$$MM \subseteq \left(\bigcup_{i=1}^{n} MM_i \right) \cup MM_{new}$$

Hierbij zijn MM_1, MM_2, …, MM_n de metamodellen van de bestaande modelleertalen en is MM de het metamodel van de gewenste taal. Daarnaast beschrijft MM_{new} de elementen die niet uit bestaande metamodellen kunnen worden afgeleid.

De koppeling tussen elementen in het domeinmodel en elementen uit het metamodel is gedefinieerd door de functie *map*. Deze partiële functie koppelt een element uit het metamodel aan een element uit het domeinmodel.

$map: MM \longmapsto D$

Nu moet gelden dat alle entiteiten uit het domeinmodel zijn afgedekt met een entiteit uit het metamodel. Dit is gedefinieerd in Axioma 1.

$$\forall d \in D(\exists m \in MM[map(m) = d])$$

Axioma 1 Alle elementen uit het domeinmodel dienen te worden afgedekt door een entiteit uit het metamodel.

Entiteit

Het *entiteit* facet beschrijft hoe de verschillende entiteiten uit het domein gelinkt kunnen worden aan elementen van de bekeken modelleertaal. Dit is dus casus afhankelijk en daarom niet opgenomen in dit framework. De entiteiten uit de verschillende modelleertalen zijn opgesomd in het *element* attribuut van het facet *structuur* van de Form View.

5.2 Modelleertechnieken

Het zojuist geschetste framework wordt gebruikt om verschillende modelleertalen te vergelijken. We gebruiken hier modelleertalen die in verschillende Web Engineering technieken worden toegepast. De verschillende Web Engineering methoden en hun modelleertechnieken worden hieronder toegelicht.

5.2.1 Object-Oriented Hypermedia Design Model

Het Object-Oriented Hypermedia Design Model (OOHDM) is een van de eerste modelleer methoden voor het web. OOHDM omvat een gefaseerd proces, waarbij met verschillende modellen een hypermedia applicatie wordt ontworpen. Het begint met het verzamelen van de requirements, gevolgd door het conceptueel ontwerp. Vervolgens wordt de navigatie in kaart gebracht waarna de gebruikers interface wordt ontworpen. Ten slotte vindt de implementatie plaats (Schwabe and Rossi 1998).

Requirements Engineering

De requirements worden verzameld aan de hand van gesprekken met gebruikers en stakeholders en voorbeelddocumentatie. Hieruit worden verschillende rollen, taken scenario's, Use Cases en requirements afgeleid. Hieruit wordt vervolgens het **User Interaction Diagram** (**UID**) afgeleid. Dit diagram beschrijft de informatiestroom tussen de gebruiker en het systeem, zonder dat daar gebruikersinterface aspecten in genoemd worden (Güell, Schwabe and Vilain 2000), (Vilain, Schwabe and Sieckenius de Souza 2000).

Conceptueel Ontwerp

Hierbij wordt een conceptueel model gemaakt, waarin de objecten en relaties uit het domein worden weergegeven. Hierbij wordt gebruik gemaakt van een **Conceptual Class Schema**, welke is overgenomen van UML (Rumbaugh, Booch and Jacobson 1999). Daarnaast worden Class and Relationship Cards (CRC-cards) gebruikt als ondersteunende documentatie.

Navigatie Ontwerp

Het navigatie model is een variant van het conceptueel model. Voor elke gebruikersgroep kan er een andere variatie zijn, welke aansluit bij die groep gebruikers. Het navigatie ontwerp bestaat uit twee schema's, het **Navigational Class Schema** en het **Navigational Context Schema**. De verschillende objecten waartussen genavigeerd kan worden, worden gedefinieerd in het Navigational Class Schema. Dit is een variant van het conceptueel model. Het Navigational Context Schema bestaat uit verschillende nodes, links en klassen. Het geeft aan op welke manier gegevens gegroepeerd en geselecteerd worden en welke informatie bij elkaar staat.

Interface Ontwerp

Het interface ontwerp beschrijft hoe de navigatie in de interface is verwerkt, aan de hand van **Abstract Data Views (ADV)**. ADVs zijn objecten met een state en een interface. De gebruiker kan acties uitvoeren op de interface. De interface objecten zijn gerelateerd aan navigatie objecten zodat de navigatie hier in naar voren komt. Verder kan het geneste ADVs en primitieve entiteiten (zoals tekstvelden en knoppen) bevatten.

Implementatie

In de implementatie fase wordt de uiteindelijke hypermedia applicatie geïmplementeerd. Hierbij worden de interface objecten gelinkt aan implementatie objecten. Hierbij wordt ook de architectuur (bijvoorbeeld Client Server modellen) gekozen.

De verschillende fasen en bijbehorende modellen van OOHDM zijn in Tabel 3 nog eens weergegeven.

Fase	Model	Omschrijving
Requirements Engineering	User Interaction Diagram (UID)	Beschrijft de informatiestroom tussen de gebruiker en het system.
Conceptueel Ontwerp	Conceptual Class Schema	Beschrijft de objecten en relaties uit het domein.
Navigatie Ontwerp	Navigational Class Schema, Navigational Context Schema	Het Navigational Class Schema geeft weer hoe er tussen verschillende objecten genavigeerd kan worden. Het Navigational Context Schema geeft weer hoe informatie gegroepeerd en geselecteerd wordt.
Interface Ontwerp	Abstract Data View	Relatie tussen navigatie objecten en de gebruikers interface.
Implementatie	-	De Implementatie van de hypermedia applicatie.

Tabel 3 Een overzicht van de verschillende modellen welke worden gebruikt binnen OOHDM

5.2.2 Web Modeling Language

De Web Modeling Language (WebML) is een ontwerpproces waarbij door middel van modellen en tools een Web Applicatie wordt gespecificeerd. Het proces bestaat uit verschillende fasen welke op een iteratieve manier worden uitgevoerd. Het begint bij de specificatie van requirements. Vervolgens worden de processen in kaart gebracht middels het proces ontwerp. Dit wordt gevolgd door het data ontwerp en de hypertext ontwerp. Na de implementatie, aan het einde van de iteratie, vind er een evaluatie plaats, waarna het systeem eventueel wordt aangepast (Ceri, Fraternali and Bongio 2000), (Brambilla, et al. 2006).

Requirements Specificatie

Gedurende de requirements specificatie verzamelen analisten de benodigde requirements van de Web Applicatie. Bijvoorbeeld de doelen van de site, de doelgroep, voorbeelddocumentatie, eventuele beperkingen en documentatie van al aanwezige systemen.

Proces Ontwerp

In deze fase worden de processen in kaart gebracht. De processen zijn de activiteiten die met de Web Applicatie worden uitgevoerd. Dit gebeurt middels een **Process Model**. Dit process model is afgeleid van de Business Process Modeling Language (BPML). Hierin wordt de workflow van de gebruiker in kaart gebracht.

Data Ontwerp

Hierbij worden de concepten en hun relaties uit het domein vastgelegd. Dit kan met een UML Class Diagram, maar meestal wordt er binnen WebML gebruik gemaakt van een **Entity-Relationship-diagram (ER-Diagram)**.

Hypertext Ontwerp

Het hypertext ontwerp specificeert welke elementen op welke pagina's terug komen en hoe ze verbonden zijn zodat is een netwerk vormen. Deze specificatie wordt gedaan doormiddel van het **Hypertext Model**. Het hypertext model bevat verschillende units die informatie geven over een dataset. Bijvoorbeeld een data unit, welke specifieke informatie geeft over één item, of een index unit welke een lijst geeft van aanwezige items.

De verschillende fasen en bijbehorende modellen van WebML zijn in Tabel 4 nog eens weergegeven.

Fase	Model	Omschrijving
Requirements Specificatie	-	Specificeert wat de Web Applicatie moet doen.
Proces Ontwerp	Process Model	Specificeert de workflow van de gebruiker.
Data Ontwerp	ER-Diagram	Beschrijft de entiteiten en relaties in het domein.
Hypertext Ontwerp	Hypertext Model	Specificeert welke elementen op welke pagina's staan en hoe ze onderling verbonden zijn.

Tabel 4 Een overzicht van de verschillende modellen die gebruikt worden binnen WebML

5.2.3 UML-based Web Engineering

UML-based Web Engineering (UWE) is een software ontwikkeling proces voor Web Applicaties gebaseerd op UML. Er wordt gebruik gemaakt van standaard UML diagrammen, omdat deze veelal bekend zijn bij de gebruikers (modelleurs, programmeurs, etc.). UWE bestaat uit verschillende fasen, te beginnen bij de requirements specificatie. Dit wordt gevolgd door het conceptueel ontwerp, het navigatie ontwerp en het presentatie ontwerp. Tenslotte worden de uit te voeren taken geanalyseerd (Koch, Software Engineering for Adaptive Hypermedia Systems: Reference Model, Modeling Techniques and Development Process 2000), (Koch and Kraus, The expressive Power of UML-based Web Engineering 2002).

Requirements Specificatie

In deze fase wordt er bepaald wat de applicatie moet gaan doen. Aan de hand van deze requirements worden Use Cases opgesteld, welke aan de basis staan van het **Use Case Model**. Hierbij worden de relaties tussen de verschillende Use Cases en gebruikers in kaart gebracht.

Conceptueel Ontwerp

Het conceptueel ontwerp beschrijft de samenhang tussen de verschillende concepten in het domein. Dit wordt gedaan aan de hand van een **UML Class Diagram**. Hierbij worden zaken als navigatie, presentatie en interactie zo veel mogelijk genegeerd.

Navigatie Ontwerp

De navigatie ontwerp fase specificeert de structuur van de Web Applicatie en beschrijft hoe de navigatie plaats vindt. Dit op basis van het conceptueel model. De navigatie wordt gespecificeerd door een tweetal modellen: Het **Navigation Space Model** en het **Navigation Structure Model**. Het Navigation Space Model beschrijft welke objecten bereikt kunnen worden door middel van navigatie. Het Navigation Structure Model beschrijft hoe deze bereikt kunnen worden.

Fase	Model	Omschrijving
Requirements Specificatie	Use Case Model	Beschrijft de relaties tussen de Use Cases en gebruikers.
Conceptueel Ontwerp	Class Diagram	Specificeert de samenhang tussen concepten in het domein.
Navigatie Ontwerp	Navigation Space Model, Navigation Structure Model	Het Navigational Space Model beschrijft welke objecten bereikt kunnen worden door navigatie. Het Navigation Structure Model beschrijft hoe ze bereikt kunnen worden.
Presentatie Ontwerp	Abstract User Interface Model	Beschrijft hoe navigatie objecten worden weergegeven aan de gebruiker.
Taakanalyse	Activity Diagram	Beschrijft de acties, transities en vertakkingen waaruit een taak bestaat.

Tabel 5 Een overzicht van de verschillende modellen die gebruikt worden binnen UWE

Presentatie Ontwerp

Het presentatie ontwerp beschrijft waar en hoe navigatie objecten worden weergegeven aan de gebruiker. Het **Abstract User Interface Model** wordt gemaakt als een UML Class Diagram. Hierbij wordt gebruik gemaakt van UML Stereotypes zoals «text», «form», «button», «image» en «anchor».

Taakanalyse

Een taak bestaat vaak uit verschillende acties en wordt uitgevoerd om een doel van de gebruiker na te streven. UWE gebruikt hiervoor **UML Activity Diagrams**. Deze beschrijven acties, transities en vertakkingen, waardoor een proces in beeld wordt gebracht.

De verschillende fasen en bijbehorende modellen van UWE zijn in Tabel 5 nog eens weergegeven.

5.2.4 Object-Oriented Web Solution

Object-Oriented Web Solution (OOWS) is een methode voor het ontwikkelen van Web Applicaties. OOWS is opgebouwd uit twee stappen: conceptual modelling en solution development. Conceptual modelling bestaat uit verschillende stappen. Eerst worden de functionele requirements verzameld. Op basis hiervan wordt er een conceptueel model gemaakt, en een proces ontwerp. Daarna wordt er een navigatie en presentatie model gecreëerd.

De tweede stap, solution development, is een architectuur gekozen voor de implementatie, en wordt deze geïmplementeerd. Dit doormiddel van de presentatie, de applicatie, en de persistentie laag (Pastor, Fons and Pelechano 2003).

Functionele Requirements Analyse

Tijdens deze fase worden de verscheidene requirements achterhaald. Aan de hand van Use Cases en scenario's worden deze in kaart gebracht.

Conceptueel Ontwerp

Aan de hand van de opgestelde requirements en Use Cases wordt het conceptueel ontwerp gemaakt. Dit bestaat uit een UML **Class Diagram**. Hierbij worden de verschillende objecten en hun relaties uit het domein duidelijk.

Proces Ontwerp

Het proces ontwerp beschrijft de processen tussen het systeem en de gebruiker. Dit wordt gedaan aan de hand van **Business Process Models**, welke zijn overgenomen van de Business Process Modeling Notation (BPMN) (Torres, Giner and Pelechano 2007), (White, Business processing modeling notation (BPMN), version 1.0 2004). Business Process Models hebben als doel leesbaar te zijn voor verschillende stakeholders. Ze beschrijven het verloop van een proces en de daarbij horende interactie tussen de gebruiker, het systeem en eventueel externe systemen.

Fase	Model	Omschrijving
Funcionele Requirements Analyse	-	Aan de hand van Use Cases en scenario's worden de requirements in kaart gebracht.
Conceptueel Ontwerp	Class Diagram	Geeft de objecten en hun relaties uit het domein weer.
Proces Ontwerp	Business Process Model	Beschrijft het verloop van een proces en de interactie tussen gebruiker en systeem.
Navigatie Ontwerp	Navigational Map, Navigational Context Model	De Navigational Map geeft de navigatiemogelijkheden weer. Het Navigational Context Model beschrijft welke klassen er binnen de context vallen.
Presentatie Ontwerp	Presentation Model	

Tabel 6 Een overzicht van de verschillende modellen die gebruikt worden binnen OOWS

Navigatie Ontwerp

Het navigatie ontwerp specificeert een navigatie variant voor elke relevante gebruikersgroep. Dit wordt gedaan door eerst een gebruikersanalyse te maken, waarna het navigatie diagram wordt gemaakt. Dit bestaat uit twee modellen: de **Navigational Map** en het **Navigational Context Model**. De Navigational Map is een gerichte graaf die de navigatie tussen de verschillende items voorstelt. De items zijn navigational contexten welke worden gedefinieerd in het Navigational Context Model. Dit wordt gemaakt op basis van het Class Diagram.

Presentatie Ontwerp

In deze fase wordt het **Presentation Model** gespecificeerd op basis van het Navigational Context Model. Dit wordt gedaan aan de hand van presentatie patterns.

De verschillende fasen en bijbehorende modellen van OOWS zijn in Tabel 6 nog eens weergegeven.

5.2.5 Web Software Architecture

Web Software Architecture (WebSA) is een Model-driven benadering van het ontwikkelen van Web Applicaties. Het sluit aan bij de Model-driven standaarden van het OMG, waardoor transformatie naar andere standaarden/talen mogelijk is. De te ontwikkelen Web Applicatie wordt vanuit verschillende standpunten bekeken. Vanuit een functioneel standpunt worden domein, proces en navigatie modellen gemaakt. Vanuit het architectuur standpunt wordt het configuratie model gemaakt. Ten slotte wordt het integratie model gemaakt (Melia and Gómez, Applying Transformations to Model Driven Development of Web Applications 2005), (Meliá, Gómez and Koch 2005).

Functioneel Standpunt

Vanuit dit standpunt wordt de functionaliteit van de te ontwikkelen Web Applicatie bekeken. Hierbij wordt een domein, proces en navigatie model gemaakt. Welke technieken hiervoor worden gebruikt, maakt niet zo heel veel uit, en ligt aan de voorkeur van de gebruiker. Aanbevolen wordt wel om het aan te laten sluiten bij de Model-driven standaarden van het OMG,

zodat transformatie naar andere modellen mogelijk is. Dit zijn bijvoorbeeld de eerder beschreven UML Class Diagram, Activity Diagram en BPMN.

Architectuur Standpunt

Vanuit het architectuur standpunt wordt een **Configuration Model** gemaakt. Dit model beschrijft verscheidene webcomponenten en hoe deze met elkaar verbonden zijn. Het configuration model is een uitbreiding op het UML Composite Structure Model. Elk component beschrijft een rol of taak die het vervult.

Integration Model

Het **integration model** beschrijft, platform onafhankelijk, het totale ontwerp van de applicatie. Deze wordt gemaakt aan de hand van de eerdere modellen. Dit gebeurt automatisch aan de hand van een *PIM-to-PIM transformatie*. PIM staat voor Platform Independent Model (Melia and Castro, An MDA approach for the development of Web applications 2004).

De verschillende fasen en bijbehorende modellen van WebSA zijn in Tabel 7 nog eens weergegeven.

Fase	Model	Omschrijving
Functioneel Standpunt	Domain Model, Process Model, Navigational Model	De keuze welke modelleertechniek wordt gebruikt is vrij aan de gebruiker, zolang het aansluit bij de Model-driven standaarden van het OMG.
Architectuur Standpunt	Configuration Model	Beschrijft de webcomponenten en hun relaties.
Integratie Model	Integration Model	Beschrijft het totale ontwerp van de applicatie.

Tabel 7 Een overzicht van de verschillende modellen die gebruikt worden binnen WebSA

5.3 Het Framework toegepast

Het aangepaste CREWS framework wordt toegepast op de zojuist beschreven Web Engineering technieken. Dit gebeurt door middel van een matrix, waarbij voor elke modelleertaal wordt aangegeven in hoeverre ze aan de verschillende attributen van het framework voldoen. De attributen zijn gegroepeerd per facet en per view. De modelleertalen zijn gegroepeerd per Web Engineering techniek.

Op basis van deze matrix en de gewenste waarden voor de verschillende attributen, kan een selectie worden gemaakt van de interessantste modelleertechnieken. De geselecteerde technieken zijn: User Interaction Diagram, Conceptual Class Schema (beiden van OOHDM), Business Process Diagram (van WebML), Activity Diagram (van UWE) en het Business Process Model (van OOWS). Hieronder worden de belangrijkste modelleertechnieken per view bekeken. De volledige tabel is (opgedeeld in stukken) te vinden in Bijlage II.

5.3.1 Content View

De content view beschrijft in welke mate aanwezige kennis wordt meegenomen. Deze zijn weergegeven in Tabel 8. Gebaseerd hierop voldoen het Activity Diagram en het Business Process

Model het beste aan de gewenste waarden omdat ze verschillende abstractielagen weer kunnen geven en de interactie tussen de gebruiker en het systeem laten zien.

Facet	Attribuut	OOHDM		WebML	UWE	OOWS	Gewenste waarde
		User Interaction Diagram	Conceptual Class Schema	Business Process Diagram	Activity Diagram	Business Process Model	
Abstractie	Abstractie	Unique	Unique	Flexible	Flexible	Flexible	Flexible
Context	Intern	-	TRUE	TRUE	TRUE	TRUE	TRUE
	Interactie	TRUE	-	-	TRUE	TRUE	TRUE
	Contextueel	-	-	-	TRUE	-	
Argumentatie	Argument	-	TRUE	-	TRUE	TRUE	TRUE

Tabel 8 De belangrijkste modelleertalen vergeleken aan de hand van de Content View

5.3.2 Form View

De form view geeft informatie over de notatie en structuur van de modelleertaal. De matrix met de verschillende modelleertalen wordt weergegeven in Tabel 9. Vooral vanwege hun functionele perspectief zijn het Activity Diagram en het Business Process Model het interessantst.

5.3.3 Domain View

De domain view vergelijkt de entiteiten uit het domein met de entiteiten van de verschillende modelleertalen. De entiteiten uit het domein, zijn afgeleid uit de domeinanalyse. De verschillende entiteiten uit het domein zijn in Tabel 10 nog eens weergegeven. Hiërarchische structuur is hierin meegenomen doordat attributen zijn ingesprongen. Deze structuur is meegenomen omdat een modelleertaal bijvoorbeeld wel *acties* kan ondersteunen, maar niet specifiek *routers* of *handlers*. Op deze manier kan er gezegd worden van een modelleertaal op welk abstractieniveau hij het domein ondersteund.

De vergelijking tussen het domein en de modelleertalen is weergegeven in Tabel 11. De entiteiten van het Conceptual Class Schema zijn veelvuldig te mappen met de gewenste modelleertaal. Dit komt omdat het entiteit *Class* veelvuldig is te gebruiken. Het nadeel hiervan is dat er geen onderscheid is tussen de verschillende entiteiten, wat het lastig maakt voor de gebruiker om het model te interpreteren. Het Activity Diagram en het Business Process Model hebben veel notatiemogelijkheden om de flow aan te geven, maar minder om formulier elementen weer te geven. Hierin is vooral het User Interaction Diagram sterk. Het Business Process Diagram van WebML valt er een beetje tussenin, het kan een groot aantal entiteiten weergeven, maar voor met name het weergeven van een flow is het niet zo geschikt.

Facet	Attribuut	OOHDM		WebML	UWE	OOWS	Gewenste Waarde
		User Interaction Diagram	Conceptual Class Schema	Business Process Diagram	Activity Diagram	Business Process Model	
Notatie	Notatie	Semi-Formal	Formal	Formal	Formal	Formal	Formal, Semi-formal
Structuur	Element	{Initial Interaction, Interaction, Optional Interaction, Interaction Alternative, Data Entry, Optional Data Entry, Element, Text, Optional Text, New Interaction, Call of Operation}	{Class, Attribute, Operation, Interface, Association, Generalization, Aggregation, Composition, Note, Package, Directed Association}	{Start Activity, End Activity, Assign Operation, If Unit, Case Unit, Service Unit, DataUnit, IndexUnit, MultiDataUnit, EntryUnit, ScrollerUnit, Operation Unit, MultiChoice, Hierarchical, FilterUnit, DirectUnit, Page, Area, Site View, Contextual Link, Automatic Link, Transport Link, Non-Contextual Link}	{Initial Node, Final Node, Activity, Control Flow Edge, Object Flow Edge, Fork, Join, Condition, Decision, Merge, Partition, Sub-activity indicator, Flow Final, Note, Use Case}	{Event, Activity, Gateway, Sequence Flow, Message Flow, Association, Pool, Lane, Data Object, Group, Annotation}	
	Intra-element relatie	Flat	Flat	Nested	Nested	Nested	Nested
	Inter-element relatie	{Precedence}	{Composition, Generalization}	{Composition, Precedence}	{Precedence, AND, OR, XOR}	{Composition, Precedence, AND, OR, XOR}	Precedence
Perspectief	Perspectief	Actor & Role	Object	Behavioral	Functional	Functional	Functional

Tabel 9 De belangrijkste modelleertalen vergeleken aan de hand van de Form View.

Facet	Attribuut	Omschrijving	Gewenste Waarde
Entiteit	Form	Formulier	
	Step	Stap	
	Subform	Subformulier	
	Form Element	Formulier element	
	Button	Knop	
	Information	Informatie	
	Text	Tekst	
	Image	Afbeelding	
	Input	Invoer	
	Text Input Field	Tekstinvoer	
	List	Lijst	
	Upload File	Bestandsupload	
	Items	Items	
	Predefinied List	Voorgedefinieerde lijst	
	Query List	Query Lijst	
	Item	Item	
	Visitor Information Type	Bezoekersinformatie-type	
	Database Information Type	Database informatie-type	
	Session Information Type	Sessie informatie-type	
	Action	Actie	
	Handler	Handler	
	Router	Router	
	Validator	Validatieregel	
	Condition	Conditie	
	Page	Pagina	
	Dataset	Dataset	

Tabel 10 De verschillende attributen van de Domain View

5.4 Conclusie

In dit hoofdstuk hebben verschillende Web Engineering technieken geïntroduceerd. Deze technieken bestaan uit meerdere modelleertalen. Deze talen zijn met elkaar vergeleken aan de hand van een framework. Dit framework, welke is afgeleid van het CREWS framework, beschrijft de modelleertaal vanuit verschillende views. Aan de hand van gebruikersinterviews zijn ook gewenste waardes voor verschillende attributen vast gesteld.

Uit de vergelijking blijkt dat vooral het Activity Diagram van UWE en het Business Process Model van OOWS het meest voldoen aan de gewenste waardes. Ze geven de interactie tussen de gebruiker en het systeem weer, hebben verschillende abstractieniveaus, hebben een functioneel perspectief en zijn sterk in het weergeven van de flow. Er ontbreekt echter de mogelijkheid om formulier elementen weer te geven. De User Interaction Diagrams van OOHDM zijn op dit gebied

interessant, omdat ze op een eenvoudige en compacte wijze de verschillende formulier elementen kunnen weergeven. Ook het Class Diagram, door verschillende web-engineering methoden overgenomen van UML, heeft de mogelijkheid verschillende formulierelementen te modelleren. Zo zou een stap een lijst van formulier elementen kunnen hebben. Het Business Process Diagram van WebML biedt dit zelfs nog iets uitgebreider, maar heeft weer een minder compacte notatie vorm en geeft de interactie tussen de gebruiker en het systeem slecht weer.

Omdat Activity Diagram en het Business Process Model bijna hetzelfde zijn gaan we verder met het Business Process Model (White, Process Modeling Notations and Workflow Patterns 2004). Het verschil tussen beide diagramtechnieken zit hem vooral in de weergave van symbolen en de gebruikte terminologie. Beiden zijn views over hetzelfde metamodel. We kiezen het Business Process Model omdat dit grafisch beter aansluit bij de gebruikers (zie ook H4, Gebruikersanalyse).

Op basis van deze drie modelleertalen (Business Process Model (BPM), User Interaction Diagram (UID) en UML Class Diagram (UML)), specificeren we de nieuwe modelleertaal. Dit vindt plaats in het volgende hoofdstuk.

Facet	Attribuut	OOHDM		WebML	UWE	OOWS
		User Interaction Diagram	Conceptual Class Schema	Business Process Diagram	Activity Diagram	Business Process Model
Entiteit	Form	Initial Interaction	Package	Area	Activity	Activity
	Step	Interaction	Class	Page	Activity	Activity
	Subform	Interaction	Class	Page	Sub-Activity Indicator	
	Form Element		Class			
	Button		Class			
	Information		Class	DataUnit		
	Text	Text	Class	DataUnit		
	Image		Class	DataUnit		
	Input	Data Entry	Class	EntryUnit		
	Text Input Field	Data Entry	Class	EntryUnit		
	List	Data Entry	Class	MultiChoice		
	Upload File	Data Entry	Class	EntryUnit		
	Items	Element Set	Class	MultiDataUnit		
	Predefinied List		Class	MultiDataUnit		
	Query List		Class	MultiDataUnit		
	Item		Class	DataUnit		
	Visitor Information Type		Interface	DataUnit		
	Database Info. Type		Interface	DataUnit		
	Session Info. Type		Interface	DataUnit		
	Action			Operation Unit		
	Handler		Operation	Operation Unit		
	Router	New Interaction	Directed Association	Non-Contextual Link	Control Flow Edge, Decision	Sequence Flow, Gateway
	Validator		Operation	Operation Unit		
	Condition		Operation	If-Unit	Condition	Gateway
	Page		Class	Area	Activity	Activity
	Dataset		Class	MultiDataUnit		

Tabel 11 De belangrijkste modelleertalen vergeleken aan de hand van de Domain View

H6 De Modelleertaal

In hoofdstuk H5 hebben we aan de hand van het aangepaste CREWS-framework verschillende webmodelleertalen met elkaar vergeleken. Op basis van deze vergelijking hebben we een drietal talen geselecteerd, waarop de te vormen modelleertaal wordt gebaseerd. Deze geselecteerde technieken zijn nog eens opgesomd in Tabel 12. De te vormen modelleertaal, welke gebruikt wordt voor het modeleren van formulieren, zal vanaf nu als WebForm Diagram aangeduid worden.

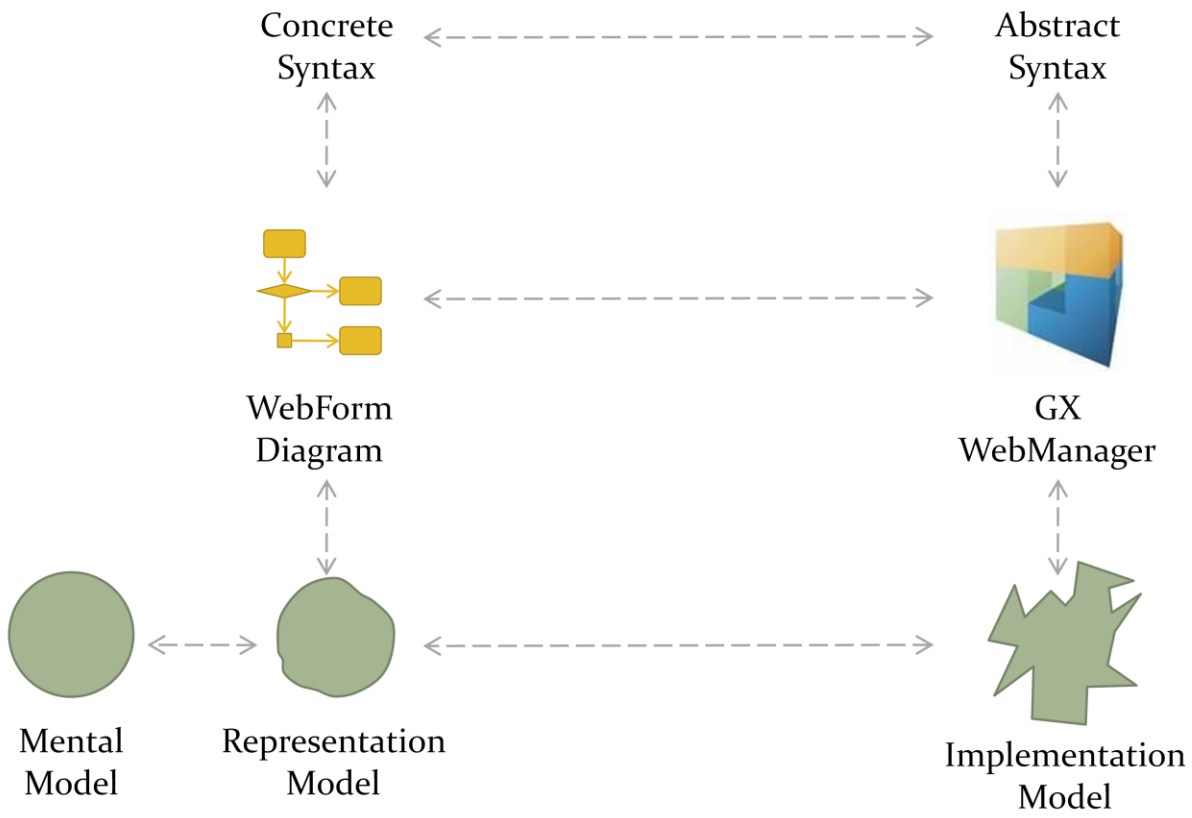
Modelleer-taal	Gebruikt in	Voordelen	Nadelen
Business Process Model (BPM)	OOWS	Weergave interactie tussen gebruiker en systeem; Verschillende abstractieniveaus; Functioneel perspectief Weergave van de flow	Geen weergave van formulier elementen
User Interaction Diagram (UID)	OOHDM	Compacte weergave van formulier elementen Weergave van de flow	Geen weergave van validators, handlers en condities
UML Class Diagram (UML)	OOHDM, WebML, UWE, OOWS	Attributen binnen een klasse kunnen formulier elementen goed weergeven	Geen grafische symbolen voor validators, handlers, condities (alleen symbolen *klasse* en *interface* zijn aanwezig)

Tabel 12 De geselecteerde modelleertechnieken die de basis vormen voor de nieuwe modelleertaal

Het metamodel van het WebForm Diagram is gebaseerd op het domeinmodel van de formulieranalyse. Het metamodel is opgebouwd uit verschillende concepten uit deze modelleertalen. De rest van dit hoofdstuk beschrijft het metamodel, opgedeeld in de concrete syntax en de abstracte syntax. Dit beschrijft ook welke concepten van bestaande modelleertalen worden gebruikt en wat de grafische notatie is van de verschillende elementen.

6.1 Abstracte syntax en Concrete syntax

Een modelleertaal bestaat uit een syntax en een semantiek. Meestal wordt eerst de syntax gedefinieerd en dan de semantiek van de taal. De syntax definieert welke constructies er bestaan in de taal en hoe deze zijn opgebouwd uit andere constructies. Zeker bij een grafische taal is het belangrijk om de syntax onafhankelijk van de notatie te definiëren. Dit is de abstracte syntax van de taal (OMG, Unified Modeling Language: Infrastructure, Version 2.0 2005). De concrete syntax is de mapping van de grafische notatie naar de abstracte syntax. De semantiek van een taal beschrijft de betekenis van de constructies.



Figuur 12 Het verband tussen de Concrete Syntax, Abstracte Syntax, de verschillende modellen van Cooper en de implementatie

De abstracte syntax is gerelateerd aan het domeinmodel en dus aan het metamodel. Dit komt overeen met het Implementation Model van Cooper (Cooper, Reimann en Cronin 2007). De grafische taal moet voldoen aan de wensen (Mental Model) van de gebruiker. Het grafische model in combinatie de concrete syntax vormen dus het Representation Model van Cooper, zie ook Figuur 12.

6.2 Concrete syntax

Het grafische model en de concrete syntax vormen een Representation Model van een formulier. Deze dienen aan te sluiten bij het mental model van de gebruiker, welke aan de hand van interviews is vastgesteld. Daarnaast hebben we verschillende bestaande (web) modelleertalen met elkaar vergeleken. Aan de hand van deze vergelijkingen en de gesprekken met de gebruikers is een grafisch model en de concrete syntax vastgesteld. Deze wordt hieronder beschreven.

Form (formulier)

Een diagram dat je tekent, is een representatie van een formulier. Een formulier kan, net als een Business Process Model (of UML Activity Diagram, (Guelfi en Mammar 2006)), User Interaction Diagram of UML Class Diagram, gezien worden als een (gerichte) graaf en is een nonupel $f = \langle N, S, FE, V, H, C, P, B, E \rangle$ waarbij N de set van Nodes is, S de verzameling stappen, FE de verzameling van formulier elementen, V de verzameling validators, H de verzameling handlers, C

de verzameling condities, P de verzameling pagina's, B de verzameling blokken en E de verzameling zijden (Edges).

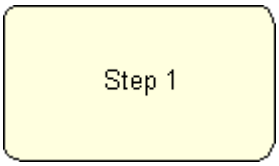
Step (stap)

Een stap is te vergelijken met een Activity (BPM), Interaction (UID) en Class (UML). Net als in de verschillende modellen vormt dit het belangrijkste onderdeel van de modelleertaal, omdat juist het weergeven van de stappen en de routering ertussen een belangrijk doel van de modelleertaal is. De grafische weergave is te zien in Figuur 13, welke gebaseerd is op gesprekken met gebruikers.

Een stap heeft een naam welke is gedefinieerd als een functie op de set S:

$$nameS: S \longmapsto String$$

Deze naam dient ter identificatie van de stap richting de gebruiker.



Figuur 13 De grafische weergave van een stap

Form Element (formulier element)

Een formulier element is te vergelijken met een Data Entry (UID) of een Attribute (UML). Het is een supertype van verschillende elementen die binnen een stap aan de gebruiker gerepresenteerd kunnen worden (bijvoorbeeld een naam-invoerveld en wachtwoord-invoerveld). De set van formulier elementen is dus ook een vereniging van verschillende velden, waarbij *INPUT* de set van invoervelden is, *BUTTON* de set van knoppen en *INFO* de set van informatie-elementen:

$$FE = INPUT \cup BUTTON \cup INFO$$

Elk formulier element is gekoppeld aan een stap. Zo heeft een stap nul of meerdere formulier elementen in een bepaalde volgorde. De functie *fields* geeft van een stap een geordende set (tupel) van form elementen. De functie *fields* heeft dan de volgende signatuur:

$$fields: S \longmapsto \mathcal{S}(FE)$$

Hierbij is $\mathcal{S}(FE)$ de set van alle mogelijke sequenties (tupels) van formulier elementen. De functie *fields* heeft hierbij de volgende eigenschappen:

$$\forall f \in FE\ \forall s_1, s_2 \in S\ \forall\ i, j \in \mathbb{N}\ [occ(f, fields(s_1), i) \wedge\ occ(f, fields(s_2), j) \Rightarrow s_1 = s_2]$$

Axioma 2 Elk formulier element is gekoppeld aan slechts één stap

Hierbij geeft de functie *occ* aan of er een element *f* op positie *i* voorkomt in de gegeven sequentie. Elk formulier element mag maar binnen één stap worden gebruikt, zie Axioma 2. Daarnaast mag

een formulier element binnen een stap ook maar één keer voorkomen. Dit is beschreven in Axioma 3.

$$\forall f \in FE \; \forall s \in S \; \forall \, i, j \in \mathbb{N} \, [occ(f, fields(s), i) \wedge \; occ(f, fields(s), j) \Rightarrow i = j]$$

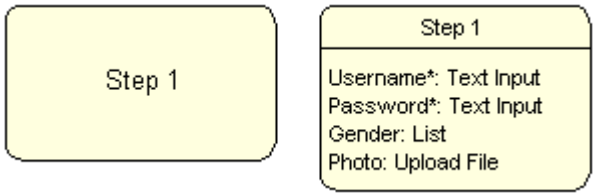
Axioma 3 Binnen één stap kan een formulier element maar één keer voorkomen

Input (invoerveld)

Een invoerveld is te vergelijken met een Data Entry (UID) of een Attribute (UML). Het is een subtype van *formulier element*. De set *INPUT* van invoervelden is dan weer een supertype voor tekstinvoer, bestandsupload en een keuzelijst:

$$INPUT = TEXTINPUT \cup UPLOAD \cup LIST$$

De grafische weergave van een invoerveld is optioneel: binnen een stap kunnen deze worden weergegeven (vergelijkbaar met Class-Attributes in UML), maar de gebruiker kan er ook voor kiezen om deze niet weer te geven, zie ook Figuur 14. Het diagram is vooral bedoeld om inzicht te geven in de verschillende stappen en routeringen daartussen, dus eventuele extra informatie kan er toe leiden dat het diagram erg druk wordt. Door toch de mogelijkheid te geven om deze informatie te tonen, kan de gebruiker zelf bepalen wat het beste aansluit bij zijn Mental Model.



Figuur 14 Links een stap waar de invoervelden niet te zien zijn, rechts dezelfde stap waarbij ze wel zichtbaar zijn.

Elk invoerveld heeft een naam welke gedefinieerd is als een functie op *INPUT*:

$$nameI: INPUT \; \longmapsto String$$

Naast de naam heeft een invoerveld een eigenschap *required* welke aangeeft of de eindgebruiker het veld verplicht in moet vullen of niet. Dit is als volgt gedefinieerd:

$$required: INPUT \longmapsto \{T, F\}$$

Text Input Field (tekstinvoer)

Een tekstveld is te vergelijken met een Data Entry (UID) of een Attribute (UML). Het is een subtype van *invoerveld*. De grafische weergave hiervan is ook hetzelfde als die van invoerveld. Naast de geërfde eigenschappen *naam* en *verplicht*, heeft het de eigenschappen *default* en *wachtwoord*. *Default* geeft een vooraf ingevulde waarde aan en is als volgt gedefinieerd:

$default: TEXTINPUT \longmapsto String \cup \{""\}$

De functie *default* geeft de default waarde van een tekstinvoer. Deze waarde kan ook een lege string zijn, wat aangeeft dat er geen voorgedefinieerde waarde is. De functie *password* geeft aan of het een wachtwoord veld is of niet. Is dit het geval, dan dient in het uiteindelijke formulier dit veld als een wachtwoordveld aan de bezoeker te worden getoond:

$password: TEXTINPUT \longmapsto \{T, F\}$

Upload File (bestandsupload)

Een bestandsupload is te vergelijken met een Data Entry (UID) of een Attribute (UML). Het is een subtype van *invoerveld.* Naast de eigenschappen van invoerveld heeft dit type een eigenschap die de maximale bestandsgrote opgeeft:

$maxSize: UPLOAD \longmapsto Integer$

List (lijst)

Ook een lijst is te vergelijken met een Data Entry(UID) of een Attribute (UML) en is een subtype van invoerveld. Naast de gebruikelijke eigenschappen, bevat een lijst een set van items waar een bezoeker uit kan kiezen. Deze set kan zijn bron in de database hebben, maar kan ook handmatig zijn ingevoerd. Deze zijn gedefinieerd in de set *LISTITEMS.*:

$itemsL: LIST \longmapsto LISTITEMS$

List Item

De set *LISTITEMS* bevat dus verzamelingen die uit de database afkomstig zijn, of handmatig is ingevoerd:

$LISTITEMS = DBITEMS \cup MANUALITEMS$

Zowel *DBITEMS* als *MANUALITEMS* bevatten sets van *ITEM* elementen, waardoor geldt:

$LISTITEMS \subseteq \wp(ITEM)$

De items die uit de database worden gelezen zijn als volgt gedefinieerd:

$$DBITEMS = \bigcup_{q \in Q} query(q)$$

Hierbij is Q een set van queries en *query* de functie die de query uitvoert op de database:

$query: Q \longmapsto \wp(ITEM)$

Een boekingsformulier van een vliegmaatschappij bevat een voorbeeld van een lijst van items die uit de database worden gelezen. De gebruiker kan opgeven naar welke bestemming hij wil vliegen. De lijst van bestemmingen wordt dan uit de database gehaald. Hierbij is q_{dest} de query die de bestemmingen uit de database leest:

$query(q_{dest}) = \{Baku, Göteborg, Johannesburg, Kampala, Madrid, Moskou, Oslo, Tbilisi\}$

De handmatig ingevoerde sets van items zijn als volgt gedefinieerd:

$MANUALITEMS \subseteq \wp(ITEM)$

Een voorbeeld van een handmatig ingevoerde lijst van items is een sekse:

$\{Man, Vrouw\} \in MANUALITEMS$

Button (knop)

Een knop is een subtype van een formulier element en maakt dus onderdeel uit van een formulier. Een knop heeft altijd een naam, wat op de knop staat weergegeven. Dit is als volgt uitgedrukt:

$nameB: BUTTON \longmapsto String$

Daarnaast is een knop ook van een bepaald type. Dit type bepaald mede wat er vervolgens gebeurd. Als een bezoeker op een bepaalde knop drukt wordt er een flag gezet. Deze kan binnen een conditie-element gebruikt worden, bijvoorbeeld om te kijken of de bewerking is geannuleerd. Dit type wordt als volgt uitgedrukt:

$typeB: BUTTON \longmapsto \{Abort, Back, Submit\}$

Information (informatie)

Het informatie object is te vergelijken met een Class uit UML. Het is een supertype en bevat verder geen eigenschappen:

$INFO = TEXT \cup IMAGE$

Text (tekst)

Een tekst is te vergelijken met een Text (UID) of een Class (UML). Het bevat alleen een eigenschap *name* wat de tekst weergeeft:

$$nameT: TEXT \longmapsto String$$

Image (afbeelding)

Een afbeelding geeft een plaatje weer aan de gebruiker. Dit object bevat alleen de fysieke afbeelding, wat tot uitdrukking komt in de functie *image*:

$$image: IMAGE \longmapsto PIC$$

Hierbij is *PIC* de verzameling van fysieke afbeeldingen.

Validator (validatieregel)

Een validatieregel zou binnen UML geïmplementeerd kunnen worden als een Operation. Dit zou een operatie kunnen zijn die dingen controleert en een boolean teruggeeft als aan wat voorwaarden voldaan wordt. Een validatieregel werd tijdens het gebruikersonderzoek getekend als een soort van sluis welke de voortgang van de executie blokt als er niet aan de validatie voldaan wordt. Daarnaast wordt er een onderscheid gemaakt tussen twee soorten validators: field validators en form validators. Een validator kan dan ook gezien worden als de vereniging van de set van field validators en form validators:

$$V = FIELDVALIDATOR \cup FORMVALIDATOR$$

Elke validator is van een bepaald type. Elk type validator heeft weer zijn eigen parameters. Zo heeft de formvalidator *Check username/password* de parameters *username* en *password* en de fieldvalidator *Check minimum length* de parameter *min length* die aangeeft wat de minimale lengte moet zijn van de invoer. Omdat ook handlers en condities te maken hebben met parameters, is dit verderop beschreven in een gezamenlijk supertype, *parameter object*.

Field validator (veldvalidatie)

Een veldvalidatie is een validatieregel op invoerveld-niveau: Deze wordt aan een veld gekoppeld en controleert of dat veld aan een bepaalde voorwaarde voldoet. Een invoerveld bevat dus een lijst van verschillende veldvalidaties:

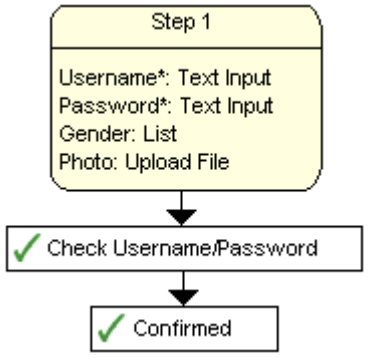
$$validationF = INPUT \longmapsto \wp(FIELDVALIDATOR)$$

Elk invoerveld is door deze functie gekoppeld aan een set van veldvalidaties. Van een veldvalidatie is geen grafische weergave, omdat dit niet tot de essentiële zaken van het diagram valt, en deze binnen een stap, binnen een veld plaats vindt.

Form validator (formvalidatie)

Een formvalidatie is een validatie of formulierstap-niveau: De invoer van verschillende velden samen, moet aan een bepaalde voorwaarde voldoen. Bijvoorbeeld om in te loggen, is een geldige gebruikersnaam en wachtwoord vereist. Zoals al eerder beschreven werd een formvalidatie door

gebruikers gezien als een soort sluis welke ongeldige invoer blokkeert. Dit resulteerde in de grafische representatie welke in Figuur 15 te zien is. De check van een formvalidatie vindt plaats nadat de bezoeker naar een volgende stap navigeert (of eigenlijk: in de tijd tussen de eerste stap en voor de tweede stap). Daarom wordt de validator als een afzonderlijk element in het diagram weergegeven, die gemodelleerd kan worden op een positie tussen twee verschillende stappen.

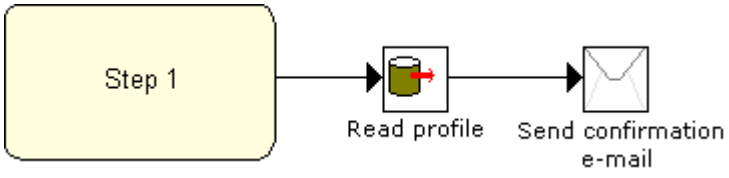


Figuur 15 Na *Step 1* vinden twee formvalidaties plaats

Handler

Een handler zou je kunnen zien als een Operation in UML. Net als formvalidators vindt de uitvoer van een handler plaats, tussen twee stappen. Een validator wordt dan ook als een losse node weergegeven, zie Figuur 16. De grafische notatie van een handler bestaat uit een icoon met daaronder de naam van de handler. De icoon is afhankelijk van het soort handler. Een handler die een e-mail verstuurd zal een afbeelding hebben welke een e-mail representeert. Een handler die gegevens uit een database leest, zal een afbeelding van een database bevatten. Doordat elk soort handler zijn eigen icoon heeft, is het voor een gebruiker van het diagram eenvoudig te herkennen wat er gebeurt. Elke handler heeft een naam, welke gedefinieerd is als een functie op de set van handlers:

$nameH: H \longmapsto String$

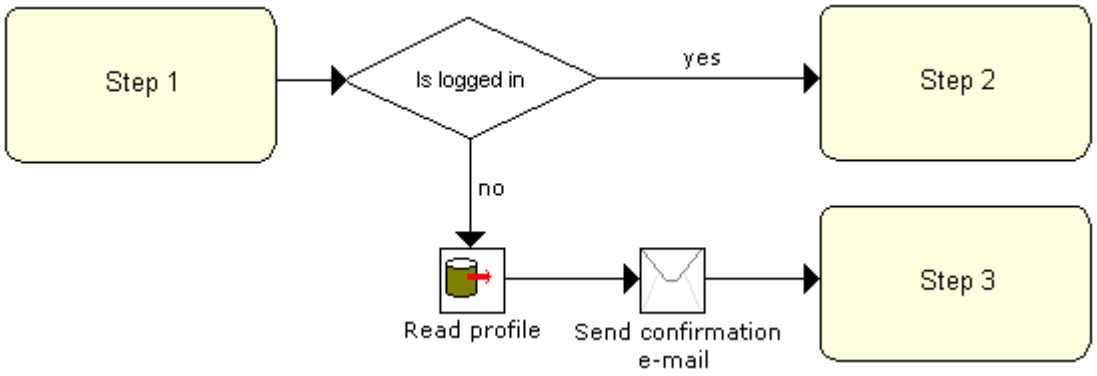


Figuur 16 De grafische weergave van een handler: elke soort handler heeft zijn eigen icoon

Net als de verschillende validatieregels hebben ook handlers verscheidene parameters. Zo heeft de handler *Read profile* de parameter *username* en de validator *Send confirmation mail* de parameters *e-mail*, *sender* en *message*. Deze parameters zijn, net als bij validatieregels en condities ondergebracht in het verderop beschreven gezamenlijke supertype *parameter object*.

Condition (conditie)

Een conditie zou je kunnen zien als een operation (UML) of een gateway (BPM). Een conditie is een keuzemoment welke een splitsing in flow teweeg brengt. Aan de hand van de conditie wordt een bepaalde sequentie van nodes wel of niet uitgevoerd. De grafische weergave is een ruit, met daarin de naam van de conditie, zie het voorbeeld in Figuur 17. In dit voorbeeld wordt met een conditie bepaald of de bezoeker is ingelogd. Zo nee, dan gaat de bezoeker naar stap 2, zo ja dan worden de handlers *Read profile* en *Send confirmation e-mail* uitgevoerd, alvorens stap 3 aan de bezoeker wordt getoond.



Figuur 17 De conditie wordt grafisch weergegeven als een ruit met daarin de naam van de conditie. De conditie brengt een splitsing in de flow teweeg.

De volgende functie geeft aan een conditie een naam:

$nameC: C \longmapsto String$

Ook een conditie heeft verscheidene parameters. Deze parameters zijn, net als bij validatieregels en handlers ondergebracht in het verderop beschreven gezamenlijke supertype *parameter object*. Één van die parameters is de xslconditie. Dit is een xsl-expressie wat de feitelijke conditie uitdrukt:

$xslCondition: C \longmapsto String$

Parameter Object

Een parameter object zou je kunnen vergelijken met een Operation uit UML. Het is een supertype voor handlers, condities en validatieregels, en kent daarom zelf geen grafische notatie. Handlers, condities en validatieregels bevatten parameters, welke zijn ondergebracht in dit supertype. De set van parameter objecten is dan ook als volgt gedefinieerd:

$PARAMOBJ = V \cup H \cup C$

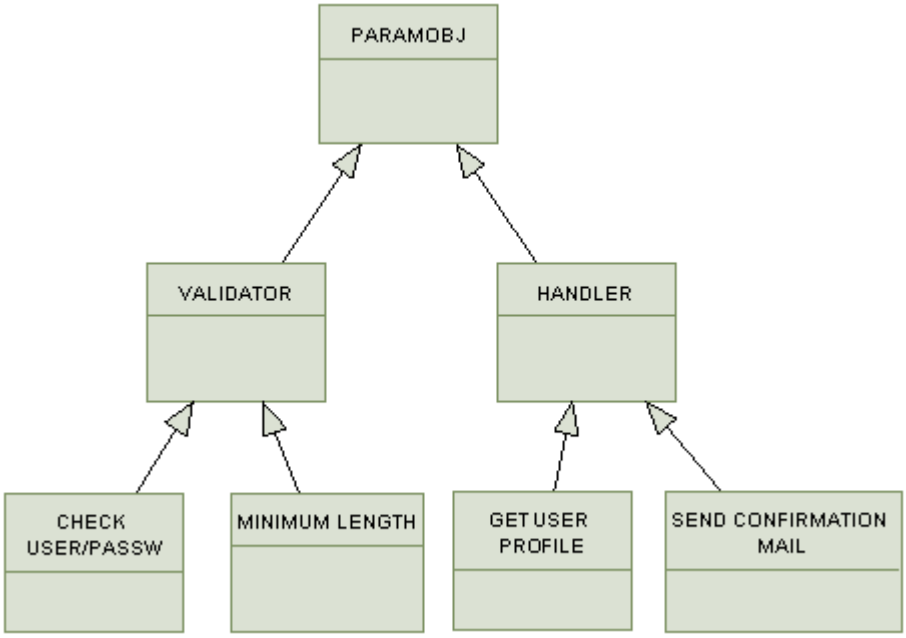
Object Type

Elke validatieregel, handler en conditie is van een bepaald type, met elk zijn eigen parameters. Zo zijn er validatieregels die controleren of de gebruiker zijn juiste inloggegevens heeft ingevuld (*Check username/ password*) of dat de ingevoerde waarde aan een minimum lengte voldoet (*Check minimum length*). Of zo kan er een handler zijn die een gebruikersprofiel uit de database haalt (*get user profile*), of die een registratiemail verstuurt (*send confirmation mail*). Elke validatieregel en handler hebben hun specifieke parameters. In Tabel 13 zijn de verschillende parameters van de zojuist gegeven voorbeelden te zien. Welke parameters er geconfigureerd moeten worden verschilt per parameter object. Elke instantie van een parameter object is van een bepaald type.

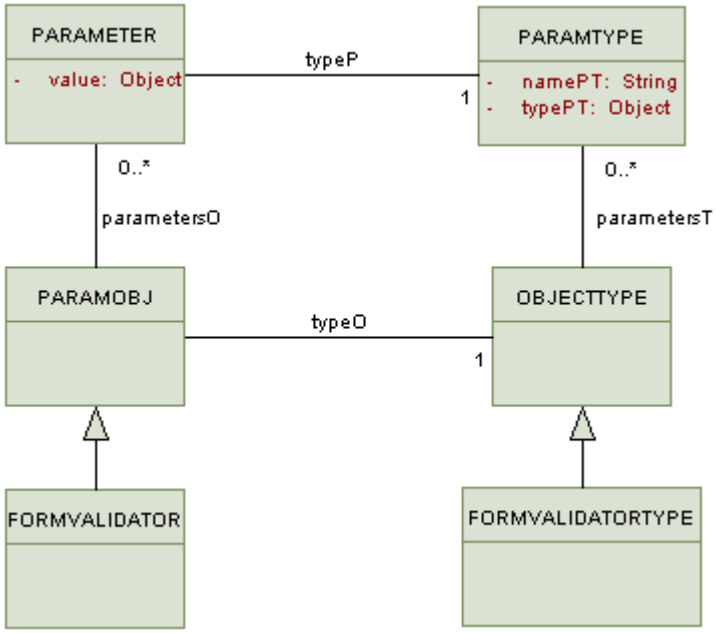
Het is natuurlijk mogelijk om elke validatieregel en handler te definiëren in het metamodel, zoals te zien is in Figuur 18. Voor alle objecten kunnen dan de juiste parameters geconfigureerd worden. Bij het maken van een diagram, wat een populatie van het metamodel is, kan je nu dus een instantie aanmaken van bijvoorbeeld de handler *Get User Profile* met de specifieke invulling van parameters. Het nadeel van alle handlers en validators beschrijven op metaniveau is dat het metamodel gewijzigd moet worden op het moment dat er nieuwe handlers en validators bijkomen. Omdat dit geen wenselijke situatie is, is een meer dynamische benadering van handlers en validatieregels noodzakelijk.

Object Type	Parameter naam	Parameter type	Beschrijving
Check username/ password	Username	Input	De gebruikersnaam wordt aan een invoerveld gekoppeld.
	Password	Input	Het wachtwoord wordt aan een invoerveld gekoppeld.
Check minimum length	Minimum length	Integer	De minimale lengte waaraan de invoer moet voldoen.
Get user profile	Username	Session	De gebruikersnaam wordt aan een sessie-variabele gekoppeld.
Send confirmation mail	E-mail	Input	Het e-mailadres wordt aan een invoerveld gekoppeld.
	Sender name	String	De afzender van de e-mail
	Message	String	Het e-mail bericht.

Tabel 13 Voorbeelden van verschillende validatieregels en handlers



Figuur 18 Een voorbeeld van een metamodel van validatieregels en handlers op een statische wijze gemodelleerd.



Figuur 19 Het metamodel toont de samenhang tussen parameter object, object type en de verschillende parameters. Ter illustratie is ook een verwijzing naar formvalidator in dit figuur opgenomen

Door een generiek object voor handler, conditie en validatieregel in het metamodel te hebben en deze te linken aan een type, kan het probleem worden opgelost. Dit type, het *object type*, beschrijft dan welke specifieke validatieregel of handler er plaats vindt. De verschillende

validatieregels, handlers en bijbehorende parameters worden dan gespecificeerd als populatie van het metamodel. Dit is op metaniveau weergegeven in Figuur 19. De formvalidator is van een bepaald type. De populatie van *FORMVALIDATORTYPE* geeft aan welke dit kan zijn. Daarnaast is van elk object type (dus ook van de *FORMVALIDATORTYPE*) beschreven welke parameters ze kunnen hebben. Tevens is van elk parameter object (in dit voorbeeld *FORMVALIDATOR*) beschreven welke waardes aan de verschillende parameters hangen. Het *OBJECTTYPE* is als volgt gedefinieerd:

$$OBJECTTYPE = FIELDVALIDATORTYPE \cup FORMVALIDATORTYPE \cup HANDLERTYPE \cup CONDITIONTYPE$$

De functie *type* kent aan een parameter object het type toe:

$$typeO: PARAMOBJ \mapsto OBJECTTYPE$$

Een voorbeeld van een parameter object en een object type is het volgende: stel we hebben een formvalidator *v* van het type *Check username/ password*. Deze is dan als volgt gedefinieerd:

$$v \in PARAMOBJ$$
$$typeO(v) = t$$
$$nameT(t) = "username/\ password"$$

Parameter type

Parameter type beschrijft hoe een parameter van een object type eruit ziet. Ook dit komt visueel niet terug in het diagram, maar wordt gebruikt bij de specificatie van parameters van bijvoorbeeld een validatieregel of een handler. De set *PARAMTYPE* bevat alle parameter types. Over deze set zijn de volgende functies gedefinieerd:

$$namePT: PARAMTYPE \longmapsto String$$

Deze functie kent een naam toe aan de parameter. De volgende functie geeft het type van een parameter. Het type kan zowel een primitief datatype zijn (zoals integer, float, boolean, string), maar ook een verwijzing naar een invoerveld, stap of pagina.

$$typePT: PARAMTYPE \longmapsto Object$$

Elk objecttype kan verschillende parameters hebben. Dit komt tot uitdrukking middels de volgende functie:

$$parametersT: OBJECTTYPE \mapsto \wp(PARAMTYPE)$$

Hierbij heeft een object type een set van parameters. Het is van belang dat binnen een object type alle parameters een unieke naam hebben. Dit komt tot uitdrukking in Axioma 4.

$$\forall o \in OBJECTTYPE\ \forall p_1, p_2 \in parametersT(o)[namePT(p_1) = namePT(p_2) \Rightarrow p_1 = p_2]$$

Axioma 4 Alle parameters met binnen een object hebben een unieke naam

De formvalidator *v* van het type *Check username/ password* zou dan de parameters als volgt gedefinieerd hebben:

$namePT(p_1) = "Username"$
$typePT(p_1) = Textfield$
$namePT(p_2) = "Password"$
$typePT(p_2) = Textfield$
$parametersT\big(typeO(v)\big) = \{p_1, p_2\}$

De parameters *username* en *password* verwijzen beide naar een invoerveld.

Parameter

Een parameter beschrijft de specifieke waarde van een parameter type. De in Tabel 13 beschreven validatieregel *Check username/password* heeft bijvoorbeeld een parameter username. De waarde van deze parameter is een verwijzing naar het invoerveld *username*. De set van alle parameters is gedefinieerd als *PARAMETER*. Deze bevat een waarde en een parameter type, wat tot uitdrukking komt in de volgende functies:

$valueP: PARAMETER \longmapsto Object$

$typeP: PARAMETER \longmapsto PARAMTYPE$

Omdat elk object alleen parameters mag bevatten die gedefinieerd zijn binnen het object type, gelden de volgende axioma:

- Alle parameters binnen het object zijn ook gedefinieerd binnen het object type, zie Axioma 5.
- Alle parameters binnen het object type zijn ook gedefinieerd binnen het object, zie Axioma 6.
- Elk parametertype komt binnen het object maar één keer voor, zie Axioma 7.

$\forall o \in PARAMOBJ\ \forall p \in parametersO(o) \exists t_q \in parametersT(typeO(o))\big[typeP(p) = t_q\big]$

Axioma 5 Parameters binnen het parameter object zijn ook gedefinieerd binnen het object type.

$\forall o \in PARAMOBJ\ \forall t_q \in parametersT(typeO(o)) \exists p \in parametersO(o)\big[typeP(p) = t_q\big]$

Axioma 6 Parameters binnen het object type zijn ook gedefinieerd binnen het parameter object.

$\forall o \in PARAMOBJ\ \forall p_1, p_2 \in parametersO(o)[typeP(p_1) = typeP(p_2) \Rightarrow p_1 = p_2]$

Axioma 7 Elk parameter type komt maar één keer voor binnen een parameter object.

De axioma's Axioma 4, Axioma 5, Axioma 6, en Axioma 7 welke hierboven zijn beschreven zijn ook in het metamodel van Figuur 19 uit te drukken in termen van OCL (Object Constraint Language, (OMG, Object Constraint Language - OMG Available Specification - Version 2.0 2001), (Warmer and Kleppe 1999)).

context OBJECTTYPE
inv: parametersT->isUnique(namePT)

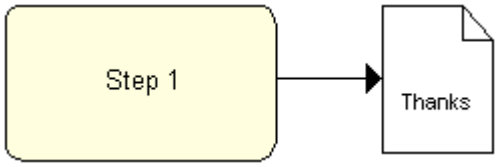
context PARAMOBJ
inv: parametersO->forAll(p | typeO.parametersT->exists(qt | p.typeP = qt))

context PARAMOBJ
inv: typeO.parametersT->forAll(qt | parametersO->exists(p | p.typeP = qt))

context PARAMOBJ
inv: typeO.parametersT->forAll(p1, p2 | p1.typeP = p2.typeP implies p1 = p2)

Page (pagina)
Een pagina is te vergelijken met een Class (UML) of Activity (BPM). De definitie van een pagina hoort buiten de definitie van het formulier. Een formulier kan wel een verwijzing hebben naar een andere pagina, bijvoorbeeld aan het einde van een flow (een *bedankt*-pagina) of een foutpagina. Het grafische symbool voor pagina is weergegeven in Figuur 20 en wordt door veel tools en applicaties gebruikt als iconische weergave. Doordat de pagina slechts als verwijzing wordt gebruikt, en niet verder geconfigureerd kan worden, bevat het alleen de eigenschap *naam* wat in de volgende functie tot uiting komt:

$nameP: P \longmapsto String$



Figuur 20 De grafische weergave van een pagina.

Figuur 21 De grafische weergave van een blok: een beschrijving in natuurlijke taal van wat er moet gebeuren.

Block (blok)

Een blok komt binnen het domeinmodel, dus ook binnen het Content Management Systeem, niet voor. Echter, uit gesprekken met toekomstige gebruikers van de diagrammen blijkt dit een nuttige toevoeging te zijn. Het komt voort uit de Business Process Diagrammen van WebML. Het wordt gebruikt als een tijdelijke node, waarin met natuurlijke taal wordt beschreven wat er moet gebeuren. De grafische notatie hiervan is weergegeven in Figuur 21.

Het blok kent zijn nut tijdens de verschillende stadia van het ontwikkelproces. Bij aanvang van het proces, gedurende het overleg met de klant, waarbij de functionele requirements moeten worden afgeleid, zijn handlers en validators nog niet van belang. Een plaatje van de verschillende stappen van een formulier en de bijbehorende samenhang kan de communicatie verbeteren. Door in natuurlijke taal aan te geven wat er tussen verschillende stappen gebeurd, kan je dit toch in je diagram meenemen, zonder de klant te confronteren met implementatie details, zoals routers en validators. In een later stadium kan het diagram dan verder geformaliseerd worden door invulling te geven aan de verschillende blokken. De functie *description* geeft een blok zijn beschrijving:

$$descriptionB: B \longmapsto String$$

Node

Een node is te vergelijken met een Interaction (UID), Class (UML) of Activity/Gateway (BPM). Een node is een supertype voor al de grafische symbolen, waaruit de volgende definitie van een node volgt:

$$N = S \cup FORMVALIDATOR \cup H \cup C \cup P \cup B$$

Edge

De verschillende grafische symbolen, welke allen zijn gedefinieerd als node, zijn met elkaar verbonden door middel van gerichte edges. Een edge is te vergelijken met een Directed Association (UML) en een Sequence Flow (BPM). Een edge kent zowel een begin (*source*) en een eind *target*), wat gedefinieerd is middels de volgende functies:

$$sourceE: E \longmapsto (N - P)$$

$$targetE: E \longmapsto N$$

Het begin van een edge kan nooit een pagina zijn, omdat de implementatie van pagina's niet in het formulier diagram valt. Een conditie heeft minstens twee of meer uitgaande edges. Er vindt immers altijd een splitsing van de flow plaats na een conditie, zie Axioma 8. Hierbij is *sourceE*$^{-1}$ de inverse van *sourceE*:

$$\forall c \in C\ [|sourceE^{-1}(c)| \geq 2]$$

Axioma 8 Een conditie heeft altijd twee of meer uitgaande edges.

Alle overige nodes hebben maximaal één uitgaande edge. Hier kan immers geen splitsing van de flow plaatsvinden. Een stap kan het eindpunt zijn van een flow en heeft daarom niet altijd een uitgaande edge, zie Axioma 9. De overige nodes (formvalidators, handlers en blocks) kunnen nooit het eindstation zijn (want er moet altijd een pagina of een stap worden getoond aan de bezoeker) en hebben daarom altijd één uitgaande edge, zie Axioma 10.

$$\forall s \in S\ [|sourceE^{-1}(n)| \leq 1]$$

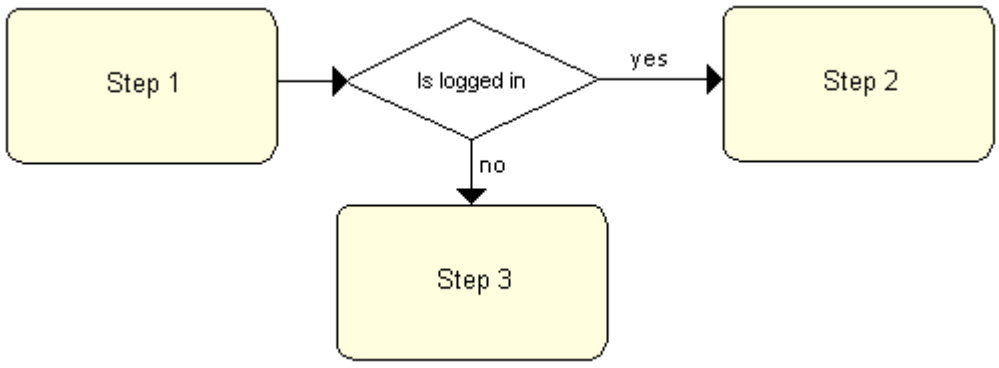
Axioma 9 Een stap heeft optioneel een uitgaande pijl, afhankelijk of het wel of niet het einde van de flow is.

$$\forall n \in FORMVALIDATOR \cup H \cup B\ [|sourceE^{-1}(n)| = 1]$$

Axioma 10 Een formvalidator, handler en block hebben precies één uitgaande edge.

Daarnaast hebben edges die van een conditie af komen een *case*. De case geeft aan of de edge wel of niet moet worden gevolgd, afhankelijk van de uitkomst van de conditie. Een case is, middels een partionele functie, als volgt gedefinieerd:

$$caseE: E \longmapsto \{T, F\}$$



Figuur 22 Een edge kan worden voorzien van een case.

Een case kan alleen voorkomen, direct na een conditie-node. Daarom geldt ook Axioma 11.

$$\forall e \in E\ [(caseE(e) = T \vee caseE(e) = F) \Longrightarrow sourceE(e) \in C\ ]$$

Axioma 11 Een case kan alleen voorkomen, bij een edge die als bron een conditie-node heeft.

6.3 Voorbeeld populatie

Zojuist hebben we de concrete syntax en verschillende eigenschappen daarvan bekeken. Om de verschillende definities wat te verhelderen volgt hier een voorbeeld populatie van het beschreven model. De grafische notatie van deze populatie is afgebeeld in Figuur 23.

Het diagram toont niet de volledige concrete syntax, omdat bijvoorbeeld verschillende parameters die bij een validatieregel of handler gedefinieerd zijn niet zijn weergegeven. De populatie is hier als volgt gedefinieerd:

$S = \{s_r, s_l, s_c, s_u\}$
$V = \{fi_1, fi_2, fi_3, fi_4, fi_5, fo_1\}$
$H = \{h_a, h_s, h_r\}$
$C = \{c_1\}$
$P = \emptyset$
$B = \emptyset$
$N = \{s_r, s_l, s_c, s_u, fo_1, h_a, h_s, h_r, c_1\}$
$E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$
$TEXTINPUT = \{ti_1, ti_2, ti_3, ti_4, ti_5, ti_6\}$
$FE = INPUT = TEXTINPUT$

$FORMVALIDATOR = \{fo_1\}$
$FIELDVALIDATOR = \{fi_1, fi_2, fi_3, fi_4, fi_5\}$
$PARAMOBJ = \{fi_1, fi_2, fi_3, fi_4, fo_1, h_a, h_s, h_r, c_1\}$
$FIELDVALIDATORTYPE = \{fit_l, fit_e, fit_f\}$
$FORMVALIDATORTYPE = \{fot_1\}$
$HANDLERTYPE = \{ht_a, ht_s, ht_r\}$
$CONDITIONTYPE = \{ct_1\}$
$OBJECTTYPE = \{fit_l, fit_e, fit_f, fot_1\ ht_a, ht_s, ht_r, ct_1\}$
$PARAMTYPE = \{pt_1, pt_2, pt_3, pt_4, pt_5, pt_6, pt_7, pt_8, pt_9, pt_{10}, pt_{11}\}$
$PARAMETER = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}\}$

Steps (Stappen)

Er zijn vier verschillende stappen, waarbij hun naam als volgt is gedefinieerd:

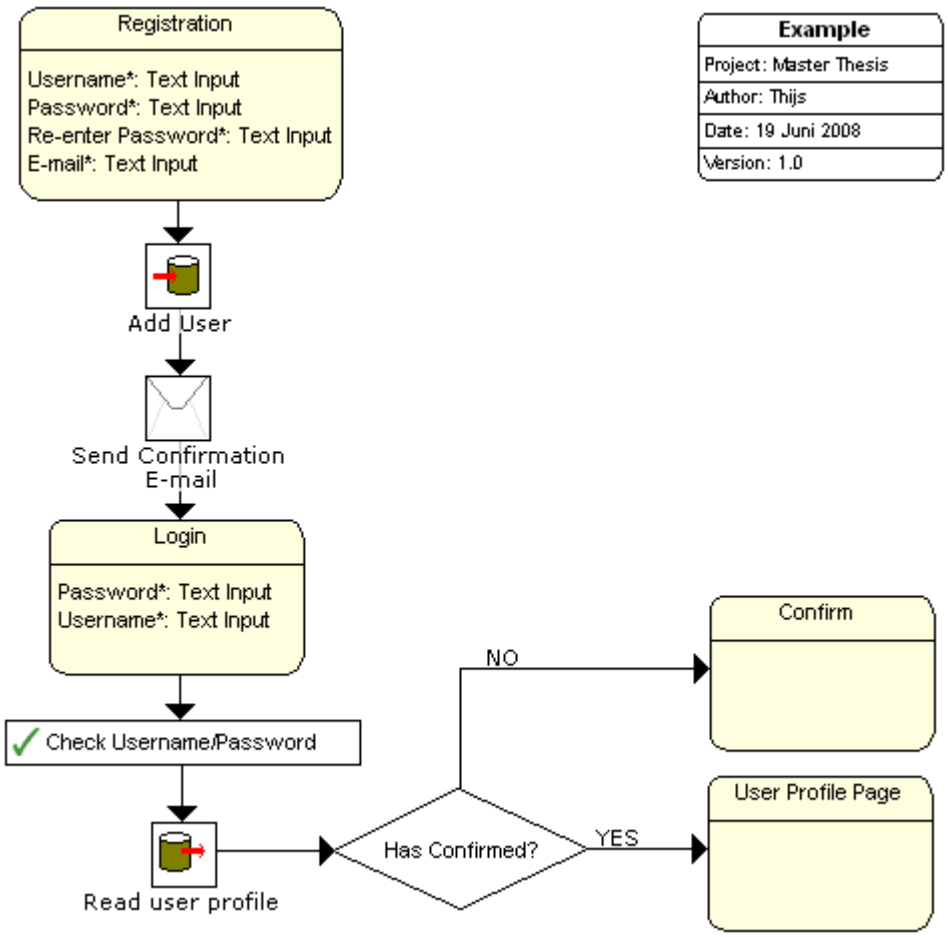
$nameS(s_r) = "Registration"$
$nameS(s_l) = "Login"$
$nameS(s_c) = "Confirm"$
$nameS(s_u) = "User\ Profile"$

Form Elements (formulier elementen)

Dit voorbeeld bevat alleen tekstinvoervelden en geen andere formulier elementen. De invoervelden zijn als volgt gedefinieerd:

$TEXTINPUT = \{ti_1, ti_2, ti_3, ti_4, ti_5, ti_6\}$

$FE = INPUT = TEXTINPUT$



Figuur 23 Een voorbeeld diagram, gebruikt in een registratieproces

$nameI(ti_1) = "Username"$
$required(ti_1) = T$
$default(ti_1) = ""$
$password(ti_1) = F$

$nameI(ti_2) = "Password"$
$required(ti_2) = T$
$default(ti_2) = ""$
$password(ti_2) = T$

$nameI(ti_3) = "Re - enter\ Password"$
$required(ti_3) = T$
$default(ti_3) = ""$
$password(ti_3) = T$

$nameI(ti_4) = "E - mail"$
$required(ti_4) = T$
$default(ti_4) = ""$
$password(ti_4) = F$

$nameI(ti_5) = "Username"$
$required(ti_5) = T$
$default(ti_5) = ""$
$password(ti_5) = F$

$nameI(ti_6) = "Password"$
$required(ti_6) = T$
$default(ti_6) = ""$
$password(ti_6) = T$

De stappen *Confirm* en *User Profile* bevatten geen velden. De verschillende velden zijn als volgt verdeeld over de andere twee stappen:

$fields(s_r) =< ti_1, ti_2, ti_3, ti_4 >$
$fields(s_l) =< ti_5, ti_6 >$
$fields(s_c) = \emptyset$
$fields(s_u) = \emptyset$

Field validators (veld validaties)

Er zijn verschillende veldvalidaties. Zo moet de gebruikersnaam een minimaal aantal karakters bevatten, moeten de ingevulde wachtwoorden gelijk zijn en moet het e-mail adres aan een bepaald invoerformaat voldoen. Om dit te bewerkstelligen definiëren we eerst de verschillende typen veldvalidaties (als *FIELDVALIDATORTYPE*) en geven we aan welke parameters deze veldvalidaties dienen te hebben. Vervolgens wordt een instantie van een type validator gekoppeld aan een invoerveld, met de daarbij behorende parameters. De verschillende typen veldvalidaties en de daarbij behorende parameters zijn als volgt gedefinieerd:

$FIELDVALIDATORTYPE = \{fit_l, fit_e, fit_f\}$

$nameT(fit_l) = "Minimum\ Length"$
$parametersT(fit_l) = \{pt_1\}$
$namePT(pt_1) = "Length"$
$typePT(pt_1) = Integer$
$nameT(fit_e) = "Equal"$
$parametersT(fit_e) = \{pt_2\}$
$namePT(pt_2) = "Other\ input"$
$typePT(pt_2) = Textfield$

$nameT(fit_f) = "Format\ e-mail"$
$parametersT(fit_f) = \emptyset$

Het type veldvalidatie *Minimum Length* controleert of een invoerveld aan de opgegeven lengte voldoet. Het type *Equal* controleert of de waarde in een ander invoerveld gelijk is aan de hier ingevulde waarde. Dit kan bijvoorbeeld gebruikt worden om te controleren of het herhaalde wachtwoord correct is ingevuld. De zojuist gedefinieerde typen kunnen nu worden gebruikt bij de definitie van de veldvalidaties. Deze validatieregels zijn niet zichtbaar in het diagram, maar zijn wel aanwezig in het concrete model.

$FIELDVALIDATOR = \{fi_1, fi_2, fi_3, fi_4, fi_5\}$

$validationF(ti_1) = \{fi_1\}$
$typeO(fi_1) = fit_l$
$parametersO(fi_1) = \{p_1\}$
$typeP(p_1) = pt_1$
$valueP(p_1) = 4$

In dit voorbeeld wordt aan het field *Username* de validator van het type *Minimum Length* gekoppeld, waarbij de minimale lengte 4 karakters moet zijn. De overige veldvalidaties zijn als volgt gedefinieerd:

$validationF(ti_2) = \{fi_2\}$
$typeO(fi_2) = fit_l$
$parametersO(fi_2) = \{p_2\}$
$typeP(p_2) = pt_1$
$valueP(p_2) = 6$

Dit beschrijft dat de minimale lengte van de invoer in het *Password* veld 6 karakters moet zijn.

$validationF(ti_3) = \{fi_3, fi_4\}$
$typeO(fi_3) = fit_l$
$parametersO(fi_3) = \{p_3\}$
$typeP(p_3) = pt_1$
$valueP(p_3) = 6$
$typeO(fi_4) = fit_e$
$parametersO(fi_4) = \{p_4\}$
$typeP(p_4) = pt_2$
$valueP(p_4) = ti_2$

Ook het *Re-enter Password* veld dient minimaal 6 karakters als invoer te hebben. Daarnaast dient deze waarde gelijk te zijn als de waarde in het *Password* veld[1].

$validationF(ti_4) = \{fi_5\}$
$typeO(fi_5) = fit_f$
$parametersO(fi_5) = \emptyset$

De waarde in het *E-mail* veld wordt gecontroleerd door een validator van het type *Format e-mail*. Hierbij behoren geen parameters.

[1] In principe hoeft hier niet nogmaals gecontroleerd te worden of de invoer 6 karakters lang is. Deze validatie is hier slechts nogmaals aangegeven voor de duidelijkheid.

Form Validators (form validaties)

Naast de zonet beschreven veldvalidaties bevat het voorbeelddiagram ook een formvalidatie. Deze controleert of de ingevulde gebruikersnaam en wachtwoord door de bezoeker goed zijn ingevuld. We definiëren eerst het validator type, alvorens de implementatie hiervan wordt gedefinieerd.

$FORMVALIDATORTYPE = \{fot_1\}$

$nameT(fot_1) = "Check\ Username/Password"$
$parametersT(fot_1) = \{pt_3, pt_4\}$
$namePT(pt_3) = "Username"$
$typePT(pt_3) = Textfield$
$namePT(pt_4) = "Password"$
$typePT(pt_4) = Textfield$

Dit type heeft 2 parameters. Zowel parametertype *Username* als parametertype *Password* moet worden gekoppeld aan een tekstveld. Bij de formvalidator worden deze gelinkt aan een tekstveld. Dit is als volgt gedefinieerd:

$FORMVALIDATOR = \{fo_1\}$

$typeO(fo_1) = fot_1$
$parametersO(fo_1) = \{p_5, p_6\}$
$typeP(p_5) = pt_3$
$valueP(p_5) = ti_5$
$typeP(p_6) = pt_4$
$valueP(p_6) = ti_6$

Handlers

De drie handlers die in het voorbeeld worden gebruikt, lijken qua definitie veel op de validatieregels. Ook de handlers bevatten namelijk parameters. Net als bij de validatieregels definiëren we eerst de handlerstypes en vervolgens de verschillende handlers.

$HANDLERTYPE = \{ht_a, ht_s, ht_r\}$

$nameT(ht_a) = "Add\ User"$
$parametersT(ht_a) = \{pt_5, pt_6, pt_7\}$
$namePT(pt_5) = "Username"$
$typePT(pt_5) = Textfield$
$namePT(pt_6) = "Password"$
$typePT(pt_6) = Textfield$
$namePT(pt_7) = "E-mail"$
$typePT(pt_7) = Textfield$

$nameT(ht_r) = "Read\ User\ Profile"$
$parametersT(ht_s) = \{pt_{11}, pt_{12}, pt_{13}\}$
$namePT(pt_{11}) = "E-mail"$
$typePT(pt_{11}) = Textfield$
$namePT(pt_{12}) = "Sender\ name"$
$typePT(pt_{12}) = String$
$namePT(pt_{13}) = "Message"$
$typePT(pt_{13}) = String$

$nameT(ht_s) = "Send\ Confirmation\ E-mail"$
$parametersT(ht_s) = \{pt_8, pt_9, pt_{10}\}$
$namePT(pt_8) = "E-mail"$
$typePT(pt_8) = Textfield$
$namePT(pt_9) = "Sender\ name"$
$typePT(pt_9) = String$
$namePT(pt_{10}) = "Message"$
$typePT(pt_{10}) = String$

De definitie van deze types wordt vervolgens gebruikt bij de definitie van de handlers:

$H = \{h_a, h_s, h_r\}$

$typeO(h_a) = ht_a$
$parametersO(h_a) = \{p_7, p_8, p_9\}$
$typeP(p_7) = pt_{11}$
$valueP(p_7) = ti_1$
$typeP(p_8) = pt_{12}$
$valueP(p_8) = ti_2$
$typeP(p_9) = pt_{13}$
$valueP(p_9) = ti_4$

Conditions (Condities)

De conditie wordt uitgedrukt middels een naam (die in het diagram is te zien) en een parameter, die de achterliggende xsl-conditie beschrijft. In dit geval is er slechts één conditie die als volgt wordt uitgedrukt:

$C = \{c_1\}$
$nameC(c_1) = "Has\ Confirmed?"$
$xslCondition(c_1) = "//fields/condirmed = true"$

Edges

De verschillende edges verbinden de nodes met elkaar. De aanwezige nodes en edges zijn als volgt gedefinieerd:

$N = \{s_r, s_l, s_c, s_u, fo_1, h_a, h_s, h_r, c_1\}$

$E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$

$sourcE(e_1) = s_r$
$targetE(e_1) = h_a$

$sourcE(e_2) = h_a$
$targetE(e_2) = h_s$

$sourcE(e_3) = h_s$
$targetE(e_3) = s_l$

$sourcE(e_4) = s_l$
$targetE(e_4) = fo_1$

$sourcE(e_5) = fo_1$
$targetE(e_5) = h_r$

$sourcE(e_6) = h_r$
$targetE(e_6) = c_1$

$sourcE(e_7) = c_1$
$targetE(e_7) = s_c$
$caseE(e_7) = \text{F}$

$sourcE(e_8) = c_1$
$targetE(e_8) = s_u$
$caseE(e_8) = \text{T}$

6.4 Abstracte syntax

De zojuist beschreven concrete syntax van het WebForm diagram sluit aan bij het mental model van de gebruiker. De abstracte syntax sluit aan bij het implementatie model van het systeem. Omdat het implementatie model niet aansluit op het mental model, is er het onderscheid gemaakt tussen de concrete en abstracte syntax.

De abstracte syntax van het WebForm Diagram lijkt veel op de concrete syntax. De meeste objecten uit de concrete syntax, zoals stap, validatieregel, handler, parameter en pagina, komen ook terug in de abstracte syntax. Formvalidaties en handlers, worden echter wel gekoppeld aan een stap. Deze worden dan in een bepaalde volgorde uitgevoerd, afhankelijk in welke volgorde ze verbonden zijn door edges. Edges worden in de abstracte syntax vertaald naar twee verschillende concepten: ofwel naar een sequentieel nummer, ofwel naar een router. Het sequentieel nummer geeft aan in welke volgorde de formvalidaties, handlers en routers moeten worden uitgevoerd. Een edge die leidt naar een stap of pagina, wordt vertaald in een router, die de bezoeker leidt naar die bewuste stap of pagina. Hoe deze router er uit komt te zien is ook afhankelijk van Conditie objecten uit het concrete model. Deze bepalen conditionele routers, bijvoorbeeld door iemand naar een andere pagina te leiden als hij niet is ingelogd. Condities komen niet meer als objecten terug in de abstracte syntax, maar zijn eigenschappen, een preconditie, van handlers en routers. Blokken komen in het geheel niet terug in de abstracte syntax, deze zijn puur bedoeld voor het (iteratieve) ontwerp van formulieren, en zijn geen onderdeel van het implementatie model.

Hieronder wordt kort een opsomming gegeven van de definities welke overeen komen met de concrete syntax. Daarna wordt er uitgebreid ingegaan op de onderdelen die specifiek zijn binnen

de abstracte syntax. Hoe de concrete syntax wordt vertaald naar de abstracte syntax, wordt beschreven in sectie 6.5.

Het WebForm diagram is een septupel $f' = \langle N', S', FE', V', H', R', P' \rangle$. Hierbij is *N'* de set van Nodes, *S'* de verzameling stappen, *FE'* de verzameling van formulier elementen, *V'* de verzameling validators, *H'* de verzameling handlers, *R'* de set van routers en *P'* de verzameling van pagina's.

Step (stap)
De definitie van een stap is, naast een uitbreiding, hetzelfde als de definitie in de concrete syntax:

$nameS' : S' \longmapsto String$
$fields' : S' \longmapsto \mathcal{S}(FE')$

Daarnaast bevat een stap formvalidaties, handlers en routers, wat wordt uitgedrukt middels de volgende functies:

$validatorsS' : S' \longmapsto \wp(FORMVALIDATOR')$
$handlersS' : S' \longmapsto \wp(H')$
$routersS' : S' \longmapsto \wp(R')$

Elke formvalidatie, handler en router kan maar aan één stap zijn toegekend. Dit is uitgedrukt in de volgende axioma's:

$$\forall v \in FORMVALIDATOR' \; \forall s_1, s_2 \in S'[v \in validatorsS'(s_1) \wedge \; v \in validatorsS'(s_2) \Rightarrow s_1 = s_2]$$

Axioma 12 Een formvalidatie kan slechts binnen één stap voorkomen.

$$\forall h \in H' \; \forall s_1, s_2 \in S'[h \in handlersS'(s_1) \wedge h \in handlersS'(s_2) \Rightarrow s_1 = s_2]$$

Axioma 13 Een handler kan slechts binnen één stap worden gebruikt.

$$\forall r \in R' \; \forall s_1, s_2 \in S'[r \in routersS'(s_1) \wedge r \in routersS'(s_2) \Rightarrow s_1 = s_2]$$

Axioma 14 Een router kan slechts aan één stap worden toegekend.

Form Element (invoerveld)

De definitie van de verschillende formulier elementen is bijna hetzelfde als in het concrete model. Echter, de eigenschap *verplicht*, welke aangeeft of een invoerveld verplicht is, is in het abstracte model geïmplementeerd als een veldvalidatieregel. Hieronder zijn de verschillende definities nog eens opgesomd:

$FE' = INPUT' \cup BUTTON' \cup INFO'$
$INPUT' = TEXTINPUT' \cup UPLOAD' \cup LIST'$
$nameI' : INPUT' \rightarrowtail String$
$default' : TEXTINPUT' \rightarrowtail String \cup \{""\}$
$password' : TEXTINPUT' \rightarrowtail \{T, F\}$
$validationF' = INPUT' \rightarrowtail \wp(FIELDVALIDATOR')$

De eigenschap *verplicht* wordt geïmplementeerd als een populatie van het abstracte model:

$fi \in validationF'(x)$
$fi \in FIELDVALIDATOR'$
$typeO'(fi) = fit$
$nameT'(fit) = "Required"$

Validator (validatieregel)

Ook validators zijn overgenomen van het concrete model. Om de volgorde aan te geven, waarin de validatieregels (in combinatie met handlers en routers) moeten worden uitgevoerd is er ook een eigenschap *ordernr* aanwezig:

$V' = FIELDVALIDATOR' \cup FORMVALIDATOR'$
$ordernr' : (FORMVALIDATOR' \cup H' \cup R') \rightarrowtail Integer$

Het ordernr geeft aan in welke volgorde de validatieregels, handlers en routers moeten worden uitgevoerd. Omdat alles sequentieel moet worden uitgevoerd, is het hierbij van belang dat alle ordernummers uniek zijn, zoals uitgedrukt in Axioma 15.

$\forall s \in S' \forall x_1, x_2 \in (validatorS'(s) \cup \text{handlerS}'(\text{s}) \cup \text{routersS}'(\text{s}) [ordernr'(x_1) \wedge ordernr'(x_2) \Rightarrow x_1 = x_2]$

Axioma 15 Elk ordernr is uniek.

Daarnaast kan een validatieregel, handler of router een preconditie hebben. Dit is de vertaling van een conditie-element uit de concrete syntax naar de abstracte syntax. Deze preconditie is een partionele functie, hij is optioneel, en is als volgt gedefinieerd:

$precondition' : (FORMVALIDATOR' \cup H' \cup R') \rightarrowtail String$

De preconditie is dan uitgedrukt als een xsl-expressie.

Handler

Naast het hierboven beschreven *ordernr* heeft een handler ook een preconditie. Deze is gebaseerd op een conditie-element:

$precondition' : H \mapsto String$

Page (pagina)

Ook de definitie van een pagina is overeenkomstig het concrete model:

$nameP' : P' \mapsto String$

Router

De router komt niet voor in de concrete syntax. Het is een combinatie van eventuele condities en een edge welke is verbonden aan een stap of een pagina. Hoe deze zaken worden omgezet in een router, staat beschreven in paragraaf 6.5. Een router is grotendeels hetzelfde opgebouwd als een handler: het bevat een naam en een ordernr en verschillende parameters. Ook de stap/ pagina waar naar verwezen wordt is een parameter. Doordat ook router een subset is van PARAMOBJ worden de parameters en het type router hier niet verder gedefinieerd:

$R' \subseteq PARAMOBJ'$

Object type en Parameters

Ook de definities van de object typen, parameters en parametertypen blijven ongewijzigd ten aanzien van het concrete model. Voor de volledigheid zijn de definities hieronder nogmaals weergegeven. In plaats van condities bestaat een objecttype uit routers:

$$PARAMOBJ' = V' \cup H' \cup R'$$
$$OBJECTTYPE' = FIELDVALIDATORTYPE' \cup FORMVALIDATORTYPE' \cup HANDLERTYPE' \cup ROUTERTYPE'$$
$$typeO' : PARAMOBJ' \mapsto OBJECTTYPE'$$
$$namePT' : PARAMTYPE' \mapsto String$$
$$typePT' : PARAMTYPE' \mapsto Object$$
$$parametersT' : OBJECTTYPE' \mapsto \wp(PARAMTYPE')$$
$$valueP' : PARAMETER' \mapsto Object$$
$$typeP' : PARAMETER' \mapsto PARAMTYPE'$$

6.5 Mapping Concrete syntax en Abstracte syntax

De concrete syntax en de abstracte syntax zijn grotendeels één op één te mappen. De verschillen tussen beide zijn zojuist beschreven. De transformatie van de concrete syntax naar de abstracte syntax wordt in dit deel beschreven.

Het attribuut 'Verplicht'

De eigenschap *verplicht* van een invoerveld, komt niet voor in de abstracte syntax. Deze is hier geïmplementeerd als een validator. Dit wordt als volgt getransformeerd:

Concrete syntax:
$\forall x \in INPUT[required(x) = T]$

Abstracte syntax:
$fi \in validationF'(x)$
$fi \in FIELDVALIDATOR'$
$typeO'(fi) = fit$
$nameT'(fit) = "Required"$

Routers, precondities en koppeling van formvalidators, handlers en routers aan een stap

Binnen de abstracte syntax komen er geen edges voor, maar zijn formvalidators en handlers gekoppeld aan een stap. Hierbij wordt een bepaalde volgorde in acht genomen. Daarnaast worden condities vertaald in precondities welke worden gekoppeld aan handlers en in routers. Al deze zaken worden met één algoritme vertaald naar de abstracte syntax

translate()
$for(s \in Step)$
$for(e \in \{x|sourceE(x) = s\})$
addChilds(s, e, o, "") //no precondition yet -> empty condition

addChild(s, e, index, cond): integer
$n = targetE(e);$

$if(n \in C)$*{ //target node is a condition*
$condT = cond \wedge xslCondition(n)$
$for(e_2 \in \{x|sourceE(n) = s \wedge caseE(n) = T\})$*//handle all True-cases*
index = addChild(s, e_2, index, condT) //recursively handle True-case
$for(e_3 \in \{x|sourceE(n) = s \wedge caseE(n) \neq T\})$*//the rests are False-cases*
index = addChild(s, e_3, index, cond) //recursively handle False-case
$}$ *else* $if(n \in S$ *or* $n \in P)$*{ //target node is a step or a page*
// create a router to navigate to the step/ page
$r \in R'$
$r \in routersS'(s)$
$precondition'(r) = cond$

```
        targetR'(r) = n
        ordernr'(n) = index
        index++
        typeO'(r) = r_t
        nameT'(r_t) = createName(n, cond)
    } else{ //target node is nor a condition nor a step nor a page
        ordernr'(n) = index
        index++
        precondition'(n) = cond
        if(n ∈ FORMVALIDATOR)
            n ∈ validatorsS'(s)
        if(n ∈ H)
            n ∈ handlersS'(s)
        for(e_2 ∈ {x|sourceE(n) = s})
            index = addChild(s, e_2, index, cond) //continue with the child
    }
    return index
```

De graaf wordt doorlopen door de verschillende edges te volgen. Per edge wordt de functie *addChild* aangeroepen. De verschillende nodes worden hier recursief aan een stap toegevoegd. Op het moment dat er een conditie-node wordt verwerkt, wordt de preconditie aangepast en recursief aan de *true-cases* toegevoegd. Indien een edge naar een stap of een pagina verwijst, moet de bezoeker hierheen worden verwezen. Dit gebeurt middels een router, die daar gedefinieerd wordt.

De functie *createName(node, condition)* zoekt aan de hand van het type eind-node en of er wel of geen conditie is de juiste naam voor de router:

```
createName(n, cond): String
    if(n ∈ S ∧ cond = "") // a step and no precondition
        return "GoToStep"
    if(n ∈ S ∧ cond ≠ "") // a step and a precondition
        return "GoToStepCondition"
    if(n ∈ P ∧ cond = "") // a page and no precondition
        return "GoToPage"
    if(n ∈ P ∧ cond ≠ "") // a page and a precondition
        return "GoToPageCondition"
```

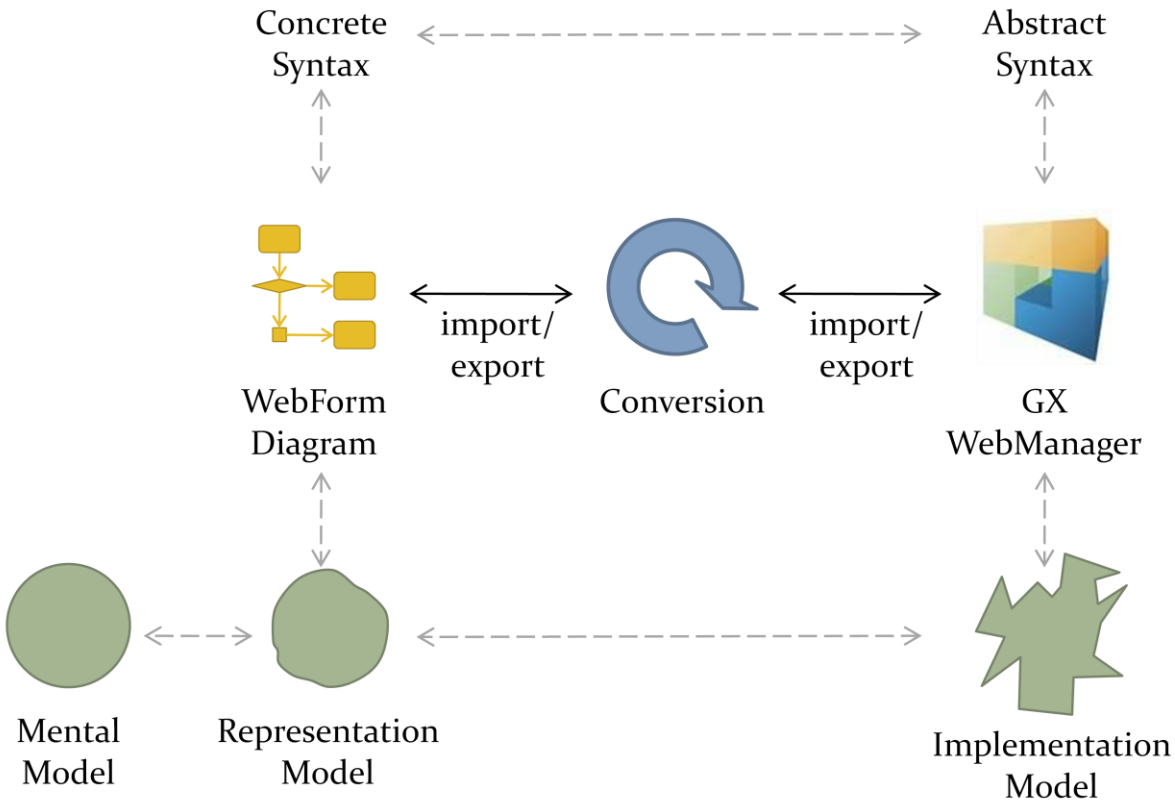
H7 Implementatie

We hebben verschillende eigenschappen van modelleertalen bekeken. Op basis hiervan en na gesprekken met verschillende gebruikers zijn we tot een definitie van de modelleertaal, de WebForm Diagram gekomen. Hiervan hebben we zowel de grafische syntax, concrete syntax en abstracte syntax van gedefinieerd.

Naast deze theoretische kaderzetting is het ook belangrijk om het WebForm Diagram in de praktijk te testen. Hierbij wordt duidelijk of de koppeling tussen het diagram en WebManager werkelijk te realiseren is. Daarbij zal ook blijken hoe het model aansluit bij het mental model van de gebruiker.

7.1 Koppeling tussen WebForm Diagram en WebManager

De concrete syntax is de formele representatie van het WebForm Diagram. Deze wordt geconverteerd en ingelezen in GX WebManager welke gelinkt is aan de abstracte syntax. Deze conversie beschrijft dus eigenlijk de conversie van concrete naar abstracte syntax. De abstracte syntax, geeft het Implementatie model weer; de concrete syntax het representatie model, welke in de buurt komt van het mental model van de gebruiker. Dit alles is nogmaals weergegeven in Figuur 24.



Figuur 24 De implementatie verbindt de concrete abstracte syntax met elkaar. Ook de relaties met het implementatie, representatie en mental model worden hier duidelijk.

Het WebForm Diagram is geïmplementeerd door middel van MetaEdit+. Een xml-export dient als invoer van het conversie proces. Binnen dit proces wordt de MetaEdit-xml vertaald naar WebManager-xml. Deze wordt vervolgens geïmporteerd binnen WebManager, waar het formulier gebruikt kan gaan worden. In het resterende gedeelte van dit hoofdstuk, wordt MetaEdit+, GX WebManager en de conversie tussen beide verder uitgelegd.

MetaEdit+

MetaEdit+ is een multi-CASE tool (Computer Aided Systems Engineering). Het is een commercieel tool voor het ontwerp van (domein specifieke) modeleer talen. De tool biedt de mogelijkheid een complete CASE omgeving inclusief metamodel te definiëren. De definitie van dit metamodel bevat grafische elementen, beperkingregels en objecten. Op basis van deze definities biedt MetaEdit+ verschillende CASE-tool functionaliteiten, waaronder een modelleertool, datamanagement en integratie met andere applicaties middels xml import/export en een API.

Het metamodel is gebaseerd op de GOPPRR (Graph, Object, Port, Property, Relationship en Role) metamodel. Graph specificeert de diagram techniek. Zo bevat UML een Class Diagram en een State Transition Diagram. De semantiek van een graph zijn gedefinieerd als relaties tussen objects, relationships, roles en ports binnen een graph. Properties zijn attributen die aan elk van deze objecten gekoppeld kunnen worden (Kelly, Lyytinen en Rossi 1996).

MetaEdit+ is gebruikt om een prototype van de modelleertaal te maken. Door dit prototype, kan worden bekeken of een WebForm diagram, naast de theoretische mogelijkheid, ook in de praktijk gebruikt kan worden om een formulier binnen WebManager te configureren. De concrete syntax is samen met de grafische notatie binnen MetaEdit+ gedefinieerd. Hier kan vervolgens een diagram van worden gemaakt, welke geëxporteerd kan worden naar xml. Deze xml, welke specifiek is voor MetaEdit+, wordt vervolgens in het conversie proces omgezet naar xml welke losstaat van de implementatie van het prototype, MetaEdit+.

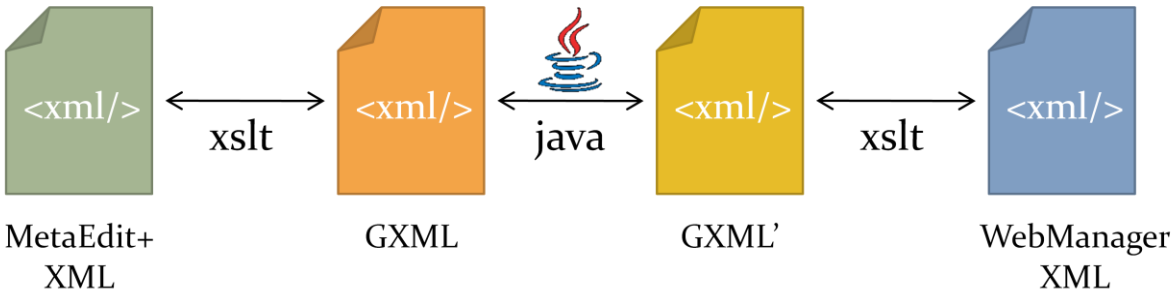
GX WebManager

Conversie van XML

De koppeling tussen MetaEdit+en WebManager wordt gelegd via xml. Het WebForm Diagram kan vanuit MetaEdit+ worden geëxporteerd als xml-bestand. Deze xml is MetaEdit+ specifiek en transformeren we middels xslt naar de concrete syntax: GXML. Vervolgens wordt dit ingelezen met Java, waar de transformatie naar de abstracte syntax plaatsvindt. Deze abstracte syntax is ook weer beschreven in een xml-bestand, GXML'. Hierna wordt de GXML' via xslt getransformeerd tot WebManager xml. Dit bestand kan geïmporteerd worden in WebManager.

Het transformatieproces is tweezijdig, dat wil zeggen dat het ook mogelijk is om een formulier binnen WebManager te exporteren en deze te transformeren naar xml welke geïmporteerd kan worden in MetaEdit+. Het gehele transformatie proces is weergegeven in Figuur 25.

De transformatie via xslt is in feite niet nodig. Met Java zou ook de MetaEdit+ xml kunnen worden ingelezen en WebManager xml worden weggeschreven. Door echter een extra transformatie stap te gebruiken, en GXML als een soort interface te laten dienen voor het Java proces, is het transformatieproces flexibeler. Op dit moment wordt MetaEdit+ ingezet als een prototype. Doordat GXML als een interface dient voor de javatransformatie, kan de implementatie van het diagram, in dit geval met MetaEdit+, worden aangepast zonder de transformatie tussen concrete syntax en abstracte syntax te hoeven aanpassen. Ditzelfde geldt als de implementatie van een formulier binnen WebManager wijzigt.



Figuur 25 De mapping tussen de modelleertaal en Webmanager. Hierbij is GXML de concrete syntax en GXML' de abstracte syntax

7.2 GXML

De GXML dient als interface voor de javatransformatie. Hierbij komt de GXML overeen met de concrete syntax en de GXML' met de abstracte syntax van een WebForm Diagram. Omdat beide syntaxis zoveel op elkaar lijken, komen beide xml varianten ook veel met elkaar overeen.

Tussen GXML en de concrete syntax zijn nog wel wat kleine verschillen De definitie van GXML is vastgelegd in xml-schema. Hierin komen ook de beperkingsregels terug, welke zijn gedefinieerd als axioma's in het vorige hoofdstuk.

Uniek ID

Alle elementen (zoals stappen, velden, validatieregels, handlers, parameters, pagina's, condities en edges) bevatten een uniek ID. Dit ID wordt gebruikt binnen verwijzingen. Zo bevat een stap verschillende velden, waarbij de verwijzing plaatsvindt aan de hand van het ID. Een voorbeeld hiervan is hieronder weergegeven:

```
<steps>
    <step id="_3_2362">
        <name>Step 1</name>
        <field ref="_3_2379" />
        <field ref="_3_2389" />
    </step>
</steps>
<fields>
    <field id="_3_2379">
        <name>Username</name>
        <type>TextField</type>
        <validator ref="_3_2527" />
        <validator ref="_3_2535" />
    </field>
    <field id="_3_2389">
        <name>Password</name>
        <type>TextField</type>
    </field>
</fields>
```

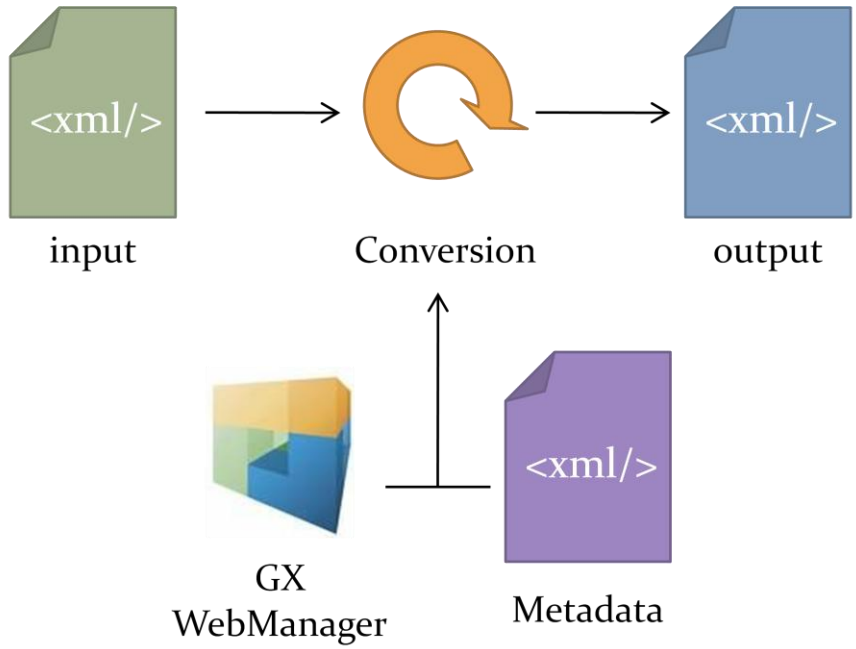
In dit voorbeeld bevatte *Step 1* twee verschillende velden: *Username* en *Password*. Het *Username* veld bevat vervolgens weer verschillende validatieregels. In verschillende parameters, die bijvoorbeeld worden gebruikt in een handler of een validatieregel, kan vervolgens weer worden verwezen naar een invoerveld. Ook dit kan dan worden gedaan door naar zijn ID te refereren.

Object typen en parameter typen

De definitie van de verschillende objecttypen en parametertypen zijn niet beschreven in het GXML bestand. Hiervoor is gekozen, omdat de definitie van bijvoorbeeld een validatieregel of handler, los staat van de definitie van een formulier. Een handler, zoals *verstuur e-mail*, kan bijvoorbeeld in verschillende formulieren worden gebruikt. Een definitie van een handler kunnen beter niet op meerdere plekken voorkomen. Als de definitie dan wijzigt, moet er immers op veel verschillende plekken de wijziging worden doorgevoerd, met het risico op fouten. Daarnaast is er ook het risico dat er verschillende versies van een definitie in gebruik zijn. Door de definitie van een validatieregel, handler of router op één plek te hebben en daar steeds naar te verwijzen, voorkomt je dit probleem. Dit komt overeen met het DRY-principe (Don't Repeat Yourself), (Hunt and Thomas 1999).

De bestaande validatieregels, handlers en routers zijn gedefinieerd binnen WebManager. Deze definities bevatten echter niet genoeg informatie. In een extern xml-bestand staat nog extra meta-informatie over de object typen en parameter typen beschreven. Deze informatie samen, wordt gebruikt bij de conversie, waardoor definitie van de object typen maar op één punt vereist is. Dit is ook weergegeven in Figuur 26.

Zo staan de verschillende typen routers beschreven in het metabestand. Hierin staat beschreven in welke situatie, welke router moet worden gebruikt. Bijvoorbeeld bij de navigatie naar een stap of een pagina, waarbij wel of geen preconditie mag gelden.



Figuur 26 De definities van validatieregels, handlers en routers uit WebManager, worden aangevuld met metadata en gebruikt in het conversie proces.

Daarnaast bevatten parameters van handlers, validatieregels en routers binnen WebManager ook een technische naam. De naam wordt niet gebruikt binnen het Mental Model. De vertaling van deze naam vindt ook plaats met informatie uit de metadata.

7.3 Transformatie in Java

De vertaling van MetaEdit+ xml naar WebManager xml gebeurt door het gebruik van Java en xslt. De xslt wordt gebruikt om de xml te vertalen van en naar GXML, zie ook Figuur 25. De vertaling van de concrete syntax naar de abstracte syntax is geïmplementeerd in Java. Dit omdat er logische bewerkingen zijn welke lastig zijn uit te voeren met xslt. Daarnaast is Java de standaard taal voor software ontwikkeling binnen GX. Het bevat veel open source componenten, bijvoorbeeld voor het werken met xml.

Een probleem wat optrad tijdens de vertaling van condities naar precondities voor bijvoorbeeld handlers, was dat het, vanwege de implementatie van WebManager, niet eenvoudig is om een preconditie aan een bestaande handler te koppelen. De preconditie is namelijk gekoppeld aan de definitie van de handler. Indien deze preconditie wijzigt, wijzigt dus ook de werking van de handler, dus ook op plekken waar deze preconditie niet wordt gebruikt.

Door binnen WebManager een extra parameter te definiëren welke gekoppeld is aan de preconditie, is het wel mogelijk om verschillende precondities aan een handler te koppelen. Een nadeel van deze methode is, dat de implementatie van het transformatietraject niet meer alle handlers ondersteunt. Alleen de handlers en routers, met een extra preconditieparameter kunnen nu gebruikt worden. Echter, omdat de huidige implementatie nog een prototype is, is dit probleem niet groot. Een eventuele wijziging in de architectuur van GX WebManager kan dan dit probleem oplossen.

Tevens kunnen er alleen xsl-condities worden gespecificeerd als conditie object. Dit zijn alle condities die binnen WebManager executeerbaar zijn als preconditie. In de toekomst zouden ook andere type condities, met meer flexibiliteit voor de eindgebruiker kunnen worden toegevoegd. Een voorbeeld hiervan is een conditie waarbij een invoerveld gecontroleerd kan worden op een bepaalde waarde (bijvoorbeeld *keuze = 'ja'*). Op dit moment is een conditie ook wel mogelijk, maar dan gespecificeerd binnen xsl, wat minder gebruikersvriendelijk is.

De vertaling van condities naar precondities en routers vindt plaats binnen Java. Daarnaast wordt ook de volgorde bepaald waarin de verschillende validatieregels, handlers en routers moeten worden uitgevoerd. Ook worden deze gekoppeld aan de stap waarop dit plaatsvindt. Dit alles gebeurt door het in sectie 6.5 beschreven algoritme.

7.4 Reverse Engineering

Naast dat er een WebForm Diagram kan worden vertaald naar een geconfigureerd formulier, kan het proces ook de andere kant op werken. Uit een geconfigureerd formulier kan, via een conversie, een diagram worden afgeleid. Dit proces heet reverse engineering.

De term *reverse engineering* stamt af vanuit de hardwareanalyse. De term werd daar gebruikt voor het ontwikkelen van specificaties van een bestaand hardwaresysteem door heel precies de onderdelen van het systeem te bekijken.

Het proces van reverse engineering binnen de software is het analyseren van een systeem om:

- De componenten en hun relaties te achterhalen;
- Een andere representatie van het systeem te verkrijgen, eventueel op een hoger abstractieniveau;

(Chikofsky and Cross II 1990)

Het reverse engineering proces kan worden ingezet om van bestaande formulieren documentatie te genereren. Deze documentatie kan bijvoorbeeld gebuikt worden bij het voortzetten van al bestaande projecten, of voor extra ondersteuning voor de helpdesk. Zij zijn niet bekend met het formulier, maar door er een diagram van te hebben, valt de werking van het formulier eenvoudig te begrijpen.

Het reverse engineering proces is precies omgekeerd geïmplementeerd dan het eerder beschreven forward engineering. Er vindt een xml-export plaats vanuit WebManager. Deze xml wordt via xslt geconverteerd naar GXML', wat overeenkomt met de abstracte syntax. Dit wordt vervolgens via een Java-applicatie vertaald naar GXML, de concrete syntax. Hierbij worden de routers en precondities bij handlers omgezet in conditie elementen en worden de edges aangemaakt die de verschillende nodes met elkaar verbindt. Uiteindelijk wordt de concrete syntax geconverteerd naar MetaEdit-xml, welke in MetaEdit+ geïmporteerd kan worden. Dit proces is ook zichtbaar in Figuur 25.

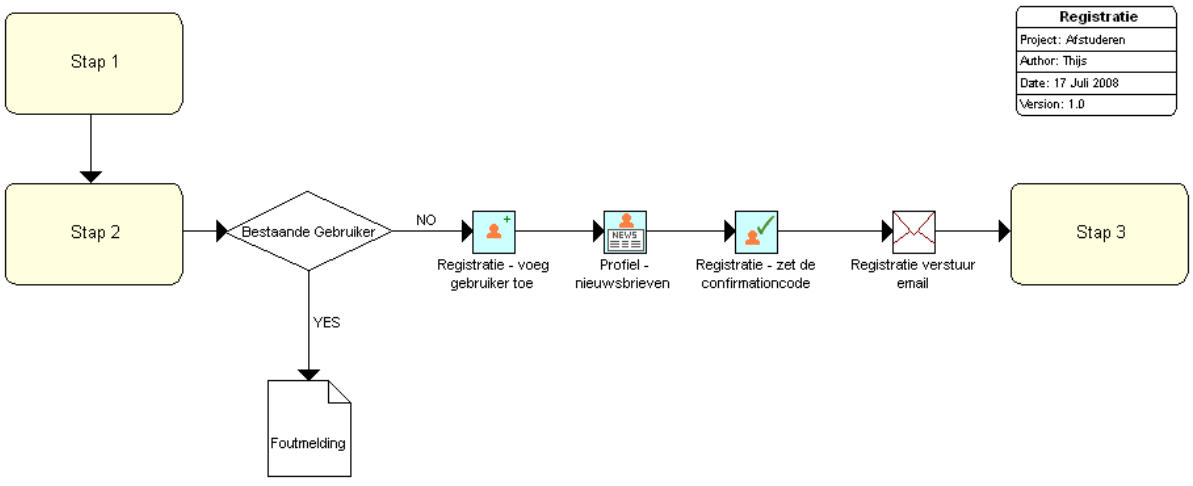
H8 Evaluatie

In de vorige hoofdstukken hebben we aan de hand van verschillende modelleertalen een WebForm Diagram gedefinieerd. Naast de formele definitie is deze geïmplementeerd en is de koppeling tussen het Diagram en een CMS, GX WebManager gerealiseerd. Dit hoofdstuk beschouwt de implementatie, door verschillende meningen van gebruikers en een implementatie van een bestaande case. Aan de hand hiervan worden de verschillende beperkingen beschreven en aanbevelingen gegeven hoe dit verbeterd kan worden.

8.1 Reflectie Gebruikers

Na een eerste gebruikersinterview (zie ook H4), is de WebForm Diagram weer voorgelegd aan een groep gebruikers. Het doel hierbij was het achterhalen of het model in de ogen van de gebruiker bruikbaar is. Hierbij zijn vier gebruikers verschillende diagrammen voorgelegd, waarin de verschillende abstractieniveaus (bijvoorbeeld door het gebruik van *blokken*) aan bod kwamen. Vervolgens werd het WebForm Diagram beoordeeld aan de hand van stellingen, waar een waardering voor moest worden gegeven.

Het diagram werd door alle onderzochte gebruikers goed leesbaar gevonden. Het heeft een heldere opzet en er worden niet te veel *ingewikkelde* diagramtechnieken gebruikt, zodat ook gebruikers met minder modelleerervaring er goed mee overweg kunnen. Ook verbetert het de communicatie met de klant en de architect. Dit is op dit moment echter nog een inschatting van de potentiële gebruikers en nog niet in de praktijk getest. Richting de klant kan het Diagram gebruikt worden om te controleren of het formulier juist gedefinieerd is. De architect kan met het diagram toetsen of alle informatie ten aanzien van de flow beschikbaar is. Over de helderheid van de weergave van handlers en validators kunnen de potentiële gebruikers nog geen goed antwoord geven. Dit is sterk afhankelijk van welke icoontjes hiervoor worden gebruikt. Wel komt naar voren dat deze icoontjes niet te veel detail moeten bevatten. Andere symbolen in het diagram zijn



Figuur 27 Het WebForm Diagram van een registratie-proces.

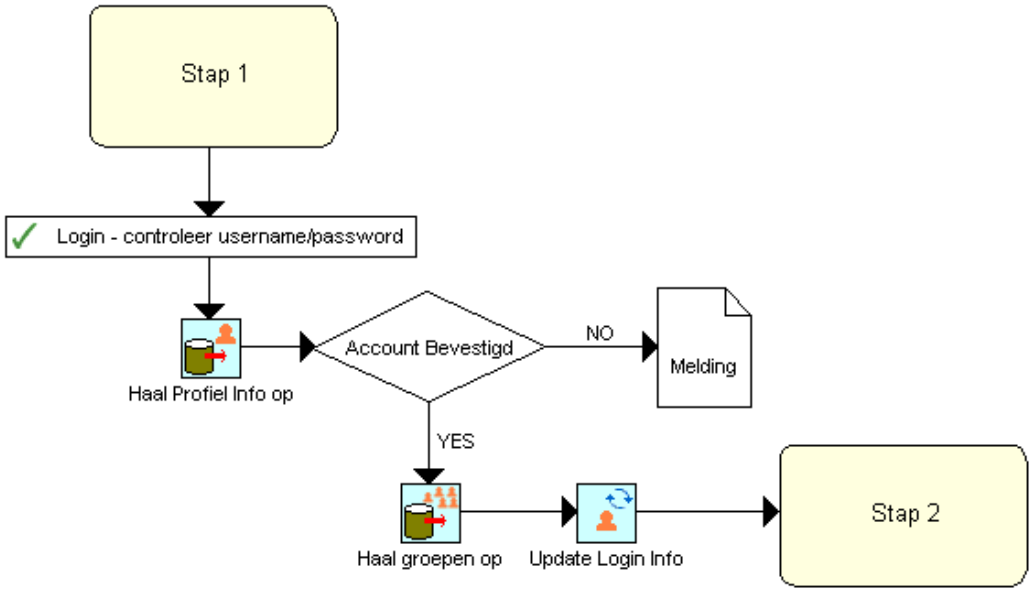
redelijk abstract en een gedetailleerd icoon zou te veel de focus krijgen. Wel vindt iedereen dat een WebForm Diagram een formulier duidelijker en inzichtelijker maakt. Ook zouden de gevraagde gebruikers, mocht er een intuïtieve modelleertool voor aanwezig zijn, het WebForm Diagram gaan gebruiken.

8.2 Implementatie van een Voorbeeld

De gehele implementatie, dat wil zeggen: de WebForm Diagrams, de conversie van diagram naar WebManager en de conversie van WebManager naar een diagram, zijn in de praktijk getest door een voorbeeld uit te werken. Het gaat hierbij om een online registratie proces, welke is gemodelleerd middels een WebForm Diagram en getransformeerd is naar WebManager en de login-functionaliteit waar een diagram is afgeleid uit een WebManager formulier. Er is hierbij geprobeerd om het model zo dicht mogelijk bij de bestaande implementatie te houden.

In Figuur 27 is het WebForm Diagram van een registratie proces weergegeven. Dit proces bestaat uit drie stappen. In de eerste stap dient de gebruiker een gebruikersnaam en een wachtwoord in te vullen. Vervolgens in stap 2 dienen persoonlijke gegevens, zoals naam, adres, woonplaats en e-mail ingevuld te worden. Vervolgens wordt er gekeken of de gebruiker al een account heeft. Is dit het geval, dan wordt de gebruiker naar een speciale foutpagina geleid. Heeft de gebruiker nog geen account, dan wordt deze aangemaakt en een bevestigingsmail naar de gebruiker gestuurd. Vervolgens komt de gebruiker bij Stap 3 waar hij een melding over de bevestigingsmail te zien krijgt. De vertaling van dit diagram naar een formulier binnen WebManager vindt probleemloos plaats.

Het proces van WebManager naar een WebForm Diagram is wat ingewikkelder. Dit komt, omdat een bestaand formulier niet zomaar naar een WebForm Diagram kan worden omgezet. Het WebForm Diagram maakt namelijk gebruik van speciale handlers en routers, met speciale



Figuur 28 Een voorbeeld van een login-proces welke is afgeleid uit WebManager.

preconditie-parameters. Deze parameters komen in de meeste gevallen niet voor in al bestaande formulieren. Daarom is er eerst een kopie van een bestaand formulier gemaakt, waar wel gebruik wordt gemaakt van handlers en routers met speciale preconditie-parameters. Vervolgens is dit formulier geëxporteerd en geconverteerd naar een WebForm Diagram. Het resultaat hiervan is weergegeven in Figuur 28.

8.3 Beperkingen en Aanbevelingen

Naast alle functionaliteit die wel aanwezig is, zitten er ook beperkingen aan de WebForm Diagrams en de (huidige) implementatie hiervan. Deze beperkingen leiden tot een aantal aanbevelingen om deze te verbeteren.

8.3.1 Precondities

De eerste beperking is het gebruik van precondities en met name over de implementatie hiervan. Binnen GX WebManager zijn precondities bij handlers en routers gedefinieerd bij de definitie van de handler of router. Deze ligt dus per handler en router vast en kan tijdens de configuratie hiervan niet gewijzigd worden. Uitzondering hierop is een handler met een expliciete *precondition* parameter. Hierbij kan tijdens de configuratie de preconditie via deze parameter worden ingesteld. Door handlers verplicht te stellen deze parameter te hebben, kan binnen het prototype toch de functionaliteit van precondities worden geïmplementeerd.

Omdat binnen handlers van bestaande projecten deze parameter niet voorkomt, is het niet mogelijk deze handlers te gebruiken binnen een WebForm Diagram. Een mogelijke oplossing zou kunnen zijn dat binnen WebManager, bij alle handlers en routers, een mogelijkheid is om de preconditie in te stellen op configuratie niveau. Hiermee wordt nog steeds dezelfde functionaliteit aan een gebruiker geboden, maar zijn wel alle handlers en routers te gebruiken in een WebForm Diagram.

8.3.2 Routers

Binnen de huidige versie van WebManager zitten verschillende soorten routers. Deze leiden de bezoeker naar een pagina of een stap, afhankelijk van een bepaalde conditie. Echter, deze conditie is niet altijd als preconditie uitgedrukt, maar soms ook als een sessie variabele, of deze controle vindt ergens onder water plaats. De functionaliteit van deze routers is niet uit de definitie in WebManager af te leiden.

Ook deze routers ondersteunt het WebForm Diagram niet. In principe kunnen alle bestaande routers vertaald worden naar routers met als parameter een preconditie. In die vorm worden routers wel ondersteund.

8.3.3 Test in de praktijk

Potentiële gebruikers geven wel aan dat de WebForm Diagrams bepaalde aspecten in het implementatietraject van CMS'en verbeterd, maar de WebForm Diagrams zijn nog niet in de praktijk getest. Een volgende stap zou dus zijn dat de WebForm Diagrams binnen een project in de praktijk getest wordt. Hieruit zal dan blijken of het inderdaad de communicatie, op verschillende niveaus, verbeterd en of de kwaliteit daarmee ook verbeterd.

Mocht blijken dat de WebForm Diagrams in de praktijk ook van nut zijn, dan kunnen deze worden uitgebreid met andere delen van het CMS.

8.3.4 Modelleertool

Op dit moment is er als prototype gebruik gemaakt van MetaEdit+ als CASE-tool. De gebruikersinterface is hiervan, vanwege het generieke aspect van MetaEdit+, niet erg intuïtief. Een webbased interface voor het specificeren van het model is hierbij een optie. Hierbij kan een formulier, direct gekoppeld aan een diagram en vice versa. Door deze directe koppeling komt een diagram altijd overeen met de laatste versie van het formulier.

Daarnaast kan de webbased interface ook gebruikt worden voor allerlei analytische toepassingen. Zo zou er binnen het diagram weergegeven worden welke paden een bezoeker het meest bewandeld. Als veelgebruikte paden buiten het verwachtingspatroon vallen, kan dat komen doordat er ergens een fout zit, of omdat zaken onduidelijk voor de bezoeker zijn gespecificeerd. Door zo'n analyse vallen dit soort problemen makkelijk op en kan er beter op worden ingespeeld. Daarnaast kan ook eenvoudig worden weergegeven welke waarden een gebruiker bij bepaalde invoervelden invult (bijvoorbeeld bij meerkeuzevelden). Al deze informatie kan door middel van een diagram inzichtelijk worden weergegeven, waardoor het een goede manier is voor *information visualization* (Tufte 1986). Information Visualization is de communicatie van abstracte data door het gebruik van interactieve visuele interfaces (Keim, et al. 2006).

8.3.5 Bibliotheek van Validators en Handlers

Binnen de huidige implementatie van het WebForm Diagram binnen MetaEdit+ dienen validators en handlers expliciet gedefinieerd te worden. Bij deze zaken moet namelijk eerst het type worden gedefinieerd met de bijbehorende parameters, voordat dit gebruikt kan worden. Dit geldt ook voor validators en handlers die in een reverse engineering proces worden gebruikt.

De definities van deze handlers en validators zijn echter ook binnen WebManager aanwezig. Vaak worden deze specifiek aangemaakt voor een specifiek project, maar zijn daarbij ook bij andere projecten inzetbaar.

Door een bibliotheek bij te houden van al bestaande implementaties van validators en handlers en deze ook automatisch beschikbaar te stellen binnen het WebForm Diagram, wordt het mogelijk om ze te herbruiken. Ook kan in een vroeg stadium van het implementatietraject al worden ingeschat welke functionaliteit al aanwezig, welke nog ontwikkeld moet worden en hoe deze eruit moet gaan zien.

Daarnaast kan hieruit ook eenvoudig een overzicht worden gegeven van welke validators, handlers en routers waar worden gebruikt. Dit kan eventueel van belang zijn als er bijvoorbeeld blijkt dat er in één van deze een bug zit. Doordat centraal is bijgehouden waar deze is gebruikt, kan deze overal eenvoudig worden geüpdatet.

H9 Conclusie

Het doel van dit onderzoek was te kijken hoe een modelleertaal eruit zou zien, waarmee op basis van requirements en use cases, modules binnen een CMS gespecificeerd kunnen worden.

Hierbij zijn bestaande modelleertalen, welke gebruikt worden voor het modelleren van webapplicaties, met elkaar vergeleken. Deze vergelijking is gemaakt aan de hand van een framework, waarbij in verschillende dimensies, verschillende eigenschappen van de talen werden bekeken. Dit framework is gebaseerd op het CREWS framework, welke gebruikt werd voor het vergelijken van ERP systemen. Hieruit bleek dat het Business Process Model (BPM), het User Interaction Diagram (UID) en het UML Class Diagram (UML) het beste aan de eisen te voldoen.

Daarnaast zijn verschillende gebruikers gevraagd welke van deze eigenschappen voor de te ontwikkelen modelleertaal van belang zijn. Hierbij is ook inzicht gekregen in het mental model van deze gebruikers; hoe stellen zij de modelleertaal voor en hoe zien zij de vertaling naar componenten van het CMS. Ook is er een analyse gemaakt van het domein van CMS systemen.

De scope van dit onderzoek heeft zich beperkt tot het modelleren van de werking van formulieren. Deze zorgen voor de belangrijkste interactie met de bezoeker en kunnen erg complex van samenstelling zijn.

Aan de hand van deze analyses is het WebForm Diagram gedefinieerd. Dit diagram, specificeert de werking van een webformulier zoals die binnen een CMS gebruikt wordt. De definitie is opgesplitst in een concrete syntax en een abstracte syntax. De concrete syntax sluit aan bij het mental model van de gebruiker; de abstracte syntax sluit aan bij de implementatie van het CMS. De modelleertaal is geïmplementeerd middels MetaEdit+ en als CMS is GX WebManager gebruikt. De koppeling tussen beide systemen vindt plaats door middel van xml, xslt en Java. De xml welke uit MetaEdit+ kan worden geëxporteerd, wordt met xslt getransformeerd naar GXML. Deze komt overeen met de concrete syntax. Vervolgens wordt dit met Java vertaald in GXML', de abstracte syntax. Dit wordt vervolgens via xslt getransformeerd en geïmporteerd in WebManager. Ook het proces de andere kant op, het reverse engineering proces, is geïmplementeerd.

Doordat het diagram verschillende abstractieniveaus ondersteunt, kan het worden toegepast in verschillende fasen van het ontwikkelproces. Tijdens het achterhalen van de requirements kan het diagram worden gebruikt in de communicatie met de klant. Een diagram kan een discussie over de werking van een formulier beter ondersteunen dan een vrij interpreteerbaar stuk tekst. Als de requirements duidelijk zijn kan verdere detail in het diagram verder gespecificeerd worden, waarna vanuit het diagram een formulier binnen het CMS gegenereerd kan worden. Ook verschillende potentiële gebruikers van de modelleertaal zijn van mening dat de modelleertaal het proces in positieve zin ondersteund.

Door het gebruik van de WebForm diagrams, wordt de communicatie verbeterd wat ten goede komt aan de kwaliteit van de implementatie en de efficiëntie waarmee deze implementatie kan worden uitgevoerd. Het maakt een formulier inzichtelijker en biedt vanwege zijn koppeling met het CMS een goede documentatie mogelijkheid.

9.1 Verder Onderzoek

Binnen dit onderzoek is slechts gekeken naar het modelleren van formulieren binnen een CMS. Potentiële gebruikers verwachten een hogere kwaliteit en een grotere efficiëntie van de implementatie van formulieren. Deze stelling zou in de praktijk nog getoetst moeten worden door het in te zetten in één of twee (grote) projecten.

Als blijkt dat een domein specifieke modelleertaal, zoals het WebForm Diagram, binnen de omgeving van een CMS van nut is, kan de modelleertaal worden uitgebreid naar andere zaken binnen het CMS. Dit kan bijvoorbeeld zijn het beheren van content op een pagina, workflow-autorisatie en interactiemanagement, personalisatie, multimediale content, enzovoorts.

Daarnaast kan worden gekeken hoe een modelleertool verder geïntegreerd kan worden binnen de GX WebEngineering Method. Op dit moment is er als prototype gebruik gemaakt van MetaEdit+ als CASE-tool. Een webbased interface voor het specificeren van het model sluit beter aan bij de GX WebEngineering Method. Daarnaast kan de web-based interface ook gebruikt worden voor allerlei analytische toepassingen zoals een weergave van de bezoekersstroom, of andere informatie visualisaties.

H10 Bibliografie

Ahn, L. von, M. Blum, en J. Langford. „Telling humans and computers apart automatically." *Communications of the ACM* 47, nr. 2 (2004): 56-60.

Amelsfoort, P. van, geïnterviewd door Thijs Kupers. *Gebruikersonderzoek Formulierenmodule van GX WebManager* (4 april 2008).

Austerberry, D. *Digital Asset Management.* Focal Press, 2004.

Baker, S., K. Baker, en G.M. Campbell. *The Complete Idiot's Guide to Project Management.* New York: Alpha Books, 2003.

Bovy, K., geïnterviewd door Thijs Kupers. *Gebruikersonderzoek Formulierenmodule van GX WebManager* (8 april 2008).

Brabrand, C., A. Møller, M. Ricky, en M.I. Schwartzbach. „PowerForms: Declarative client-side form field validation." *World Wide Web* 3, nr. 4 (2000): 205-214.

Brambilla, M., S. Ceri, P. Fraternali, en I. Manolescu. „Process modeling in Web applications." *ACM Transactions on Software Engineering and Methodology (TOSEM)* 15, nr. 4 (2006): 360-409.

Burzagli, L., M. Billi, F. Gabbanini, P. Graziani, en E. Palchetti. „The Use of Current Content Management Systems for Accessibility." *Lecture Notes in Computer Science* 3118 (2004): 331-338.

Ceri, S., P. Fraternali, en A. Bongio. „Web Modeling Language (WebML): a modeling language for designing Web sites." *Proceedings of the 9th international World Wide Web Conference on Computer Networks : the international Journal of Computer and Telecommunications Networking*, 2000: 137-157.

Chikofsky, E.J., en J.H. Cross II. „Reverse Engineering and Design Recovery: A Taxonomy." *IEEE Software* 7, nr. 1 (1990): 13-17.

Cooper, A., R. Reimann, en D. Cronin. *About Face 3: The Essentials of Interaction Design.* New York, NY, USA: Whiley publishing inc., 2007.

Deursen, A. van, P. Klint, en J. Visser. „Domain-specific languages: an annotated bibliography." *ACM SIGPLAN Notices* 35, nr. 6 (2000): 26-36.

Dourish, P., et al. „Extending document management systems with user-specific active properties." *ACM Transactions on Information Systems (TOIS)* 18, nr. 2 (2000): 140-170.

Evans, A., en S. Kent. „Core meta-modelling semantics of UML : The pUML approach." In *UML '99 - The Unified Modeling Language*, door G. Goos, J. Hartmanis en J. van Leeuwen, 140-155. Berlin / Heidelberg: Springer, 1999.

Fein, R.M., en G.M., Olson, J.S. Olson. „A mental model can help with learning to operate a complex device." *INTERACT '93 and CHI '93 Conference Companion on Human Factors in Computing Systems*, 1993: 157-158.

Fielden, N. *History of Information Retrieval Systems & Increase of Information Over Time.* 10 5 2002. http://online.sfsu.edu/~fielden/hist.htm (geopend 03 21, 2008).

Geerts, G., en T. den Boon. *Van Dale groot woordenboek der Nederlandse taal 13e editie.* Utrecht/Antwerpen: Van Dale Lexicografie, 1999.

Gnaho, C. „Web-Based Information Systems Development - A User Centered Engineering Approach." *Lecture Notes In Computer Science* 2016 (2001): 105-118.

Guelfi, N., en A. Mammar. „A formal framework to generate XPDL specifications from UML activity diagrams." *Proceedings of the 2006 ACM Symposium on Applied Computing*, 2006: 1224-1231.

Güell, N., D. Schwabe, en P. Vilain. „Modeling Interactions and Navigation in Web Applications." *Lecture Notes In Computer Science* 1921 (2000): 115-127.

Henderson-Sellers, B., en F. Barbier. „Black and White Diamonds." In *The Unified Modeling Language: Beyond the Standard*, door R. France en B. Rumpe, 139-156. Berlin: Springer-Verlag, 1999.

Hirschheim, R., H.K. Klein, en K. Lyytinen. *Information Systems Development and Data Modeling: Conceptual and Philosophical Foundations.* New York, NY, USA: Cambridge University Press, 1995.

Hoogerwerf, C., geïnterviewd door Thijs Kupers. *Gebruikersonderzoek Formulierenmodule van GX WebManager* (16 april 2008).

Hrbacek, K., en T. Jech. *Introduction to Set Theory, third edition, Revised and Expanded.* New York: Marcel Dekker, 1999.

Hunt, A., en D. Thomas. *The Pragmatic Programmer: from Journeyman to Master.* Addison-Wesley Longman Publishing Co., Inc., 1999.

Kang, K.C., S.G. Cohen, J.A. Hess, W.E. Novak, en A.S. Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study.* Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 1990.

Keim, D.A., F. Mansmann, J. Schneidewind, en H. Ziegler. „Challenges in Visual Data Analysis." *Proceedings of the Conference on information Visualization*, 2006: 9-16.

Kelly, S., K. Lyytinen, en M. Rossi. „MetaEdit+ A fully configurable multi-user and multi-tool CASE and CAME environment." *Advanced Information Systems Engineering*, 1996: 1-21.

Kleppe, A.G., J. Warmer, en W. Bast. *MDA Explained: the Model Driven Architecture: Practice and Promise.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

Koch, N. *Software Engineering for Adaptive Hypermedia Systems: Reference Model, Modeling Techniques and Development Process.* PhD Thesis, Ludwig-Maximilians-Universität München: Uni-Druck Publishing Company, 2000.

Koch, N., en A. Kraus. „The expressive Power of UML-based Web Engineering." *Proc. of IWWOST02, CYTED*, 2002: 105-119.

Koenen, M., geïnterviewd door Thijs Kupers. *Gebruikersonderzoek Formulierenmodule van GX WebManager* (2 april 2008).

Krogstie, J. „Conceptual Modeling for Computerized Information Systems Support in Organizations." PhD Thesis, Trondheim, 1995.

Ladage, I., geïnterviewd door Thijs Kupers. *Gebruikersonderzoek Formulierenmodule van GX WebManager* (2 april 2008).

Lee, S.C., en A.I. Shirani. „A component based methodology for Web application development." *Journal of Systems and Software* 71, nr. 1-2 (2004): 177-187.

McKeever, S. „Understanding Web content management systems: evolution, lifecycle and market." *Industrial Management & Data Systems* 103, nr. 9 (2003): 686-792.

Melia, S., en C.C. Castro. „An MDA approach for the development of Web applications." *Lecture Notes in Computer Science* 3140 (2004): 300-305.

Melia, S., en J. Gómez. „Applying Transformations to Model Driven Development of Web Applications." *Lecture Notes in Computer Science* 3770 (2005): 63-73.

Meliá, S., J. Gómez, en N. Koch. „Improving Web Design Methods with Architecture Modeling." *Lecture Notes in Computer Science* 3590 (2005): 53-64.

Mierlo, M. van, geïnterviewd door Thijs Kupers. *Gebruikersonderzoek Formulierenmodule van GX WebManager* (31 maart 2008).

Mobasher, B. „Data Mining for Web Personalization." *Lecture Notes in Computer Science* 4321 (2007): 90-135.

OMG. *MDA Guide Version 1.0.1.* OMG, 2003.

OMG. *Object Constraint Language - OMG Available Specification - Version 2.0.* Object Management Group, 2001.

—. „Unified Modeling Language: Infrastructure, Version 2.0.” *The Object Management Group.* Object Management Group. 05 07 2005. http://www.omg.org/docs/formal/05-07-05.pdf (geopend 06 12, 2008).

Pastor, O., J. Fons, en V. Pelechano. „OOWS: A Method to Develop Web Applications from Web-Oriented Conceptual Models.” *International Workshop on Web Oriented Software Technology (IWWOST)*, 2003: 65-70.

Rolland, C., en N. Prakash. „Bridging the Gap Between Organisational Needs and ERP Functionality.” *Requirements Engineering* 5, nr. 3 (2000): 180-193.

Rolland, C., et al. „A proposal for a scenario classification framework.” *Requirements Engineering* 3, nr. 1 (1998): 23-47.

Rossi, G., D. Schwabe, en R. Guimarães. „Designing personalized web applications.” *Proceedings of the 10th international Conference on World Wide Web*, 2001: 275-284.

Rumbaugh, J., G. Booch, en I. Jacobson. *The Unified Modeling Language reference manual.* London, UK: Addison-Wesley, 1999.

Schmidt, D.C. „Guest Editor's Introduction: Model-Driven Engineering.” *Computer* 39, nr. 2 (2006): 25-31.

Schwabe, D., en G. Rossi. „An object oriented approach to Web-based applications design.” *Theory and Practice of Object Systems* 4, nr. 4 (1998): 207-225.

Soffer, P., B. Golany, en D. Dori. „ERP modeling: a comprehensive approach.” *Information Systems* 28, nr. 6 (2003): 673-690.

Souer, J., I. van de Weerd, J. Versendaal, en J. Brinkkemper. „Situational requirements engineering for the development of Content Management System-based web applications.” *International Journal of Web Engineering and Technology* 3, nr. 4 (2007): 420-440.

Thompson, S., en T. Torabi. „A Process Improvement Approach to Improve Web Form Design and Usability.” *18th International Conference on Database and Expert Systems Applications (DEXA 2007)*, 2007: 570-574.

Torres, V., P. Giner, en V. Pelechano. „Web Application Development Focused on BP Specifications.” *Taller sobre Procesos de Negocio e Ingeniería del Software (PNIS)*, 2007: 1-7.

Troyer, O.M.F. De, en C.J. Leune. „WSDM: a user centered design method for Web sites.” *Computer Networks and ISDN Systems* 30 (1998): 85-94.

Tufte, E.R. *The Visual Display of Quantitative Information.* Cheshire, CT, USA: Graphics Press, 1986.

Vilain, P., D. Schwabe, en C. Sieckenius de Souza. „A Diagrammatic Tool for Representing User Interaction in UML." *Lecture Notes in Computer Science* 1939 (2000): 133-147.

Warmer, J., en A. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

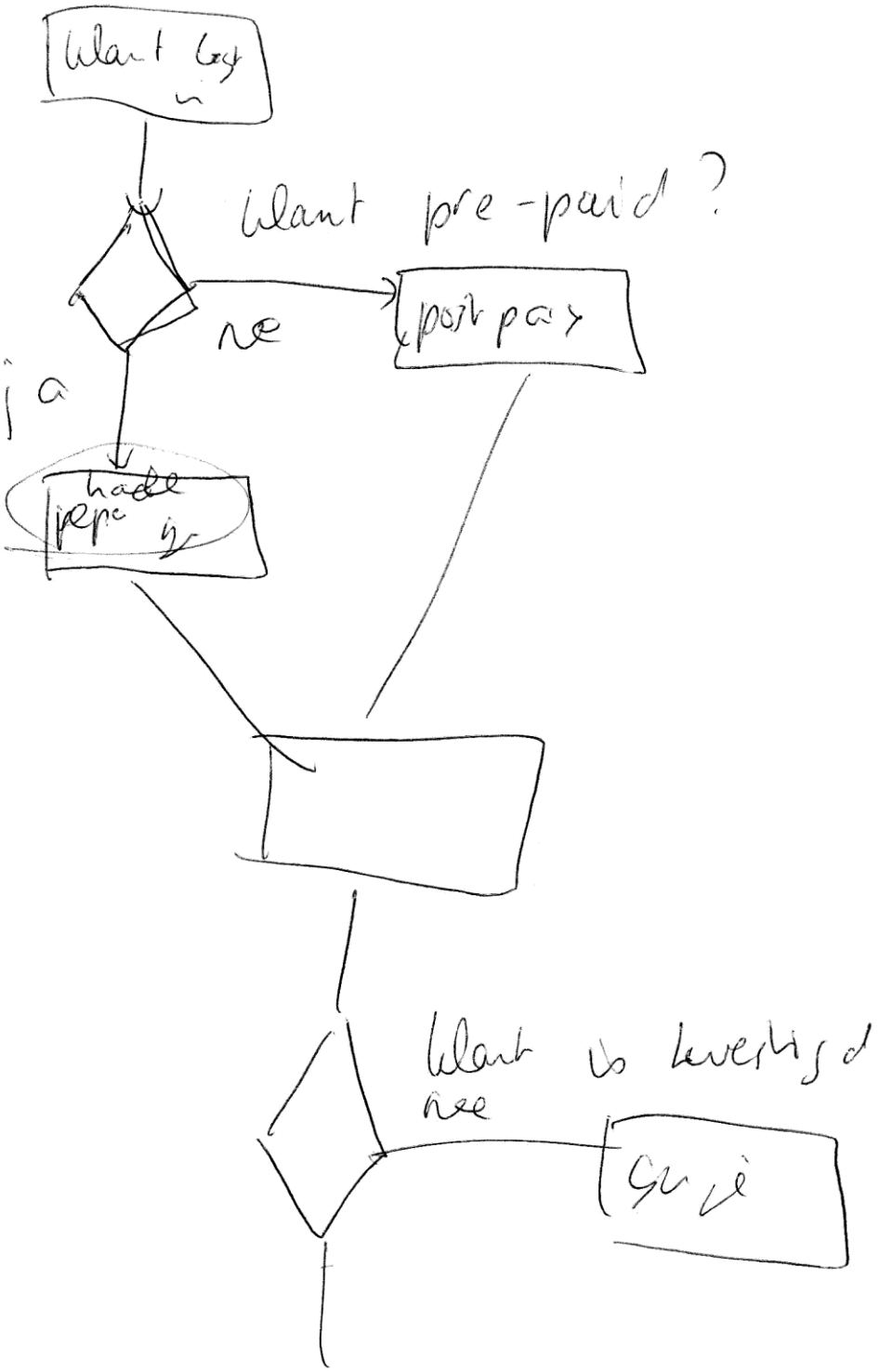
Watters, C., en R. Zhang. „PDA Access to Internet Content: Focus on Forms." In *HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 4*, 105-113. Washington, DC, USA: IEEE Computer Society, 2003.

Weerd, I. van de, S. Brinkkemper, J. Souer, en J. Versendaal. „A Situational Implementation Method for Web-based Content Management System-applications: Method Engineering and Validation in Practice." *Software Process Improvement and Practice* 11 (2006): 521-538.

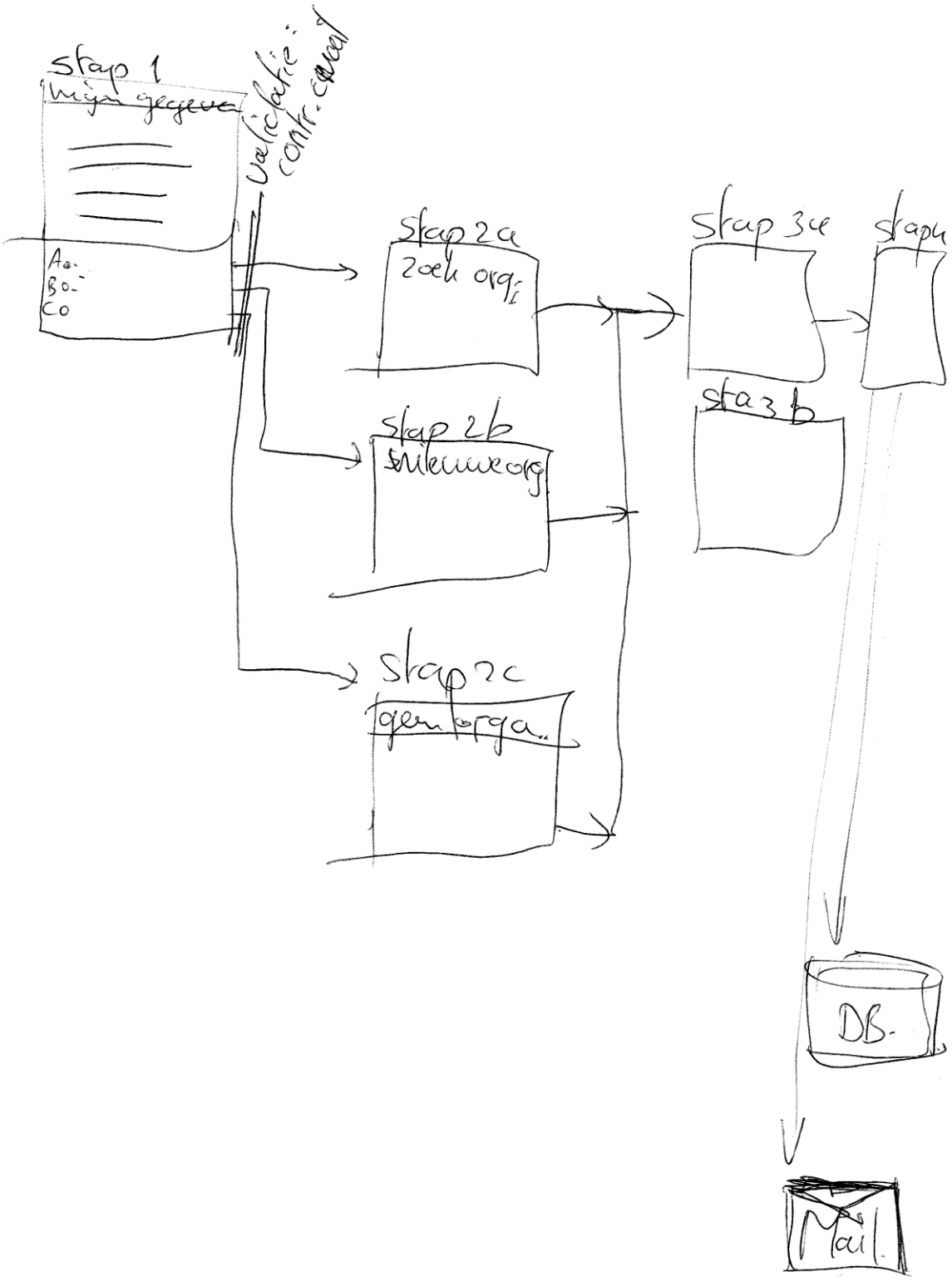
White, S. „Business processing modeling notation (BPMN), version 1.0." http://www.bpmn.org, 2004.

White, S. „Process Modeling Notations and Workflow Patterns." *Workflow Handbook*, 2004: 265-294.

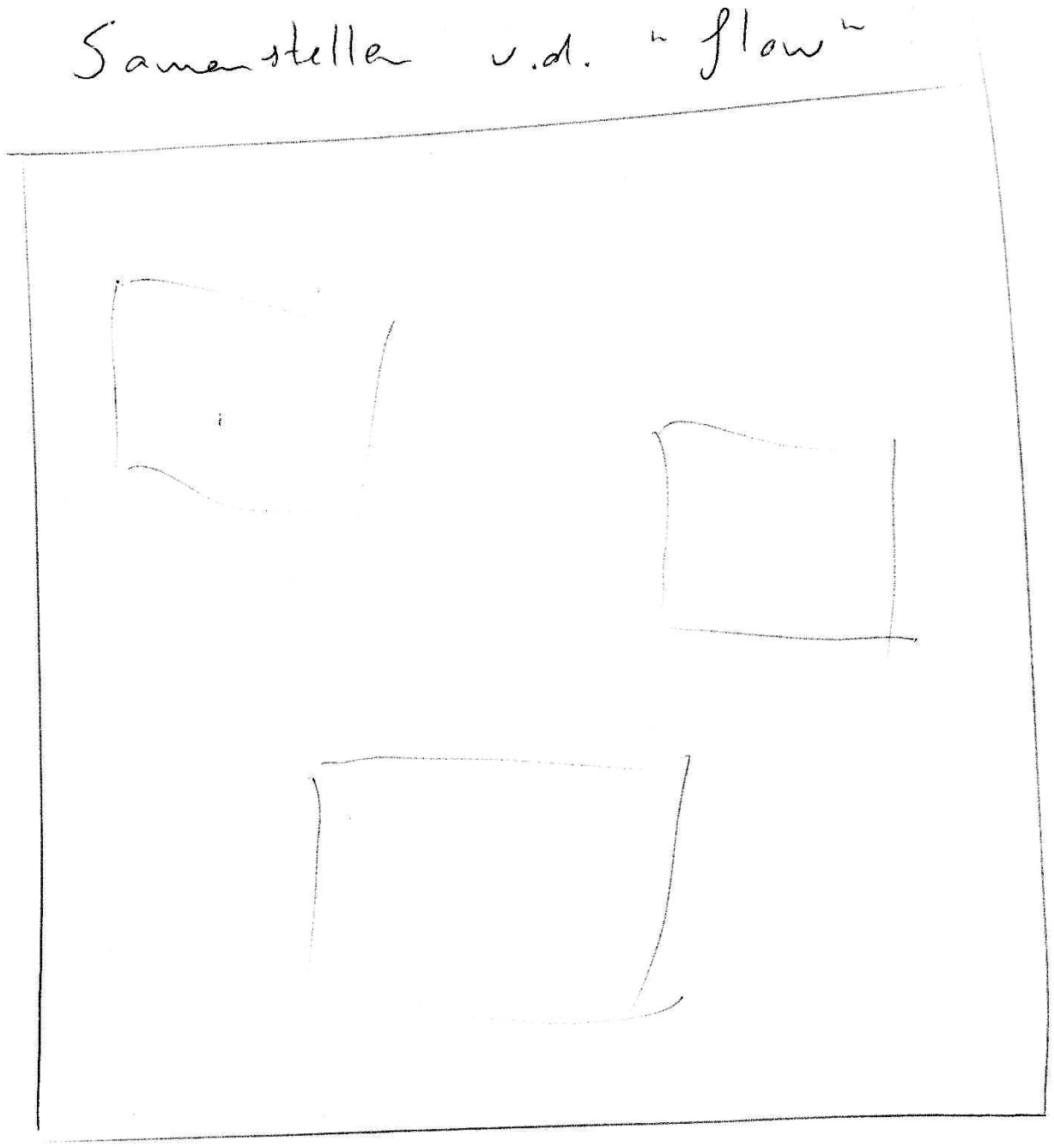
Bijlage I – Gebruikersmodellen



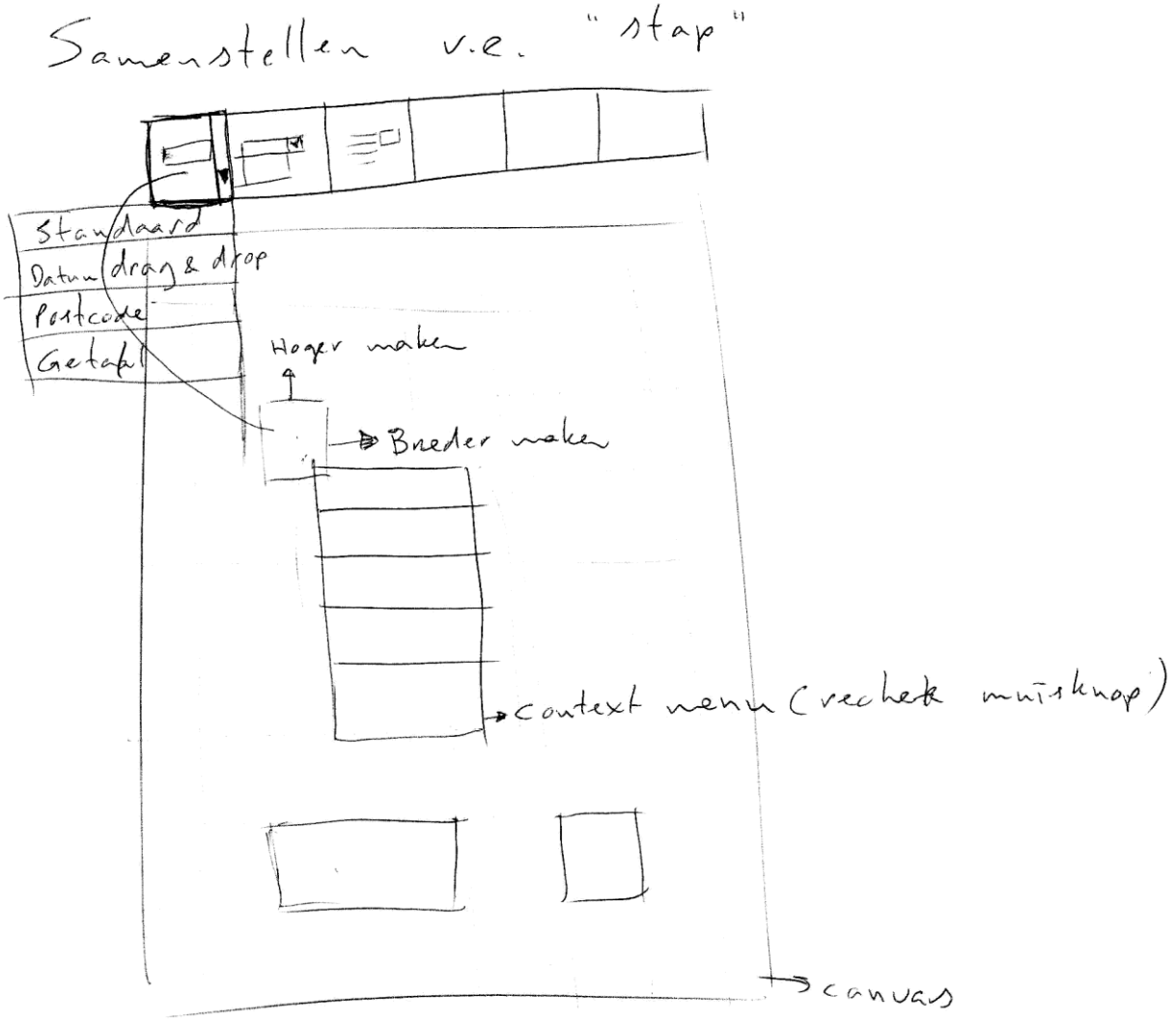
Figuur 29 Het model geeft de flow van een formulier weer (Bovy 2008).



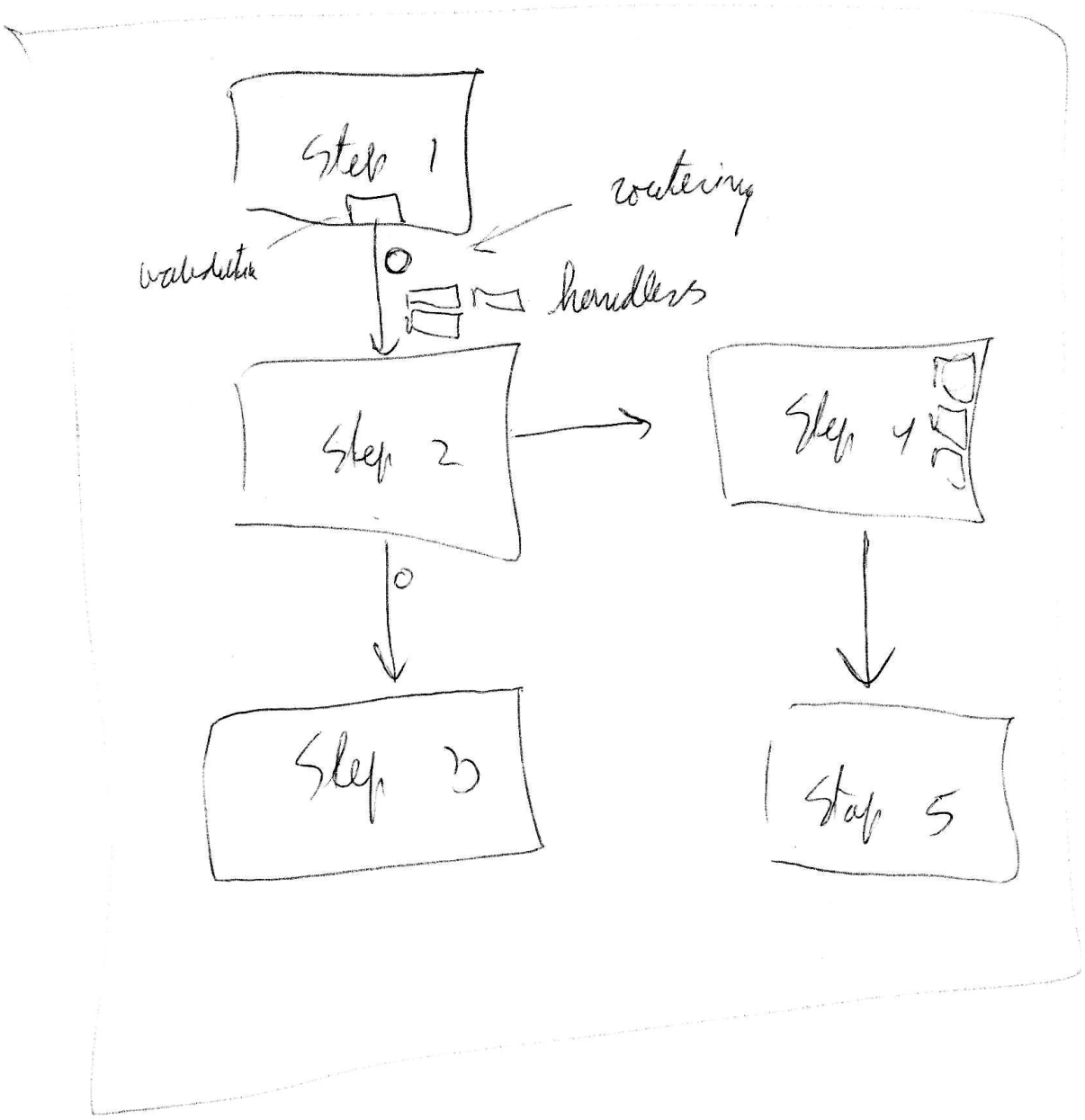
Figuur 30 De flow van een formulier, met handlers en validators (Hoogerwerf 2008).



Figuur 31 Het samenstellen van de flow bestaat uit het tekenen van verschillende stappen (Koenen 2008).



Figuur 32 De velden binnen een stap worden apart van de flow gedefinieerd (Koenen 2008).



Figuur 33 De flow weergegeven met verschillende handlers en routers (Ladage 2008).

Bijlage II – Vergelijkingen Modelleertalen

		OOHDM					**WebML**		
		User Interaction Diagram	Conceptual Class Schema	Navigational Class Schema	Navigational Context Schema	Abstract Data Views	Data Model	Hypertext Model	Business Process Diagram
Facet	Attribuut								
Abstractie	Abstractie	Unique	Unique	Unique	Unique	Unique	Unique	Unique	Flexible
Context	Intern	-	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
	Interactie	TRUE	-	-	-	-	-	-	-
	Contextueel	-	-	-	-	-	-	-	-
Argumentatie	Argument	-	TRUE	TRUE	-	-	TRUE	-	-
Facet	Attribuut								
Notatie	Notatie	Semi-Formal	Formal	Formal	Formal	Semi-Formal	Formal	Formal	Formal
Structuur	Element	{Initial Interaction, Interaction, Optional Interaction, Interaction Alternative, Data Entry, Optional Data Entry, Element, Text, Optional Text, New Interaction, Call of Operation}	{Class, Attribute, Operation, Interface, Association, Generalization, Aggregation, Composition, Note, Package, Directed Association}	{Class, Attribute, Association, Generalization, Aggregation, Directed Association}	{Node, Link, Context Class, Context Group, Index}	{Interface Object, Attribute}	{Class, Attribute, Operation, Interface, Association, Generalization, Aggregation, Composition, Note, Package, Directed Association}	{DataUnit, IndexUnit, MultiDataUnit, EntryUnit, ScrollerUnit, Operation Unit, MultiChoice, Hierarchical, FilterUnit, DirectUnit, Page, Area, Site View, Contextual Link, Automatic Link, Transport Link, Non-Contextual Link}	{Start Activity, End Activity, Assign Operation, If Unit, Case Unit, Service Unit, DataUnit, IndexUnit, MultiDataUnit, EntryUnit, ScrollerUnit, Operation Unit, MultiChoice, Hierarchical, FilterUnit, DirectUnit, Page, Area, Site View, Contextual Link, Automatic Link, Transport Link, Non-Contextual Link}
	Intra-element relatie	Flat	Flat	Flat	Flat	Nested	Flat	Nested	Nested
	Inter-element relatie	{Precedence}	{Composition, Generalization}	{Composition, Generalization}	{Composition}		{Composition, Generalization}	{Composition, Precedence}	{Composition, Precedence}

		UWE				
		Conceptual Model	Navigation Space Model	Navigation Structure Model	Presentation Model	Activity Diagram
Facet	Attribuut					
Abstractie	Abstractie	Unique	Unique	Unique	Flexible	Flexible
Context	Intern	TRUE	TRUE	TRUE	TRUE	TRUE
	Interactie	-	-	-	-	TRUE
	Contextueel	-	-	-	-	TRUE
Argumentatie	Argument	TRUE	TRUE	TRUE	-	TRUE
Facet	Attribuut					
Notatie	Notatie	Formal	Formal	Formal	Formal	Formal
Structuur	Element	{Class, Attribute, Operation, Interface, Association, Generalization, Aggregation, Composition, Note, Package, Directed Association}	{Navigation Class, External Node, Association, Invariant, Note}	{Index, Guided Tour, Query, Menu, Aggregation, Composition, Navigation Class, External Node, Association, Invariant, Note}	{Presentation Class, Text, Anchor, Button, Image, Audio, Video, Form, Collection, Anchored Collection, Adapted Language, Layout Variant}	{Initial Node, Final Node, Activity, Control Flow Edge, Object Flow Edge, Fork, Join, Condition, Decision, Merge, Partition, Sub-activity indicator, Flow Final, Note, Use Case}
	Intra-element relatie	Flat	Flat	Flat	Nested	Nested
	Inter-element relatie	{Composition, Generalization}	{Precedence}	{Composition, Precedence}	{Composition}	{Precedence, AND, OR, XOR}

		OOWS					WebSA	
		Class Diagram	Navigational Map	Navigational Context Model	Presentation Model	Business Process Model	Configuration Model	Integration Model
Facet	Attribuut							
Abstractie	Abstractie	Unique	Fixed	Fixed	Fixed	Flexible	Unique	Unique
Context	Intern	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
	Interactie	-	-	-	-	TRUE	-	-
	Contextueel	-	-	-	-	-	-	-
Argumentatie	Argument	TRUE	-	-	-	TRUE	-	-
Facet	Attribuut							
Notatie	Notatie	Formal	Formal	Formal	Formal	Formal	Formal	Formal
Structuur	Element	{Class, Attribute, Operation, Interface, Association, Generalization, Aggregation, Composition, Note, Package, Directed Association}	{Navigational Sub System, Link}	{Navigational Class, Manager Class, Complentary Class, Association, Composition, Generalization}	{Navigational Class, Manager Class, Complentary Class, Association, Composition, Generalization}	{Event, Activity, Gateway, Sequence Flow, Message Flow, Association, Pool, Lane, Data Object, Group, Annotation}	{WebComponent, WebPort, WebInterface, WebConnector, WebPattern, UserInterface Component, View, ServerPage, Server, Persistence Component, Datasource}	{Concrete Component, Mudle, Concrete Connector}
	Intra-element relatie	Flat	Flat	Flat	Flat	Nested	Nested	Flat
	Inter-element relatie	{Composition, Generalization}	{Precedence}	{Composition, Generalization}	{Composition, Generalization}	{Composition, Precedence, AND, OR, XOR}	{Composition, Association}	{Association}

		OOHDM					WebML		
		User Interaction Diagram	Conceptual Class Schema	Navigational Class Schema	Navigational Context Schema	Abstract Data Views	Data Model	Hypertext Model	Business Process Diagram
Perspectief	Perspectief	Actor & Role	Object	Object	Object	Object	Object	Object	Behavioral
Facet	Attribuut								
Entiteit	Form	Initial Interaction	Package				Package	Area	Area
	Step	Interaction	Class	Class	Context Group	Interface Object	Class	Page	Page
	Subform	Interaction	Class	Class	Context Group	Interface Object	Class	Page	Page
	Form Element		Class	Class	Context	Interface Object	Class		
	Button		Class	Class		Interface Object	Class		
	Information		Class	Class		Interface Object	Class	DataUnit	DataUnit
	Text	Text	Class	Class		Interface Object	Class	DataUnit	DataUnit
	Image		Class	Class		Interface Object	Class	DataUnit	DataUnit
	Input	Data Entry	Class	Class		Interface Object	Class	EntryUnit	EntryUnit
	Text Input Field	Data Entry	Class	Class		Interface Object	Class	EntryUnit	EntryUnit
	List	Data Entry	Class	Class		Interface Object	Class	MultiChoice	MultiChoice
	Upload File	Data Entry	Class	Class		Interface Object	Class	EntryUnit	EntryUnit
	Items	Element Set	Class	Class		Attribute	Class	MultiDataUnit	MultiDataUnit
	Predefinied List		Class	Class			Class	MultiDataUnit	MultiDataUnit
	Query List		Class	Class			Class	MultiDataUnit	MultiDataUnit
	Item		Class	Class			Class	DataUnit	DataUnit
	Visitor Information Type		Interface				Interface	DataUnit	DataUnit
	Database Information Type		Interface				Interface	DataUnit	DataUnit
	Session Information Type		Interface				Interface	DataUnit	DataUnit
	Action							Operation Unit	Operation Unit
	Handler		Operation				Operation	Operation Unit	Operation Unit
	Router	New Interaction, Interaction Alternative	Directed Association	Directed Association	Link		Directed Association	Non-Contextual Link	Non-Contextual Link
	Validator		Operation				Operation	Operation Unit	Operation Unit
	Condition		Operation				Operation	FilterUnit	If-Unit
	Page		Class	Class	Context		Class	Area	Area
	Dataset		Class	Class			Class	MultiDataUnit	MultiDataUnit

		UWE				
		Conceptual Model	Navigation Space Model	Navigation Structure Model	Presentation Model	Activity Diagram
Perspectief	Perspectief	Object	Object	Object	Object	Functional
Facet	Attribuut					
Entiteit	Form	Package				Activity
	Step	Class	Navigation Class	Navigation Class	Presentation Class	Activity
	Subform	Class			Presentation Class	Sub-Activity Indicator
	Form Element	Class				
	Button	Class			Button	
	Information	Class				
	Text	Class			Text	
	Image	Class			Image	
	Input	Class			Form	
	Text Input Field	Class			Form	
	List	Class			Form	
	Upload File	Class			Form	
	Items	Class				
	Predefinied List	Class				
	Query List	Class				
	Item	Class				
	Visitor Information Type	Interface			Layout Variant	
	Database Information Type	Interface				
	Session Information Type	Interface				
	Action					
	Handler	Operation				
	Router	Directed Association	Association	Guided Tour		Control Flow Edge, Decision
	Validator	Operation				
	Condition	Operation	Invariant	Query		Condition
	Page	Class	External Node	External Node		Activity
	Dataset	Class				

		OOWS					WebSA	
		Class Diagram	Navigational Map	Navigational Context Model	Presentation Model	Business Process Model	Configuration Model	Integration Model
Perspectief	Perspectief	Object	Object	Object	Object	Functional	Object	Object
Facet	Attribuut							
Entiteit	Form	Package		Navigational Class	Navigational Class	Activity		
	Step	Class	Navigational Sub System	Manager Class	Manager Class	Activity	WebComponent	Module
	Subform	Class					WebComponent	
	Form Element	Class		Complementary Class	Complementary Class		WebComponent	Concrete Component
	Button	Class					UserInterface Component	
	Information	Class					UserInterface Component	
	Text	Class					UserInterface Component	
	Image	Class					UserInterface Component	
	Input	Class						
	Text Input Field	Class						
	List	Class						
	Upload File	Class						
	Items	Class						
	Predefinied List	Class						
	Query List	Class						
	Item	Class						
	Visitor Information Type	Interface					WebInterface	
	Database Information Type	Interface					WebInterface	
	Session Information Type	Interface					WebInterface	
	Action							
	Handler	Operation						
	Router	Directed Association	Link			Sequence Flow, Gateway		Concrete Connection
	Validator	Operation						
	Condition	Operation				Gateway		
	Page	Class				Activity	WebInterface	
	Dataset	Class					Datasource	