Vrije Universiteit Brussel
Faculteit Wetenschappen – Departement Informatica
Academiejaar 1999 – 2000


# Ontwerpen van Applicatie Type Web Sites binnen WSDM

**Bart Deprez**

Proefschrift ingediend met het oog
*op het behalen van de graad*
*Licentiaat in de Toegepaste Informatica*

Promotor: Prof. Dr. Olga De Troyer

# Dankwoord

Eerst en vooral wil ik hierbij **Prof. Dr. Olga De Troyer** bedanken, om mij de mogelijkheid te geven mijn thesis binnen haar onderzoeksgroep uit te voeren. Ik wil haar ook bedanken voor het geduld dat ze met mij heeft gehad, vooral bij de trage start van het onderzoek voor deze thesis, voor het lezen van de eerste versies van deze thesis, en vooral ook voor de feedback en hulp bij het zoeken naar een correcte oplossing.

Vooral voor haar steun gedurende de moeilijkste dagen wil ik hierbij ook mijn vriendin **Vicky** bedanken, zonder haar hulp zou deze thesis er nooit geweest zijn. Ook een grote dank aan **mijn ouders**, vooral om mij de mogelijkheid te geven te kunnen studeren en mij gedurende al deze jaren te steunen. Daarnaast wil ik mijn broer **Kurt** en mijn neef bedanken om mij geregeld de juiste ontspanning te geven.

Ik wil hierbij ook de firma **Jetimport N.V.** bedanken om mij toe te laten gedurende twee jaar hun co-systeembeheerder te zijn, waarbij ik een heel pak ervaring kon opdoen binnen het echte bedrijfsleven.

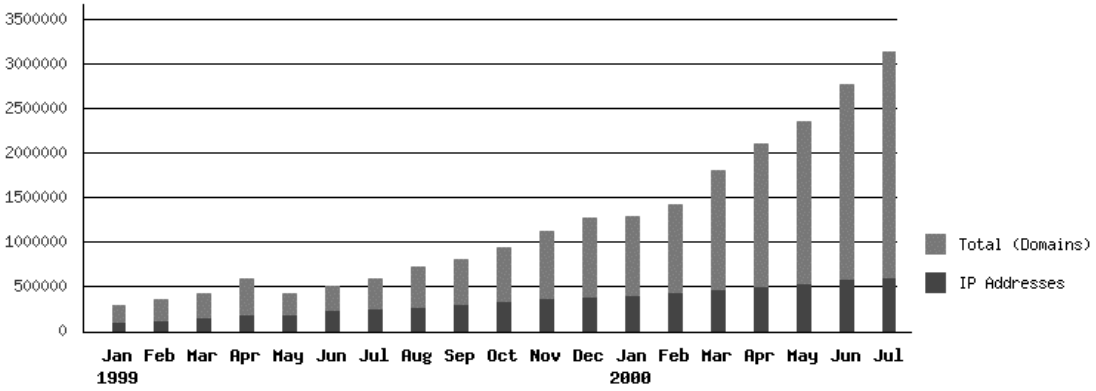
Als laatste wou ik al mijn vrienden bedanken die mij gesteund hebben en tips hebben gegeven. Een speciale dank gaat daarbij uit naar **Jan Plasqui**.

# Inhoud

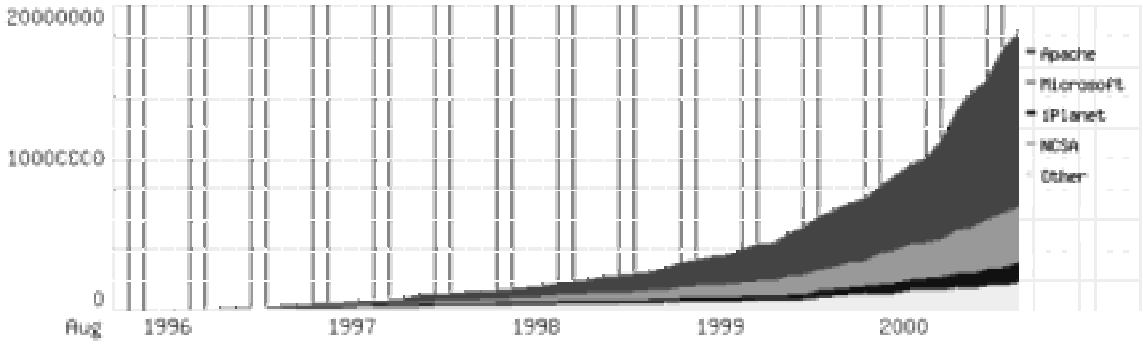
# Lijst van Figuren

# Inleiding

In de laatste jaren is het aantal potentiële gebruikers van het WWW (World Wide Web of internet) gigantisch toegenomen. Daarbij rekening houdend dat het publiceren van web sites heel eenvoudig is, weet iedereen dat het aantal web sites dan ook explosief gestegen is en nog zal stijgen.
Het bijhouden van het aantal web sites dat dagelijks toeneemt, lijkt een onmogelijk opdracht te zijn. Alleen al aan het stijgend aantal servers, waarop web sites geplaatst moeten worden om publiek te zijn, kan men zien dat dit aantal web sites enorm moet zijn.




Figuur 1: aantal web servers dat gebruik maakt van PHP, een scripting taal. [Php]




Figuur 2: aantal servers over alle domeinen over de gehele wereld. [Net]

Web sites zijn voor vele bedrijven ook een bron van inkomsten geworden, aanvullend met het enorme reclamegebeuren dat via het internet mogelijk is geworden.

Daarbij is het een feit dat Internet Systemen meer dan andere applicaties het onderwerp zijn van wijzigingen en dit door diverse redenen.
Uitwendige redenen zijn bijvoorbeeld nieuwe browsers, andere servers, nieuwe standaarden, nieuwe technologieën, …
Inwendige redenen kunnen zijn: een verandering binnen de bedrijfsstructuur, een wijziging van identiteit of profilering van het bedrijf, nieuwe services die het bedrijf wil aanbieden.

Web sites zijn dus praktisch per definitie verplicht om zich constant aan te passen en mee te groeien met de ontwikkelingen binnen het Internet en het bedrijf.
Ze moeten ook meer en meer kunnen samenwerken en communiceren met andere web sites of met andere systemen die al dan niet gekend zijn bij de ontwikkeling van de web site.
Dit zijn situaties die heel complex kunnen worden.

Het ontwikkelen van een web site werd door vele bedrijven tot voor kort gezien als een project dat weinig tijd in beslag mocht nemen. De site moest voor het bedrijf een “interessante” look hebben, maar de rest was voor hen niet belangrijk.
Daardoor werden vele web sites zonder grondige voorbereiding of uitgewerkte structuur op het internet geplaatst.

De gevolgen bleven echter niet uit. Bezoekers van de web site vonden moeilijk hun weg terug, of vonden wat ze zochten helemaal niet, en besloten hun informatie of produkten elders te zoeken.
Een ander probleem dat na verloop van tijd opdook was het onderhoud van de web site. Door de slechte structuur werd dit heel moeilijk, soms zelfs onmogelijk.

Een ander, nieuw probleem dat we bij de vroegere, “klassieke” informatiesystemen nooit tegenkwamen, is de competitie voor de gebruikersaandacht [GMD 98].
Dit probleem doet zich vooral voor bij commerciële web sites, waar het vooral belangrijk is om de aandacht van de gebruikers vast te houden, en om er daarbij voor te zorgen dat ze in de toekomst terugkomen.
Meer dan ooit blijken functionaliteiten zoals gebruiksvriendelijkheid en “user interfaces” van primair belang te zijn.

Door deze verschillende problemen, met tevens een aantal nieuwe die bij de klassieke informatiesystemen eigenlijk niet aan bod kwamen, moest er

uitgekeken worden naar een nieuw soort software engineering. Dit is dan uiteindelijk ontstaan onder de naam: “web engineering”.
Het is toen ook duidelijk geworden dat een dergelijke methode voor ontwerpers van web sites onmisbaar is geworden [DeT 98].

Een aantal onderzoekers hebben dit gebrek aan methoden voor web ontwikkelingen ingezien en zijn met een aantal voorstellen gekomen.

De belangrijkste methoden die beschikbaar zijn op de markt van vandaag zijn RMM [ISB 95], RMM+ [Zol], HDM [GPS 93], OOHDM [SRB 96], WSDM [DL 98, DeT], W3DT [BN 96]. Daarnaast zijn er dan ook methoden die verbeteringen willen aanbrengen door middel van “Pattern Systems” (het zoeken naar patronen) [GRS 97, RSL 98].

Eén van deze methoden, WSDM, dat in het volgend hoofdstuk in detail wordt uitgelegd, is recentelijk verschenen binnen dit onderzoeksgebied.

De methode bevat echter nog enkele onvolmaaktheden en problemen, waarvan ik in deze thesis zal proberen een oplossing te bieden voor één van deze problemen.
Daardoor zou WSDM nog efficiënter moeten zijn in het gebruik en kan er later voorzien worden om er een tool rond te bouwen.

Het aan te pakken probleem is vooral belangrijk bij web sites die gebruik zullen maken van databanken. Dit zijn dan voornamelijk de dynamische web sites.
Terwijl de informatie bij statische web sites praktisch nooit veranderd, worden de pagina’s van een dynamische web site dagelijks, ieder uur of zelfs iedere seconde aangepast. Bij een dynamische web site kunnen we eigenlijk niet meer spreken van pagina’s, omdat de informatie die op de pagina’s wordt weergegeven slechts uit de databank wordt gehaald indien de gebruiker daarom vraagt.

Statische web site zijn voldoende voor organisaties waarvan de informatie heel weinig veranderd. Bedrijven die echter willen communiceren met de gebruiker, of die ervoor willen zorgen dat het web werk voor hen verricht (bijvoorbeeld on-line winkels), hebben nood aan dynamische web sites.

Enkele voordelen van dynamische web sites zijn:

- dynamische web sites vragen minder werk om te onderhouden. Bij statische web sites moeten alle pagina’s afzonderlijk bewerkt worden indien er zich een wijziging voordoet. Vooral voor grote web sites met honderden tot duizenden pagina’s is dit bijna een onmogelijke taak. Bij een dynamische web site moet de structuur slechts op één plaats gewijzigd worden.

- dynamische web sites geven de informatie tijdloos weer, vermits die informatie altijd up-to-date kan gehouden worden.
- dynamische web sites kunnen uiteindelijk goedkoper zijn, vermits het onderhoud heel wat vlotter verloopt.
- indien een bedrijf zaken wil voeren via het internet is een dynamische web site zeker vereist.
- gebruikers maken veel liever gebruik van dynamische web sites, omdat ze de gewenste informatie over het algemeen vlugger terugvinden.


Figuur 3: het verschil tussen een statische en dynamische web site. [Int]

Bij WSDM was er totnogtoe geen onderzoek (of werk) verricht naar het modelleren van web systemen om informatie aan databanken toe te voegen. Enkel het modelleren van de nodige informatie is voorzien.
In deze thesis wil ik hierbij een methode voorzien om de databanktoepassingen die de ontwerper in zijn web site wil voorzien, ook vooraf te kunnen modelleren.

# Hoofdstuk 1:

# Wat is WSDM?

## 1.1 Inleiding

In dit hoofdstuk zullen we een bespreking geven van de methode WSDM, wat voor filosofie er achter WSDM zit en welke de belangrijke stappen van die methode zijn.

Op het einde van dit hoofdstuk worden dan een aantal problemen of tekortkomingen van WSDM kort besproken.

WSDM, of Web Site Design Method (“WiSDoM”) is een nieuwe benadering ([DL 98, DeT]) op gebied van web site design. In plaats van de beschikbare data als startpunt te nemen en te bepalen hoe deze informatie het best op de web site wordt geplaatst, wordt er bij WSDM gestart met het bepalen van de potentiële gebruikers van de web site, daarbij rekening houdend met hun noden en vereisten.
Deze aanpak wordt “audience-driven” genoemd. WSDM houdt dus zoveel mogelijk rekening met de mensen die deze web site zullen bekijken, door in acht te nemen dat er verschillende “soorten” gebruikers kunnen zijn. De verschillende types van gebruikers kunnen dan ook verschillende vereisten hebben.

In de eerste stap van de methode worden de types gebruikers dan ook opgesplitst in gebruikersklassen. Iedere klasse zal dan een beschrijving krijgen van de vereisten die de leden hebben, zodat ze op een goeie manier hun weg kunnen vinden op de web site. Deze aanpak leidt tot een grotere gebruiksvriendelijkheid en tevredenheid van de gebruikers.

Er wordt ook een duidelijk verschil gemaakt tussen het conceptuele design (dat onafhankelijk is van ieder implementatiegedeelte) en de design van de presentatie zelf. Die laatste houdt immers rekening met de implementatietaal, gebruik van menu’s, statische en dynamische links, …
Dit onderscheid kan vergeleken worden met het onderscheid gemaakt in database design tussen het conceptuele schema en het logische schema.
Het opsplitsen van het conceptuele en de implementatiedesign heeft als voordeel dat de methode niet afhankelijk is van de huidige mogelijkheden en beperkingen van de web technologie.

Een ander voordeel van het conceptueel schema dat wordt verkregen tijdens het uitwerken van het conceptueel design is dat dit schema ook beschikbaar kan worden gesteld aan het publiek van de web site. Op die manier kunnen bezoekers van de web site onmiddellijk zien of de informatie die ze zoeken al dan niet aanwezig is. Ook kan het “lost-in-hyperspace” syndroom [Ibi, Ise, Pii] op die manier opgelost worden.

WSDM is vooral gericht op “statische” web sites. Hoewel, in dit geval hier, de term statisch misleidend kan zijn, wordt het gebruikt om het onderscheid duidelijk te maken met de zogenaamde “dynamische” of “actieve” web sites [Int].
In tegenstelling tot wat de gebruiker zou denken, kan een statische web site toch dynamische elementen bevatten zoals video of animatie en kunnen de pagina’s “on the fly” gegenereerd worden. Het uitzicht dat de gebruiker krijgt blijft hierbij echter statisch.

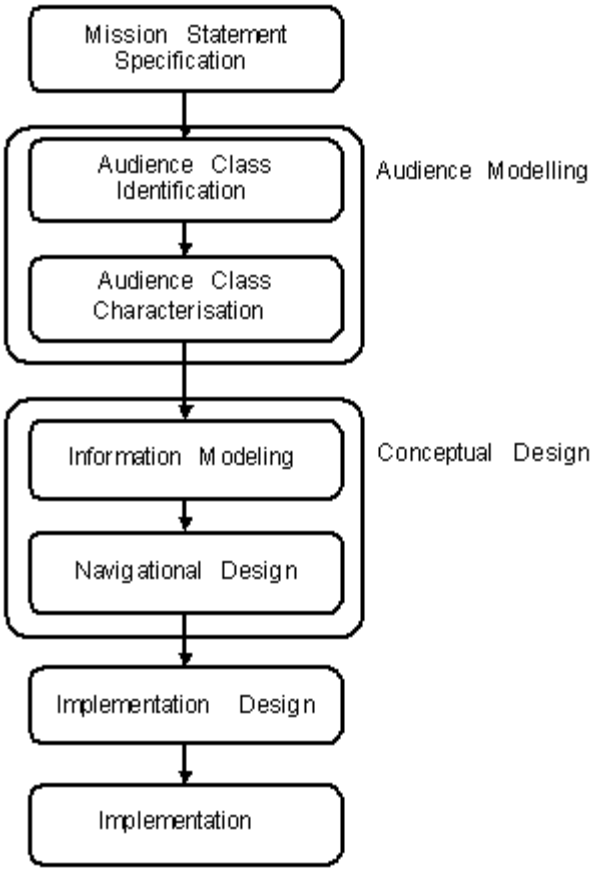
Er wordt ook een onderscheid gemaakt tussen kiosk type en applicatie type web sites. Een kiosk type web site levert voornamelijk informatie en de gebruikers kunnen daarbij door deze informatie navigeren. Een applicatie type web site is een soort interactief informatiesysteem waarbij de user interface bepaald wordt door een verzameling webpagina’s.
Bij de design van kiosk web sites is het niet alleen de bedoeling om ze aantrekkelijk te maken, maar vooral om onderhoudbaarheid, betrouwbaarheid en bruikbaarheid te verbeteren.
Daardoor is het belangrijkste doel het structureren van de informatie binnen het juiste domein en de presentatie naar de gebruikers toe op een duidelijke en eenvoudige wijze te verzorgen.

Het structureren van deze informatie komt overeen deze bij database design. Zonder structuur komen dezelfde onderhoudsproblemen als bij databanken te voorschijn: redundantie, inconsistentie, onvolledigheid, veroudering van informatie. Deze problemen leiden bij web sites dan nog eens extra tot gebruikersproblemen.
Bij redundantie van informatie zal deze informatie hoogstwaarschijnlijk meerdere keren herhaald worden bij het navigeren wat zeer vervelend overkomt.
Zowel bij inconsistentie als veroudering van de informatie zal de geloofwaardigheid van web site sterk verminderen bij de gebruiker.
Ook de onvolledigheid door “gebroken” links of het niet vinden van informatie is heel vervelend waardoor bezoekers van de web site snel hun informatie elders zoeken.

Zoals voordien reeds vermeld is WSDM gebaseerd op de “gebruikers” en niet op de data. De web site zal dus gebaseerd zijn op de vereisten van de (waarschijnlijke) bezoekers. Om dit te kunnen verwezenlijken moet

WSDM dus rekening houden met het feit dat een web site verschillende types van bezoekers kan hebben.
De vereisten van de gebruikers kunnen ook van verschillende aard zijn. Bij het bepalen van de informatie die bepaalde gebruikers moeten kunnen bekijken wordt al duidelijk dat verschillende types van gebruikers ook verschillende informatiebehoeften hebben.
Daarnaast kunnen de verschillende types gebruikers ook nood hebben aan andere representaties van dezelfde informatie. Dit wordt bepaald door de karakteristieken van de gebruikers.

In figuur 1.1 wordt een overzicht van de WSDM methode gegeven. De eerste stap is het bepalen van het “mission statement”. Deze stap moet het doel en het onderwerp van de web site aanduiden, samen met het doelpubliek.




Figuur 1.1: een overzicht van de stappen binnen WSDM.

Gebaseerd op dit “mission statement” wordt het modelleren van de gebruikersklassen uitgewerkt, en wel in twee stappen. Eerst is er de classificatie van de verschillende types gebruikers, daarna worden de karakteristieken van de klassen bepaald. Het resultaat van deze “audience modelling” is een verzameling gebruikersklassen samen met

een “informele” beschrijving van hun vereisten (zowel informatief, functioneel als gebruikersvereisten) en karakteristieken.
Soms kunnen “varianten” optreden indien leden van eenzelfde gebruikersklasse verschillende karakteristieken zouden hebben.
Daarna komt het conceptueel design van de site. Het opsplitsen van het conceptuele design en de effectieve implementatie heeft het voordeel dat de site kan ontworpen worden zonder als basis de huidige internet technologie nodig te hebben.

Het conceptueel design bestaat uit twee stappen: “information modelling” en “navigation design”. Bij het modelleren van de informatie worden “Audience Object Models” gecreëerd. Deze bevatten de informatievereisten voor de verschillende gebruikersklassen. De “Audience Object Models” worden bekomen door de informatievereisten te “vertalen” of om te zetten in “object chunks”. De verschillende “Audience Object Models” worden dan verwerkt tot één enkel objectmodel wat we het “Business Object Model” noemen. Dit Business Object Model bevat informatie dat beschikbaar is in de organisatie, onafhankelijk van één of ander gebruik.
Alle “Audience Object Models” worden dan beschreven als “views” op dit model, zodat alle redundantie gekend is en daarmee ook controleerbaar is.
In het “navigation design” bespreken we de structuur van de web site en modelleren we hoe de leden van de verschillende gebruikersklassen door de site zullen kunnen navigeren. Voor iedere gebruikersklasse wordt een navigatieschema gecreëerd. Ook kunnen deze tracks samengesteld worden door vereisten van de gebruikersklassen te vertalen in zogenaamde “navigation chunks”. Al deze navigatieschema ‘s vormen dan het “navigation model” van de site.
Tijdens de design van het implementatiegedeelte gaan we dan de “look en feel” van de web site gaan bepalen. Het doel is om een consistente en efficiënte “look en feel” te creëren, startend vanuit het conceptueel design en rekening houdend met de gebruikersvereisten en karakteristieken van de verschillende gebruikersklassen. Indien de informatie, die via de web site beschikbaar moet worden gesteld, zal onderhouden worden via een databank, moet het “implementation design” ook het logische design van de databank bevatten. Het logische databank schema kan uit het Business Object Model afgeleid worden.
In de laatste fase, de implementatie, wordt de effectieve implementatie van de web site uitgewerkt in een op voorhand gekozen web technologie.

In het volgend stuk zullen we de belangrijkste stappen in meer detail uitleggen en verduidelijken aan de hand van een klein voorbeeld.

## 1.2 Mission Statement

Dit is de eerste stap binnen de WSDM methode. De bedoeling is van een antwoord te krijgen op de volgende vragen: wat is het doel van de web site, wat is het onderwerp en welke zijn de gebruikers waarvoor deze site bedoeld is.

Zonder echt rekening te houden met een doel van de web site wordt het moeilijk om beslissingen te maken of de effectiviteit van de web site te bepalen.

De gebruikers naar wie we ons willen richten met de web site zijn diegenen die een interesse voor de site zouden kunnen hebben. Om deze gebruikers op een goede manier te kunnen benaderen, is het belangrijk om ze eerst goed te leren kennen of te begrijpen. Daarvoor is eerst een bepaling van die gebruikersklassen nodig.

Het onderwerp van de web site is dan gekoppeld aan het doel en de gebruikersklassen van de web site. Het onderwerp van de site moet er voor zorgen dat het doel bereikt wordt en dat dit op een geschikte manier voor de gebruikerklassen kan beschikbaar zijn.
Het onderwerp van de web site kan ook belangrijk zijn om te bepalen welke informatie al dan niet op de site beschikbaar zal moeten worden gesteld.

Als voorbeeld zal hier het “mission statement” bepaald worden van een web site voor een onderzoeksgroep van een universiteit. Het mission statement is:

- informatie over de huidige onderzoeksprojecten binnen de onderzoeksgroep op het web plaatsen om via deze weg sponsoring van bedrijven aan te trekken en studenten of onderzoekers aan te trekken.
- publicaties van geschreven werken via het web beschikbaar stellen om deze sneller toegankelijk te maken voor andere onderzoekers.

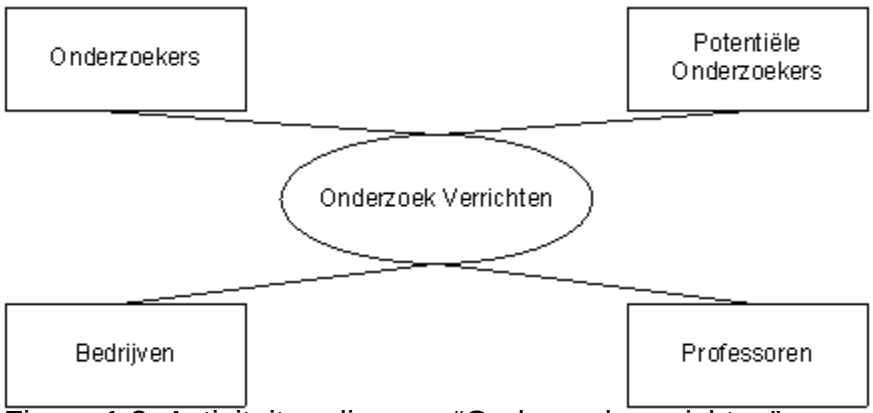
De gebruikersklassen in dit voorbeeld zijn potentiële onderzoekers, onderzoekers en bedrijven.
Het doel van de web site is de communicatie tussen de onderzoeksgroep en de bedrijven verbeteren (om sponsoring te krijgen) en nieuwe onderzoekers aantrekken.
Het onderwerp is onderzoek en publicaties.

## 1.3 Audience Modelling

Vermits het Mission Statement niet voldoende is om van de gebruikers alle nodige informatie te hebben, wordt deze informatie bepaald door het bekijken van de activiteiten van de organisatie waarvoor deze web site wordt opgebouwd.
Op die manier kan dan dikwijls beslist worden of bepaalde types van gebruikers in één gebruikersklasse of meerdere zullen gemodelleerd worden.
De activiteiten van de organisatie splitsen we stuk voor stuk op, om in iedere deelstap die we hierdoor verkrijgen te kunnen bepalen of we het doelpubliek uit het mission statement al dan niet moeten verfijnen.
Het is hierbij belangrijk om op te merken dat enkel de activiteiten van het bedrijf, die op de web site moeten weergegeven worden, van belang zijn, voor het bepalen van die verfijning.
In het voorbeeld van de onderzoeksgroep moeten we bijvoorbeeld geen rekening houden met de vakken die door proffen van de onderzoeksgroep worden gegeven.

Bij ieder van de activiteiten zullen er mensen betrokken zijn. Deze personen zijn potentiële gebruikers indien ze tot de oorspronkelijke doelgebruikers uit het Mission Statement behoren.

Als we in het voorbeeld de activiteit “onderzoek” bekijken, merken we dat daarin zowel onderzoekers, bedrijven, professoren als potentiële onderzoekers betrokken zijn.



Figuur 1.2: Activiteiten diagram “Onderzoek verrichten”.

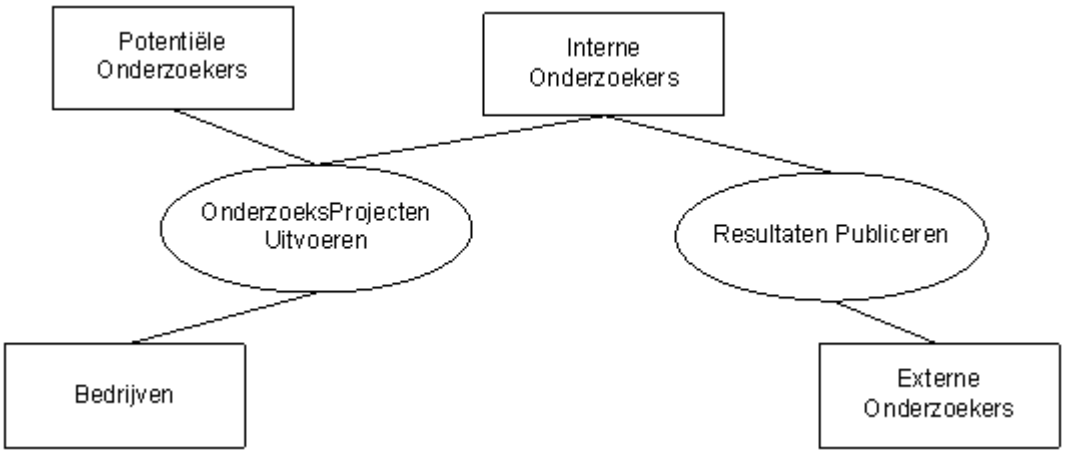
Uit het Mission Statement halen we echter dat professoren niet tot de originele doelgebruikers horen en dus verder ook niet meer in aanmerking komen.
Van de overige gebruikers (onderzoekers, potentiële onderzoekers en bedrijven) moet dan bepaald worden of ze al dan niet tot dezelfde gebruikersklasse kunnen behoren, en dit op basis van hun vereisten.

In het voorbeeld merken we dat de informatieve vereisten voldoende verschillend zijn om de doelgebruikers in aparte gebruikersklassen op te splitsen:

- potentiële onderzoekers: hebben interesse voor de lopende projecten om er verder te kunnen aan meewerken en op die manier de lopende vacatures in te vullen.
- bedrijven: zoeken algemene informatie over de projecten en zijn geïnteresseerd indien de resultaten ervan voor hen nut kunnen hebben.
- onderzoekers: zijn vooral geïnteresseerd in de standen van de projecten en de opgeleverde resultaten.

Als we de gebruikersklassen dan hebben, moeten we nog onderzoeken indien het nodig is om ze te verfijnen. Dit gebeurt door de activiteit, indien mogelijk, op te splitsen in subactiviteiten.

In het voorbeeld, waar we de activiteit “Onderzoek verrichten” hebben besproken, kunnen we deze activiteit onderverdelen in “Onderzoeksprojecten uitwerken” en “Resultaten publiceren”.




Figuur 1.3: Opsplitsing van het activiteiten diagram “Onderzoek verrichten”.

We zien hierbij dat we een onderscheid kunnen maken tussen “Interne Onderzoekers” en “Externe Onderzoekers”, vermits beide groepen verschillende informatieve behoeften hebben. We zullen dan ook de Gebruikersklasse “Onderzoekers” onderverdelen in “Interne Onderzoekers” en “Externe Onderzoekers”.

Bij ieder van de activiteiten die we hebben, controleren we dan op een gelijkaardige manier of er al dan niet subactiviteiten zijn en of deze dan ook subklassen opleveren.

In dit voorbeeld zijn er geen interessante subactiviteiten die verdere opsplitsingen van gebruikersklassen teweeg brengen.

Een samenvatting van de methode kan als volgt gegeven worden:

1. Bepaal al de activiteiten van de organisatie die verband houden met het Mission Statement.
2. Voor al deze activiteiten:
   2.1 identificeer al de mensen betrokken bij de activiteit
   2.2 hou enkel rekening met de mensen die oorspronkelijk in het Mission Statement bepaald zijn
   2.3 verdeel de personen in gebruikersklassen gebaseerd op verschillende functionele en informatieve vereisten
   2.4 splits, indien mogelijk, de activiteit in subactiviteiten
   2.5 herbegin vanaf stap 2 tot geen onderverdeling meer mogelijk is of geen nieuwe klassen gevonden worden.

Na het vastleggen van de gebruikersklassen door middel van de functionele en informatieve vereisten, moeten we bepalen of al de leden van een gebruikersklasse ook dezelfde gebruikerseisen hebben. Er kunnen bijvoorbeeld verschillen optreden doordat oudere mensen minder kennis hebben van het internet dan jonge gebruikers.

Indien er binnen een gebruikersklassen een onderscheid kan gemaakt worden tussen personen met verschillende karakteristieken, kan het nodig zijn dat “audience class variants” geïntroduceerd worden.

In het voorbeeld zijn er geen groepen binnen bepaalde gebruikersklassen die verschillende gebruikerseisen hebben, daarom is het niet nodig om in dit geval varianten te introduceren.

Enkele redenen waarom het wel nodig zou kunnen zijn om varianten in te lassen zijn de volgende:
- er is binnen een gebruikersklasse een groot verschil aan kennis van computergebruik en/of het internet.
- er zijn binnen een klasse taalverschillen (ene groep is bijvoorbeeld de moedertaal machtig, terwijl bij de anderen communicatie via het Engels nodig is).

## 1.4 Conceptual Design

Het doel van het conceptueel design is het omzetten van de vereisten (informatief -, functioneel - en gebruikers-) in een formele beschrijving op hoog niveau, die later kan gebruikt worden om efficiënte web sites te genereren.

Bij het conceptuele design is het ook de bedoeling om zich te concentreren op het conceptuele “wat en hoe”, en niet op het visuele “wat en hoe”.
Dit komt overeen met het conceptueel ontwerp die in “database design” wordt gegeven, namelijk “de beschrijving van de manier waarop de informatie gerepresenteerd wordt (objecten, object types, relaties)”.

Bij het database design wordt er echter geen rekening gehouden met de manier waarop door deze informatie zal genavigeerd worden. Bij WSDM is dit wel van essentieel belang (dit komt overeen met het conceptuele “hoe”).
We hebben dit nodig omdat we later op de web site door de informatie willen navigeren.

Zoals eerder reeds werd vermeld bestaat het conceptuele design uit twee stappen: “Information Modelling” en “Navigation Design”.
De eerste stap is vooral bedoeld voor data-intensieve web sites, maar kan ook van pas komen bij minder intensieve. Doorgaans is het ook zo dat web sites een mix zijn van beide.
Het doel van deze informatie modelleer stap is een model op te stellen voor de data die op de web site moet komen. Het grote voordeel hierbij is dat de data goed kan onderhouden worden, eventueel zelfs via een databank.

Een groot verschil met database design is dat we bij het modelleren van die informatie niet eindigen met één groot conceptueel schema, maar met een aantal verschillende schema’s.
Dit komt omdat we, vanuit de filosofie van WSDM, willen werken met een gebruikersgebaseerde benadering.
In deze benadering starten we met het modelleren van de informatie-vereisten voor de verschillende gebruikersklassen. Voor ieder van deze klassen stellen we dan een conceptueel schema op. We noemen dit het “Audience Object Model”.

In het “Audience Object Model” beschrijven we de verschillende “Object Types”, de relaties ertussen en de “rules” en “constraints” die er op van toepassing zijn. Dit komt dan wel weer volledig overeen met het “Database design”.

Het opstellen van een “Audience Object Model” gebeurt stapsgewijs door de informatievereisten van de gebruikersklassen te vertalen in zogenaamde “Object chunks”.
De volgende stappen worden ondernomen om het “Audience Object Model” voor één klasse te verkrijgen:

1. We houden enkel rekening met de informatievereisten.
2. Enkel de informele informatievereisten worden uitgewerkt.

3. We splitsen deze vereisten op in zogenaamde “elementaire informatievereisten”.
4. Voor ieder van deze “elementaire informatievereisten” stellen we een object model op, de “object chunk”, met één van de bestaande informatie modelleer technieken.
5. Al de object chunks van een gebruikersklasse worden tot één object model samengevoegd, het “Audience Object Model”.

Indien we terug ons voorbeeld van de onderzoeksgroep nemen, waarbij we de gebruikersklasse bedrijven als doelklasse gebruiken. Eén van de vereisten van de gebruikersklasse is de volgende:

*“Algemene informatie over projecten en onderzoekers weergeven”.*

Indien we deze vereiste verder uitdiepen, dan merken we dat we volgende informatie elementen nodig hebben om de vereiste volledig te kunnen uitvoeren:

*“Voor ieder project hebben we de volgende informatie nodig: projectnaam, projectstatus, onderzoeksgroep, projectbeschrijving, en voor iedere onderzoeker die aan het project meewerkt: naam, ID, zijn bureau, zijn telefoonnummer en zijn e-mail adres”.*

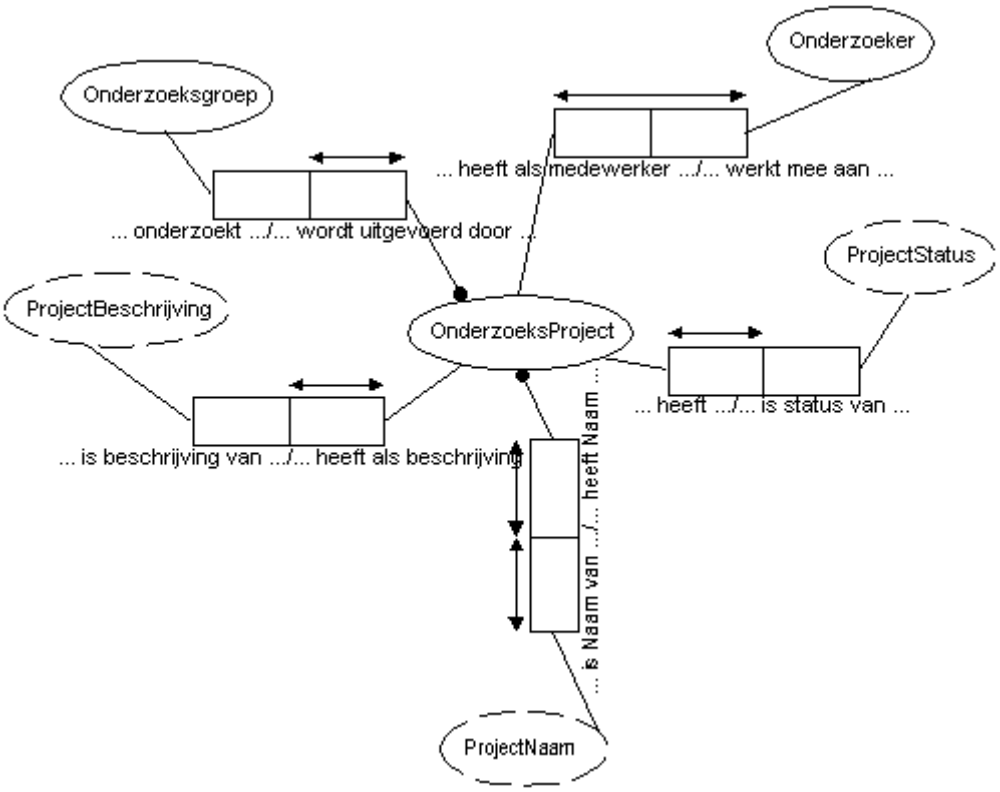
Deze vereiste kan opgesplitst worden in twee vereisten zoals hieronder is weergegeven:

1. *“Voor ieder project hebben we de volgende informatie nodig: projectnaam, projectstatus, onderzoeksgroep, projectbeschrijving, de onderzoekers die aan het project meewerken”.*

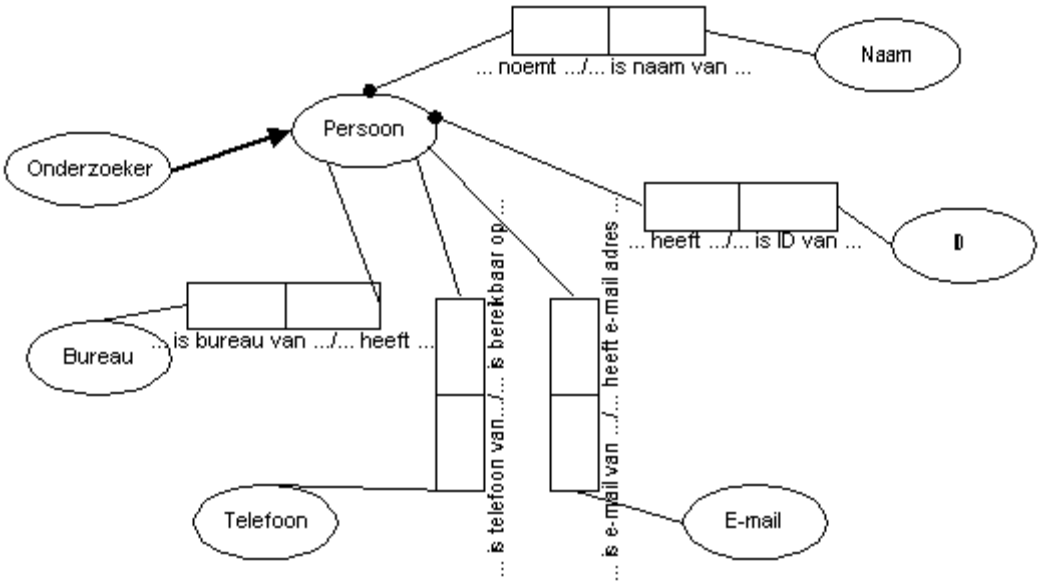
2. *“Voor iedere onderzoeker die aan een project meewerkt: naam, ID, zijn bureau, zijn telefoonnummer en zijn e-mail adres”.*

Het volgende wat we creëren is een “object chunk” waarin al de informatie elementen per vereiste worden weergegeven. De “object chunks” kunnen gezien worden als een deel van een groter “object model”, wat dan het “Audience Object Model” moet worden.

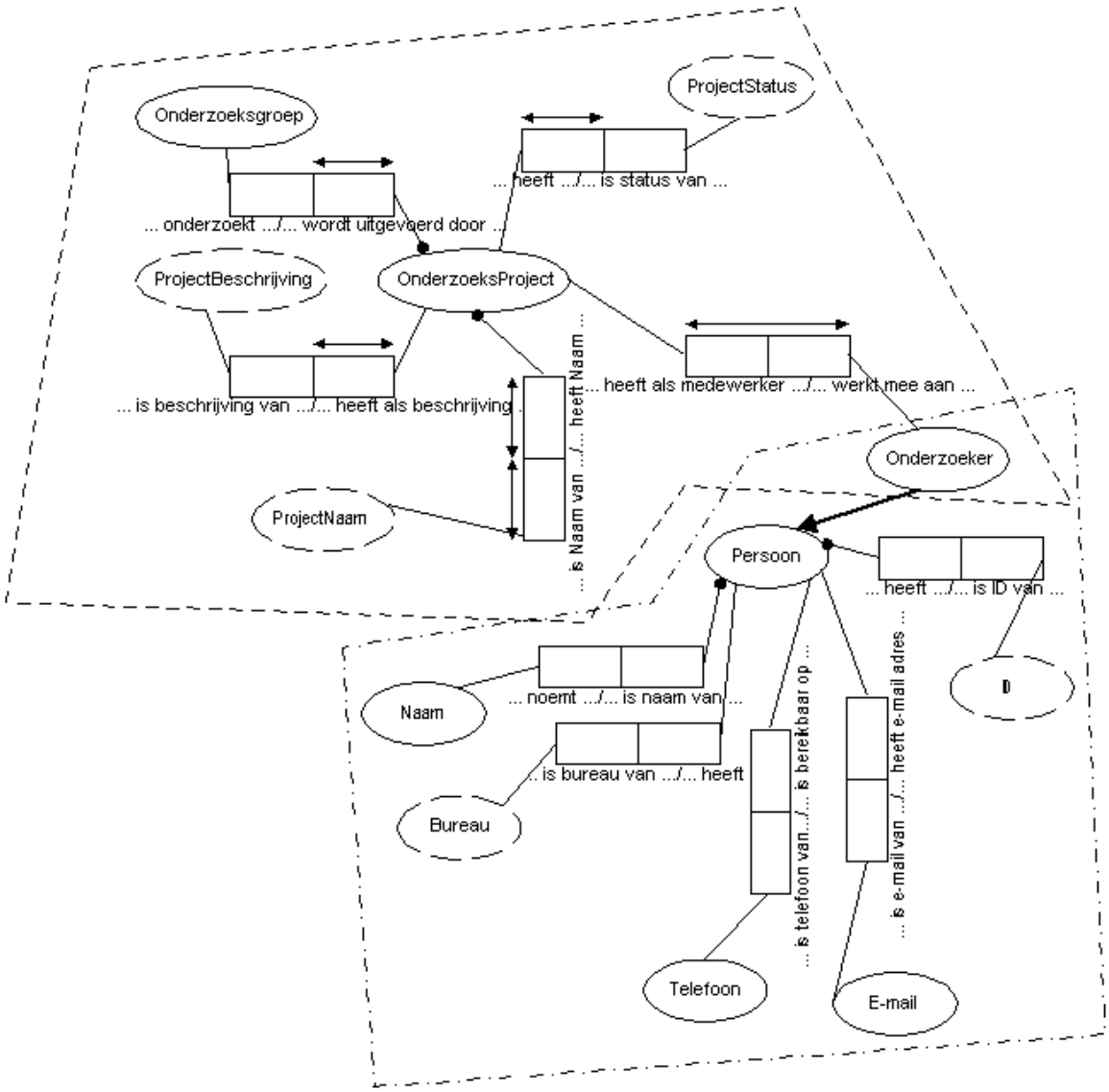
De “object chunks” worden ook allemaal gemodelleerd volgens éénzelfde modelleertechniek, in het voorbeeld hier is dit ORM [Hal 99].
In figuren 1.4 en 1.5 zijn de “object chunks” voor vereisten 1 en 2 afgebeeld, in figuur 1.6 wordt dan een deel van het “Audience Object Model” weergegeven door de twee “object chunks” samen te voegen.




Figuur 1.4: de object chunk voor een onderzoeksproject.



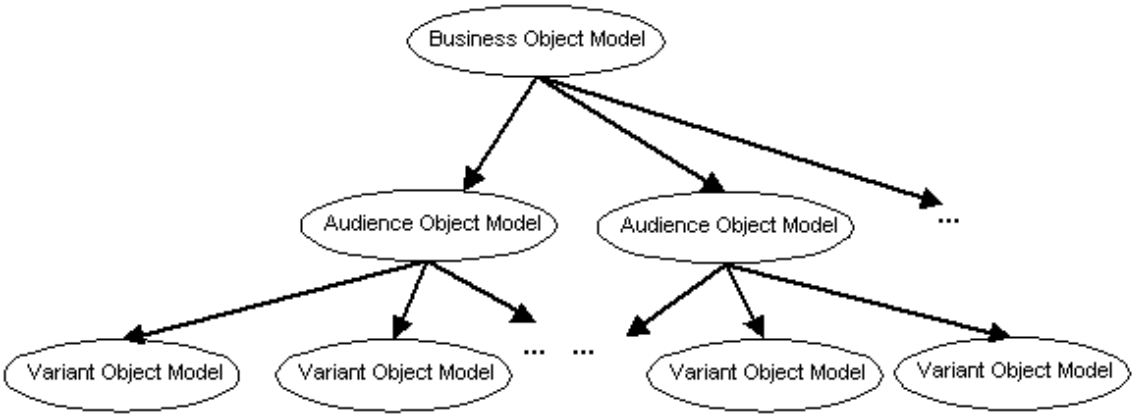

Figuur 1.5: de object chunk voor een persoon.




Figuur 1.6: een deel van het “Audience Object Model”, door de twee eerste “object chunks” samen te voegen.

Na het bepalen van de “Audience Object Models” brengen we deze modellen in relatie met elkaar door ze samen te voegen in één algemeen model: het “Business Object Model” (zie figuur 1.7). De reden waarom we dit doen is de volgende: op die manier kunnen we zien dat bepaalde gebruikersklassen dezelfde informatie nodig hebben, zelfs al is dit op een andere visuele manier weergegeven. Doordat verschillende gebruikersklassen dezelfde informatie nodig hebben, weten we ook dat deze informatie redundant zal zijn. Door dit echter te weten is de redundantie controleerbaar, door ze bijvoorbeeld op een centrale plaats te beheren.
Het “Business Object Model” beschrijft de informatie, die we nodig hebben bij de verschillende gebruikersklassen, op een onafhankelijke manier.

Daarna kunnen we de verschillende “Audience Object Models” beschrijven als views op het algemene “Business Object Model”.




Figuur 1.7: Relaties tussen verschillende types van “Object Models” in WSDM.

De tweede stap van het conceptuele design, het “Navigation Design”, beschrijft dan de structuur van de web site en de wijze waarop de leden van de verschillende gebruikersklassen door de web site zullen kunnen navigeren.
Net zoals bij de vorige stap, het “Information Modelling”, wordt ook hier voor iedere gebruikersklasse een verschillende “Navigation Track” opgesteld.
Al de navigatieschema ‘s zullen dan samen het “Navigation Model” vormen.

Een navigatieschema wordt beschreven in termen van componenten en links.
Componenten stellen informatie eenheden voor. Deze eenheden kunnen zowel tekst, figuren, andere multimediale informatie zijn als gestructureerde informatie gemodelleerd in de “Audience Object Models”.
De verschillende componenten kunnen dan met elkaar verbonden worden door middel van links. Deze links kunnen gebruikt worden om de structuur van de web site te modelleren, maar ook om de nood aan navigatielinks op een bepaalde plaats uit te drukken.

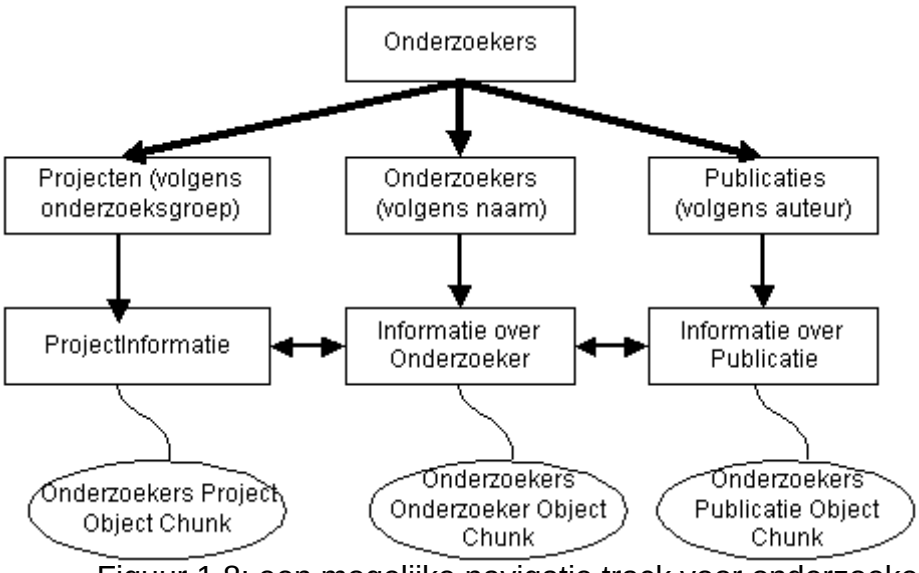
De structuur die we verkrijgen bij het navigatiemodel, door de beschrijving van componenten en links, is van conceptueel niveau. Bij het omzetten naar de web pagina’s zelf (in een latere fase) kan het zijn dat niet alle links identiek zullen zijn of dat een component juist op één pagina zal geplaatst worden. Het kan voorkomen dat een component over verschillende pagina’s verspreid zal worden, of dat enkele componenten samen op één pagina zullen weergegeven worden.

Het opbouwen van een navigatieschema is gelijkaardig met het creëren van een “Audience Object Model”. Ook hier wordt enkel rekening gehouden met de relevante vereisten, die opgedeeld worden in elementaire delen.
Voor ieder van de verkregen elementaire vereisten stellen we een “navigation chunk” op. Deze chunks combineren we achteraf tot een navigatieschema voor iedere gebruikersklasse.
Om de navigatieschema ‘s op een unieke wijze te kunnen identificeren starten we met een unieke component aan de top van het schema.

De informatievereisten zetten we dus om in componenten en de navigatievereisten in links.
Indien een component overeenstemt met gestructureerde data, dan is dit geassocieerd met de “object chunk” dat deze data modelleert.
Indien alle informatievereisten van een gebruikersklasse besproken werden, dan is alle informatie, gemodelleerd in het “Audience Object Model”, ook uitgewerkt in componenten.

Bij een navigatie track moet ook duidelijk zijn dat alle informatie bereikbaar moet zijn vanaf de top van deze track, anders zitten er fouten in. Ook bestaat de mogelijkheid dat bepaalde componenten bereikbaar zijn vanuit verschillende plaatsen in de navigatie track.

Figuur 1.8 toont een deel van de navigatie track voor de onderzoekers uit de onderzoeksgroep van ons voorbeeld.

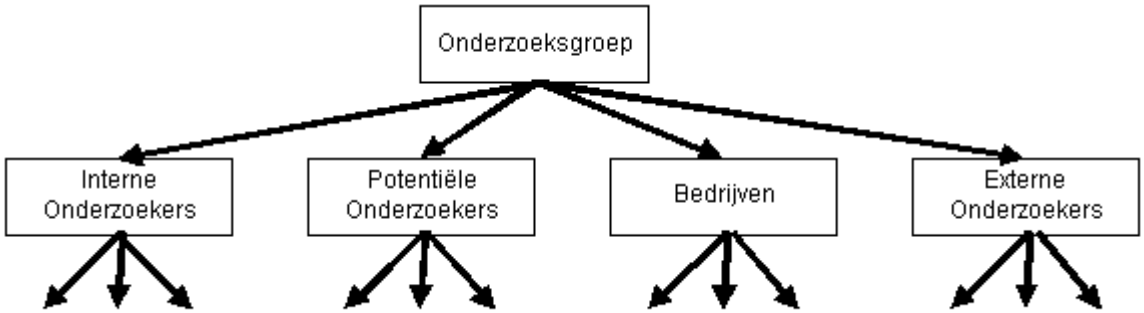


Figuur 1.8: een mogelijke navigatie track voor onderzoekers.

Om dan, allerlaatst in deze fase, het volledige navigatiemodel te verkrijgen, moet er een nieuwe topcomponent gecreëerd worden. Vanuit deze component wordt dan een verbinding gemaakt met alle andere

topcomponenten. De nieuwe component bevat de nodige informatie zodat de gebruiker onmiddellijk kan uitmaken welke navigatie track voor hem het interessantste is.

In figuur 1.9 wordt het navigatie model voor de onderzoeksgroep weergegeven.




Figuur 1.9: de verschillende navigatie tracks worden samengevoegd tot één model.

## 1.5 Implementation Design en Implementation

Bij het ontwerpen van de implementatie bepalen we essentieel de structuur van de pagina's van de web site, de "look en feel" van de site en optioneel ook het logische databankschema voor een databank dat de web site kan gebruiken.

De paginastructuur wordt afgeleid van het navigatiemodel. Voor de componenten en links in het navigatiemodel definiëren we dan de pagina's en hyperlinks.

De hoeveelheid informatie op een pagina mag de gebruiker niet overweldigen. Daarbij is het ook van belang dat het openen of downloaden van een pagina niet te veel tijd in beslag neemt, maar er mag ook niet te weinig informatie op iedere pagina staan, anders zal de gebruiker te veel moeten navigeren.
De ontwerper van de web site kan beslissen om bepaalde componenten samen te brengen zodat meer informatie op één pagina zou staan, of juist omgekeerd een component op te splitsen over meerdere pagina's.

Bij het design van de "look en feel" moet er vooral rekening gehouden worden met de karakteristieken van de gebruikersklassen. Dit kan dan resulteren in verschillende stijlen voor verschillende "audience tracks".

Om het onderhoud te verbeteren maken web sites meer en meer gebruik van databanken om hun informatie bij te houden.

De basis voor de te gebruiken databank kan gevonden worden in het “Business Object Model”. Dit model is in principe het conceptueel schema voor de databank, waaruit een logisch schema kan gegenereerd worden.
De queries die nodig zijn om de informatie uit de databank te halen kunnen dan afgeleid worden uit de “Audience Object Models”, die opgesteld werden als views op dit “Business Object Model”. Op die manier kunnen de pagina’s dan automatisch opgesteld worden door de juiste informatie uit de databank te halen.

Het is natuurlijk ook mogelijk dat het “Implementation Design” afhankelijk is van de gekozen omgeving waarin de web site zal geïmplementeerd worden. Daardoor kan het nodig zijn om beperkingen op te design te leggen.

De laatste fase is dan de implementatie zelf van de web site in de gekozen implementatie omgeving. Daar wordt verder weinig aandacht aan besteed vermits deze totaal afhankelijk is van de ontwikkelingsomgeving zelf.

## 1.6 Problemen of tekortkomingen

Nu volgen nog enkele tekortkomingen van WSDM, waaronder deze die ik zal proberen op te lossen en enkele andere die ondertussen reeds onder handen worden genomen.

- De definitie van gebruikersklassen en hun varianten is niet altijd duidelijk afgelijnd en lijkt in sommige gevallen te wazig.

- Het modelleren van de informatie in “Information Modelling” heeft tot gevolg dat het mogelijk is om in een latere fase te beslissen of de web site al dan niet gebruik zal maken van een databank om de informatie uit te halen.
  Indien de web site er wel gebruik van zal maken, dan kan uit het model gemakkelijk een logisch schema voor de databank afgeleid worden.
  Ook indien de ontwerper beslist om geen gebruik van databanken te maken, kan het verkregen model van het “Information Modelling” handig gebruikt worden om de structuur van de informatie bij te houden en te gebruiken bij het opbouwen van de navigatieschema ‘s.

  Wat echter niet aanwezig is, is een model om via de web site informatie toe te voegen of te wijzigen aan de databank. Dit komt dan neer op het implementeren van “application web sites”, wat niet het allereerste doel van WSDM was, maar

inmiddels toch onmisbaar indien WSDM zich wil blijven profileren tussen de andere methoden.

Het is namelijk zo dat meer en meer bedrijven al hun informatie op de web site ook via de web site willen onderhouden.

Over deze tekortkoming zal ik het hoofdzakelijk hebben in deze thesis.

- Op gebied van Navigation Design is er ook geen duidelijke beschrijving die aangeeft hoe de gebruiker te werk moet gaan. Er wordt enkel aangegeven dat voor iedere gebruikersklasse een navigatieschema moet worden opgemaakt en waarbij de verschillende schema 's op het einde worden samengevoegd.
  Er is verder niet aangegeven welke symbolen de ontwerper best gebruikt, zodat er bij het ontwerpen van een web site met meerdere personen onduidelijkheden of conflicten kunnen ontstaan.

- Implementation design.
  Doordat de gebruikersaandacht een grote plaats inneemt bij het ontwerpen van een web site, is het ook vrij belangrijk om bij de design van de implementatie veel rekening te houden met die gebruikersaandacht en de bijhorende interfaces.
  Dit concept is echter niet zo eenvoudig als het lijkt, vermits het moeilijk is, misschien zelfs bijna onmogelijk, om goeie en algemene gebruikersinterfaces te definiëren.

  Die gebruikersinterfaces hebben namelijk een grote diversiteit door de vele mogelijkheden die de ontwerper ermee kan aanvangen. De interfaces zijn verder heel erg afhankelijk van de situatie waarin men zich bevindt en van de gebruikersklassen van de te ontwerpen web site.

  Misschien is het in zo een geval beter om zich niet bezig te houden met het bepalen van een goede definitie van een interface (of meerdere interfaces) voor WSDM, maar zich eerder te richten op richtlijnen die interessant kunnen zijn voor de ontwerper.
  Deze richtlijnen kunnen dan positieve of negatieve adviezen zijn over het gebruik van bepaalde kleuren, fonts, …. Ook bepaalde richtlijnen waar de ontwerper op moeten letten tijdens het opstellen van de design.

- Ook bij de implementatie kunnen we moeilijk gaan bepalen hoe de ontwerper te werk moet gaan, omdat het onmogelijk is op

voorhand te weten welke technieken gebruikt zullen worden tijdens de implementatie.
Zelfs indien dit in de methode zou worden ingewerkt, blijft dit onmogelijk omdat er steeds nieuwe applicaties, technieken, … blijven bijkomen.

Maar misschien kunnen ook hier een aantal richtlijnen voor de ontwerper worden meegegeven.

# Hoofdstuk 2:

# Het modelleren van applicatie web sites

De problematiek, die in het vorige hoofdstuk reeds kort werd gesitueerd, is vooral gericht naar data intensieve applicatie web sites.

In het beginstadium van het Internet, of toch bij de start van de populariteit ervan, werd dit probleem nog niet zozeer opgemerkt, omdat het toen nog niet mogelijk was web sites aan applicaties of databanken te koppelen.
De enige web sites die toen beschikbaar waren, bevatten enkel web pagina's met informatie. Dit waren HTML-pagina's. Bij iedere wijziging van die informatie was de persoon, die instond voor het onderhoud van de web site, verplicht alle pagina's waarin de informatie moest gewijzigd worden van de server te halen, de informatie effectief wijzigen, en de nieuwe pagina's terug op de server plaatsen.

Naargelang de technologie echter evolueerde, en de ontwerper van web sites meer mogelijkheden kreeg, konden de mogelijkheden binnen de web pagina's ook uitgebreid worden.

Om de ontwerper van web sites de kans te geven een goede structuur voor hun web site te maken, met daarbij nog een aantal andere interessante mogelijkheden, werden een aantal methoden uitgewerkt.
Die methoden ontstonden om in te gaan tegen het enorme aantal web sites die toen beschikbaar waren, en waar de gebruiker niks aan had omdat ze totaal ongestructureerd waren en daardoor enorm gebruiksonvriendelijk.

De meeste van die methoden bieden een manier aan om de informatie, die op de web site moet gebracht worden, op een degelijke wijze te structureren.
Daarnaast wordt er bij alle methoden ook een uitleg verschaft over de design van het navigatiegedeelte, waarbij het, afhankelijk van de filosofie, eerder gericht is op de informatie of op de gebruikers.
Ook op gebied van implementatie en/of design zijn er enkele methoden die een techniek hebben uitgewerkt. Die techniek bestaat dan in de meeste gevallen enkel uit een aantal te volgen richtlijnen.
Bij het overgrote deel van de methoden blijft een voorstel voor implementatie achterwege, voornamelijk door het feit dat het aantal

ontwikkelingsmogelijkheden voor een web site te groot is en de technieken te snel evolueren.

Wat de meeste methoden echter nog onderkennen, is het gebruik van databanken voor hun web sites. Vooral het gebruik van databanken is bij data intensieve web sites heel handig, omdat meer en meer bedrijven hun informatie via het web willen gebruiken, en niet alleen voor het verspreiden van hun informatie, maar ook het onderhoud ervan (zoals toevoegen, wijzigen en verwijderen).

Het is vooral de bedoeling om dus alle databank acties via een web site te laten gebeuren, zodat de gebruikers geen andere applicaties nodig hebben om de data toe te voegen of te wijzigen.
Het uitwerken van deze acties via het web moet natuurlijk in het implementatiegedeelte gebeuren, maar het zou voor de ontwerper van de web site nuttig zijn om de verschillende acties reeds op voorhand te kunnen modelleren.
Dit zou dan kunnen overeenkomen met het modelleren van de informatie die op de web site zal geplaatst worden.

Wat we dus moeten kunnen modelleren is de informatie die de gebruiker via de web site zou kunnen invoeren, wijzigen en verwijderen en hoe dit moet gebeuren.
Deze modellen kunnen gelijkaardig zijn met de huidige “Audience Object Models” uit WSDM, en zijn eveneens heel belangrijk om in een later design proces (van de implementatie) op een eenvoudige wijze interfaces of forms te creëren waarmee data kan gemanipuleerd worden, net zoals de “Audience Object Models” belangrijk zijn om later de juiste informatie weer te geven.

Ook moet de ontwerper in het model duidelijk kunnen maken welk type gebruikers (uit de gebruikersklasse) bepaalde acties op bepaalde delen informatie al dan niet mogen uitvoeren.
Ook dit komt overeen met de “Audience Object Models”, waarin aangegeven wordt welke informatie voor welk type gebruiker belangrijk is.
Hier kan dan aangegeven worden welke gebruikers het recht zouden krijgen om bepaalde stukken informatie te wijzigen, toe te voegen, te verwijderen, …

In een volgend hoofdstuk wordt uitgezocht in hoeverre de andere methoden dit probleem reeds hebben opgelost. Daarbij zal dan ook aangegeven worden waarom deze oplossingen niet (direct) bruikbaar zijn binnen WSDM.
In het daaropvolgende hoofdstuk wordt dan een oplossing voorgesteld voor dit probleem.

# Hoofdstuk 3:

# Methoden voor het modelleren van applicatie web sites

Er zijn in het nabije verleden reeds een aantal web site methoden beschikbaar gekomen. Door middel van een korte bespreking van de belangrijkste methoden zal nagegaan worden in hoeverre zij geschikt zijn voor het ontwerpen van applicatie web sites.
Bij de methoden die een oplossing hebben zal ook gecontroleerd worden in hoeverre we deze oplossing binnen WSDM kunnen integreren.
Indien dit niet mogelijk is zullen we aangeven wat de belangrijkste redenen daarvoor zijn.
Zoals duidelijk zal zijn, zijn sommige van de besproken methoden gebaseerd op voorgaande. Bij de nieuwere methoden worden enkel de wijzigingen besproken.

## 3.1 Hypertext Design Model (HDM)

HDM ([GPS 93]) was één van de eerste methoden die werd uitgewerkt om een gestructureerd model te brengen waarmee “hypertext” ontwikkeld kon worden. HDM had toen een aantal vernieuwende kenmerken zoals: de notie van perspectief; de identificatie van verschillende categorieën van links; het verschil tussen “hyperbase” en toegangsstructuren,…

HDM maakt gebruik van een applicatieschema, die algemene klassen van informatie elementen beschrijft in termen van hun karakteristieken. De belangrijke elementen binnen die karakteristieken waarmee we rekening moeten houden zijn de interne structuur en de types van connecties tussen de verschillende informatie elementen.

HDM is vooral een modelleermethode maar kan evengoed gebruikt worden om implementaties van hypertekst applicaties te genereren.
HDM heeft enkele bijzonderheden waardoor het verschilt van andere modellen, maar gebruikt tevens ook een aantal kenmerken van bestaande modelleertechnieken (vooral uit de databasewereld).

Een HDM-applicatie bestaat uit het structureren van informatie “chunks” die enititeiten genoemd worden. Een entiteit moet dan een fysisch of conceptueel object uit de reële wereld van het applicatiedomein weergeven.

De entiteiten kunnen ook ingedeeld worden in entiteittypes. De definitie voor een entiteit die hier gehanteerd wordt is: het kleinste “autonome” stukje informatie dat aan een applicatie kan toegevoegd of verwijderd worden. Met een “autonoom” stukje informatie wordt hier bedoeld dat zijn aanwezigheid of bestaan niet afhankelijk is van de aanwezigheid van andere informatie.

Een entiteit is opgebouwd uit een hiërarchie van componenten. Componenten bestaan op hun beurt uit eenheden. Iedere eenheid geeft de inhoud van een component weer gezien vanuit een bepaald perspectief.
Eenheden zijn de kleinste informatie “chunks” die kunnen gevisualiseerd worden binnen een HDM-applicatie. Daardoor zijn ze heel gemeenschappelijk met de “nodes” uit de standaard “hypertext”.

De informatiestructuren kunnen binnen HDM verbonden worden door middel van links. Er zijn drie soorten links: Structurele links verbinden componenten binnen éénzelfde entiteit; de zogenaamde “Perspectieve” links verbinden de verschillende eenheden die bij dezelfde component horen: als laatste de applicatielinks die de domeinafhankelijke relaties weergeven door componenten en entiteiten te verbinden, of zij nu al dan niet van hetzelfde entiteittype zijn. De applicatielinks kunnen we ook onderverdelen in linktypes.

Binnen HDM wordt er ook een onderscheid gemaakt tussen de noties “schema” en “instantie van een schema”. Een schema is een collectie van typedefinities die een applicatie op globaal niveau beschrijven. Een “instantie van een schema” is een collectie van entiteiten, componenten, eenheden en links die aan de definitie van een schema voldoen.
“Outlines” zijn daarbij toegangsstructuren die de gebruikers toelaten om onmiddellijk toegang te krijgen tot de informatiestructuren van een bepaalde instantie.
Als laatste is er de “browsing” semantiek die als doel heeft het specificeren van de visualisatie van de informatiestructuren (naar de gebruiker toe), en hoe de gebruiker door de informatie zal kunnen navigeren.

### 3.1.1 Entiteit en entiteittypes

Een entiteit is een relatief grote structuur van informatie die een object binnen een applicatiedomein representeert.
Entiteiten kunnen gegroepeerd worden in types, die overeenkomen met objectklassen binnen dit applicatiedomein.

### 3.1.2 Componenten

Een entiteit is een verzameling van componenten, op een hiërarchische wijze geordend. Een component is een abstractie voor een verzameling eenheden, die de eigenlijke informatie bijhouden.
Componenten nemen het type over van de entiteit waartoe ze behoren en kunnen dan ook enkel bestaan als onderdeel van een entiteit.

### 3.1.3 Perspectief

In “hypertext” applicaties moeten bepaalde stukken informatie soms in verschillende views weergegeven worden. In HDM wordt dit probleem opgevangen door het concept van verschillende perspectieven voor dezelfde informatie te creëren.
Indien een entiteit twee verschillende perspectieven heeft, dan hebben de componenten die bij die entiteit horen ook twee perspectieven.

De verzameling perspectieven wordt door alle entiteiten van een bepaald type gedeeld. Daarbij wordt die verzameling dan ook gedefinieerd op entiteittype niveau.
De notie van perspectieven wordt enkel gebruikt om consistentie te creëren binnen de verschillende views waarin de informatie (van de entiteiten van eenzelfde type) gepresenteerd wordt.

Verder is er achter het begrip perspectief geen semantiek verscholen. De gebruiker van HDM moet zelf bepalen hoe en wanneer hij dit wil gebruiken, en wat de betekenis ervan is.

### 3.1.4 Eenheid

Een eenheid komt overeen met een component dat geassocieerd is met een bepaald perspectief. Een eenheid bevat een naam en een body. De body bevat de effectieve informatie van de eenheid.

### 3.1.5 Links

Zoals eerder vermeld zijn er binnen HDM drie soorten links gedefinieerd: perspectieve-, structurele- en applicatielinks.

- Perspectief

  Perspectieve links verbinden eenheden die bij eenzelfde component horen. Deze links zijn voor de lezer goed begrijpbaar vermits het navigeren via deze links ervoor zorgt dat het onderwerp ongewijzigd blijft.

- Structuur

  Structurele links verbinden dan op hun beurt componenten die bij dezelfde entiteit gegroepeerd worden. Ze kunnen gebruikt worden om te navigeren tussen verschillende informatie “chunks” die bij dezelfde entiteit horen.

- Applicatie

  Deze links representeren de domeinafhankelijke relaties tussen entiteiten of hun componenten.

  Applicatielinks worden over het algemeen onderverdeeld in types (bij de vorige was dit niet van toepassing). Deze types worden in HDM gespecificeerd door een naam, een verzameling van bron en doel entiteittypes en een "symmetrisch attribuut”. Dit laatste attribuut moet bepalen of de link als dan niet een “inverse” link nodig heeft.

### 3.1.6 HDM Schema

Een HDM schema specificeert een verzameling van entiteittypes en linktypes. Instanties van een schema worden toegelaten tot de applicatie indien ze aan alle constraints en regels van het opgestelde schema voldoen.

### 3.1.7 Outlines

Voordat er in het algemeen kan gestart worden met navigeren binnen een applicatie moeten er natuurlijk toegangsmogelijkheden zijn. “Outlines” worden hier gedefinieerd als toegangsstructuren met als doel de gebruiker een startpunt te laten selecteren van waaruit hij verder zal kunnen navigeren.

### 3.1.8 Besluit

Zoals duidelijk is uit deze korte bespreking is er bij HDM geen sprake van een stapsgewijs proces om web sites op te bouwen.
Het was in de beginfase van de methoden ook nog niet duidelijk welke weg het internet zou volgen, en welke evoluties er nog zouden plaatsvinden. Daarom was het enkel de bedoeling om een goed schema op te bouwen waaruit dan later een web site kon ontwikkeld worden.

Er is dus bij HDM zeker nog geen sprake van het modelleren van informatie voor applicatie web sites, vermits dit type web sites, bij de ontwikkeling van HDM, nog volop in de kinderschoenen stond.
Deze methode is dan ook te verouderd om verder nog interessante informatie te bieden voor het modelleren van applicatie web sites.

In het volgende stuk wordt een bespreking gegeven van OOHDM, de opvolger van HDM, die wel een stapsgewijze methode voorstelt.

## 3.2 Object-Oriented Hypertext Design Model (OOHDM)

OOHDM ([SRB 96]) is net als zijn voorganger HDM een methode om web applicaties op te bouwen, gebaseerd op een model.

OOHDM bestaat echter uit een stapsgewijze werkwijze. De stappen zijn in deze methode: conceptueel design, navigatiedesign, “abstract interface design” en implementatie.
De uitwerking van deze stappen gebeurt incrementeel, iteratief en met een ontwikkelingsstijl die gebaseerd is op het gebruik van prototypes.
Gedurende iedere stap, met uitzondering van de implementatiestap, wordt een verzameling objectgeoriënteerde modellen opgebouwd die elk een stuk van de design beschrijven.

### 3.2.1 Conceptueel Design

Gedurende deze eerste stap wordt een model van het applicatiedomein opgebouwd volgens een objectgericht modelleer principe, waarbij enkele primitieven zoals attribuutperspectieven en subsystemen aanwezig zijn.

De conceptuele klassen kunnen uitgewerkt worden door middel van aggregatie en generalisatie- of specificatiehiërarchieën zoals gekend bij objectgerichte design.
De bedoeling is ook om hierbij de informatie van het applicatiedomein zo neutraal mogelijk te houden, waarbij er dus zo weinig mogelijk rekening mag gehouden worden met mogelijke gebruikers of andere externe invloeden.

Het resultaat van deze stap is een klasse en instantieschema dat bestaat uit subsystemen, klassen en relaties.

### 3.2.2 Navigatie Design

Om een applicatie bruikbaar te maken voor de doelgebruikers kan het nodig zijn om de informatie uit het conceptueel model te reorganiseren. Dit gebeurt bij OOHDM in het navigatiemodel, dat in feite een view is op het conceptueel model.
Het is de bedoeling om in dit navigatiemodel rekening te houden met de types van gebruikers en de taken die deze gebruikers zullen moeten kunnen uitvoeren. Op die manier kunnen er verschillende

navigatiemodellen gebouwd worden uit één conceptueel schema, die elk voor andere types van gebruikers of voor verschillende taken verantwoordelijk zijn.

Het navigatiedesign wordt uitgewerkt in twee schema's: het navigatie-klasse schema en het navigatiecontekst schema.

Het navigatieklasseschema maakt gebruik van knopen, links en toegangs-structuren zoals dit het geval is bij algemene "hypermedia" applicaties.
Dit schema reflecteert de gekozen "view" op het applicatiedomein.

De knopen worden gedefinieerd als objectgerichte views van conceptuele klassen, die tijdens het conceptuele design opgesteld werden.
De links reflecteren de relaties en worden ook gedefinieerd als views op de relaties uit het conceptuele schema;

De ontwerper moet ook rekening houden met de wijze waarop de gebruiker door de informatie zal navigeren om ervoor te zorgen dat er geen redundante informatie voorkomt in de applicatie.

In OOHDM wordt de navigatieruimte gestructureerd door gebruik te maken van de notie van navigatiecontekst. Deze contekst is een verzameling knopen, links, contekstklassen en andere navigatieconteksten.

### 3.2.3 Abstract Interface Design

Om de applicatie toegankelijk te maken voor de gebruiker moet er een interface zijn. Dit is dan ook de bedoeling van deze stap.
Het opsplitsen van het navigatiemodel en het interfacemodel laat toe om een hogere onafhankelijkheid te creëren van de user interface technologie die later kan gebruikt worden tijdens te implementatie.
In OOHDM maakt men gebruik van "Abstract Data View (ADV)" om de interfaces voor de gebruikers te beschrijven.

### 3.2.4 Implementatie

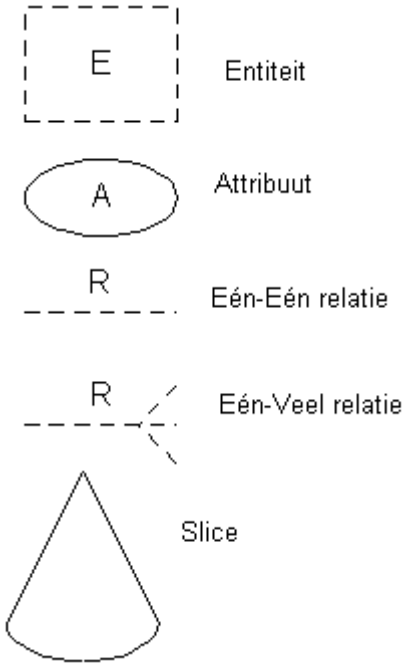
Om een werkende implementatie te bekomen, moet de ontwerper de navigatie en interface modellen omzetten in concrete objecten uit de gekozen implementatieomgeving. Deze omgeving kan dan één van de vele hypermedia platformen zijn.
De objectgerichte stijl van de abstracte interface specificatie laat immers toe om de interface van de implementatie in verschillende platformen te realiseren.

### 3.2.5 Besluit

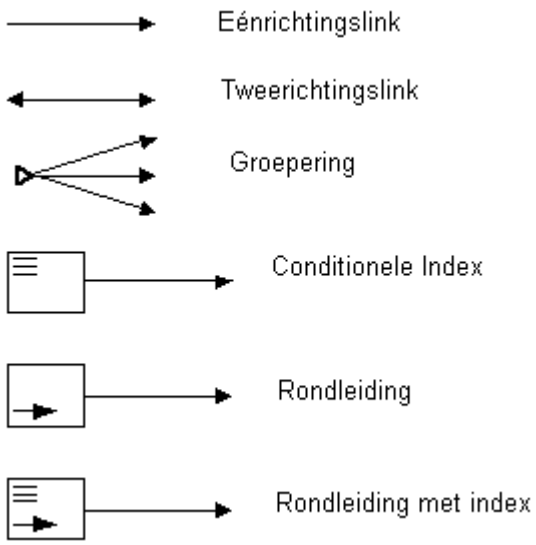
Ook het vervolg van HDM stelt slechts een design op van de informatie die in de webapplicatie moet worden weergegeven.
OOHDM bevat evenwel een aantal stappen, waarbij nieuwe technologieën of mogelijkheden binnen het internet nog niet ter sprake worden gebracht.
Er wordt binnen dezen methode ook slechts in geringe mate gesproken over de belangrijkheid van de gebruikers op het web.

## 3.3 Relationship Management Methodology (RMM)

RMM ([ISB 95]) maakt gebruik van het “Relationship Management Data Model (RMDM)” om de informatie die tot de web applicatie moet behoren te modelleren.
RMDM is een datamodel met een aantal logische objecten, die gebruikt worden om een abstractie te maken van een deel van de reële wereld. In dit geval het deel dat tot het applicatiedomein behoort. RMDM beschrijft echter ook, naast de datamodellering die in de meeste “klassieke” modelleertechnieken voorkomt, de navigatiemechanismen binnen de hypermedia applicaties.




Figuur 3.1: een eerste deel van de notaties uit het “Relationship Management Data Model (RMDM)”. *De notaties in deze figuur zijn de E-R domein notaties en de slice notatie.*




Figuur 3.2: de “access” notaties die gebruikt worden binnen RMDM om navigatiemodellen op te stellen.

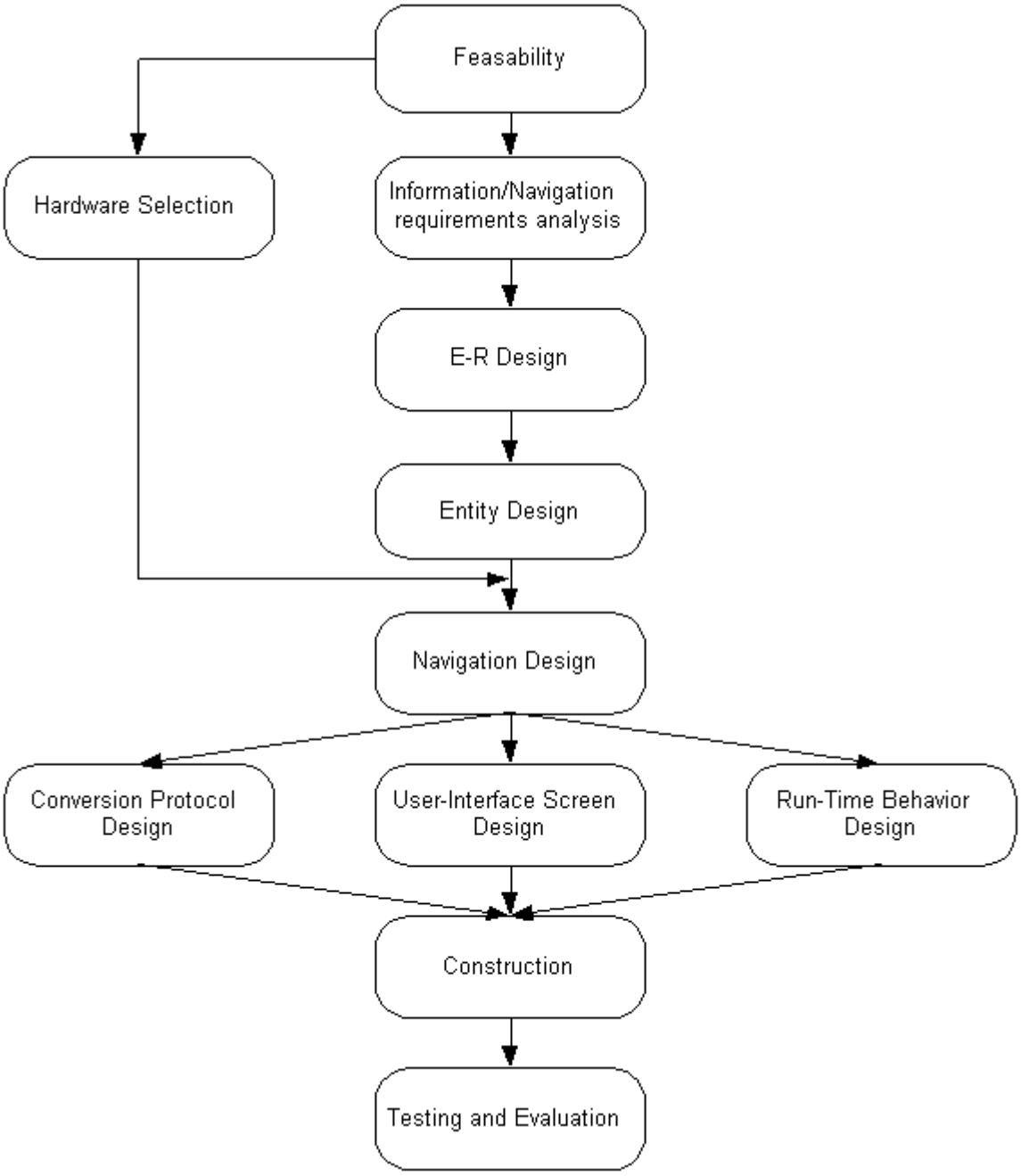
RMM maakt als één van de eerste methoden gebruik van een stapsgewijze procedure voor hypermedia design en ontwikkeling.

### 3.3.1 RMDM

Vermits entiteiten een groot aantal attributen kunnen bevatten, kan het onhandig zijn om al die attributen bij iedere instantie van die entiteit weer te geven. Daarom worden attributen van een entiteit in een “slice” (zie figuur 3.1) geplaatst. Die “slice” wordt als een soort verzameling van attributen gezien.

De navigatie wordt bij RMDM mogelijk gemaakt door de toegangsnotaties die in figuur 3.2 worden weergegeven. De links worden gebruikt om de toegang tussen de slices van een entiteit voor te stellen.
Het navigeren tussen de entiteiten zelf gebeurt via de indexen, rondleidingen en groeperingen.
Een groepering kan gezien worden als een menuachtig mechanisme dat “access” toegankelijk maakt naar andere delen van de web applicatie.

In figuur 3.3 wordt aangegeven welke de stappen zijn binnen een volledige web applicatie ontwikkeling volgens RMM. RMM legt zich daarbij vooral toe op design, ontwikkeling en constructie.




Figuur 3.3: de verschillende stappen binnen de RMM design methode

### 3.3.2 E-R Design

De eerste stap is het representeren van het informatiedomein van de applicatie via een “Entiteit-Relatie” diagram. Deze entiteiten en relaties vormen, net zoals in de vorige methoden, de basis van iedere hypermedia applicatie, waardoor ze in de eindfase van de ontwikkeling dikwijls als knopen en links van de web applicatie terug te vinden zijn.

### 3.3.3 Slice Design

Deze stap, die enkel bij hypermedia applicaties voorkomt, bepaalt hoe de informatie binnen de gekozen entiteiten zal gepresenteerd worden en hoe de gebruikers toegang zullen krijgen tot deze informatie. Dit houdt in dat de entiteiten opgesplitst worden in betekenisvolle slices en georganiseerd worden in een hiërarchie van entiteiten.
De slices kunnen gezien worden als een deelverzameling van de attributen van een entiteit.
Het resultaat van deze fase is een slice diagram (kan gezien worden als een netwerk van slices).
Het diagram modelleert ook de navigatie tussen de slices door middel van links, al dan niet in twee richtingen.

### 3.3.4 Navigatie Design

Hier worden de links beschreven die nodig zijn binnen de hypertext applicatie. Iedere relatie die in het slice diagram voorkomt moet worden geanalyseerd. Indien, afhankelijk van de vereisten, een associatieve relatie mogelijk moet zijn voor het navigatiegedeelte, dan wordt deze relatie vervangen door één of meerdere RMDM toegangsstructuren.

In eerste instantie wordt de navigatie, dat gebaseerd is op die associatieve links, tussen de entiteiten uitgevoerd. Daarna gebeurt de design van de toegangsstructuren op een hoger niveau door de interessante elementen te groeperen.

Gedurende de derde stap moet de ontwerper dus zowel de informatiecomponenten als de relaties die bezocht zullen worden, identificeren. Daarnaast geeft de ontwerper in deze stap ook een beschrijving van de groeperingen die zullen weergegeven worden en de daarbij horende toegangsstructuren.

### 3.3.5 Conversieprotocol Design

Deze stap gebruikt een verzameling van conversieregels om ieder element van het RMDM diagram om te zetten in een bruikbaar object van het doelplatform waarin de applicatie uiteindelijk zal ontwikkeld worden.

### 3.3.6 Gebruikersinterface Design

Hier wordt de design van de schermlay-out uitgewerkt voor ieder object dat bekomen is in de derde stap, het navigatiedesign.

### 3.3.7 Runtime Behaviour Design

Beslissingen, zoals de implementatie van de “history”, backtracking- en navigatiemechanismen en andere gebeuren in deze stap.

### 3.3.8 Constructie en testen

Deze allerlaatste stap komt volledig overeen met deze, die in de klassieke software engineering gevolgd wordt voor het testen van de software (in dit geval hier de web sites).

### 3.3.9 Besluit

Ook in RMM is duidelijk geen rekening gehouden met mogelijke data-manipulatie via web sites.
Wellicht was er bij de ontwikkeling van deze methode, net zoals dit bij HDM het geval was, nog maar weinig (of zelfs geen) sprake van de mogelijkheid om applicaties via web sites te gebruiken.

In tegenstelling tot HDM is er bij de opvolger van RMM, namelijk RMM+, wel sprake van data manipulatie. In deze laatste wordt RMM+ besproken waarbij er speciale aandacht zal uitgaan naar dit gedeelte waarin de design uitgewerkt wordt van mogelijke toepassingen om data via een web site in een databank te manipuleren.

## 3.4 Relationship Management Methodology “Extended” (RMM+)

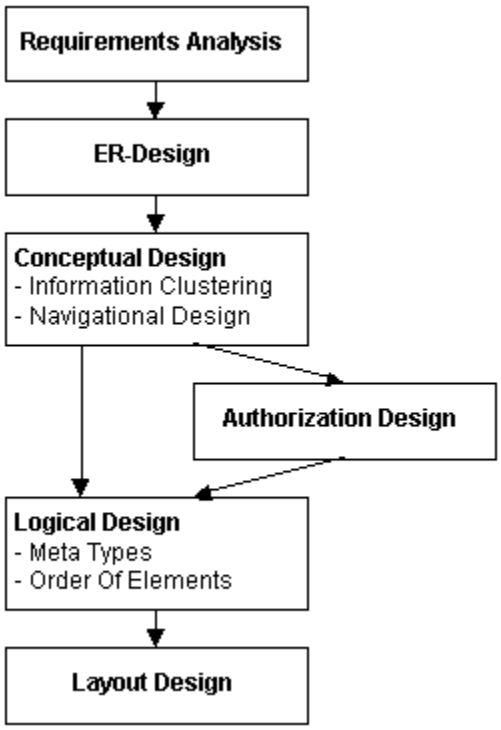
RMM+ ([Zol]) geeft in eerste instantie reeds aan dat zowel RMM als andere methoden zich enkel specialiseren in de informatie representatie. De uitbreidingen van die methoden zouden zich daarbij vooral ook moeten bezighouden met een hoge mate van interactiviteit met de gebruikers.
Vooral sinds kort zijn interfaces, die toelaten om een interactie te creëren met de data van het systeem, een basisvereiste geworden.
Aan de ene kant houdt dit in dat de mogelijkheid moet bestaan om bijvoorbeeld query-interfaces en componenten, die de onderliggende data van de web site onderhouden, te modelleren.
Aan de andere kant is het creëren van interfaces die data op een server kunnen manipuleren onoverkomelijk verbonden met een stevig autorisatieconcept.

Om maximale consistentie te behouden bij de frequente wijzigingen en technieken voor hypermedia applicaties, is het niet minder dan normaal dat ook dit autorisatieconcept tot de modelleermethode behoort.

Een eerste (en onmiddellijk belangrijk) verschil met RMM is een onderscheid dat wordt gemaakt tussen p-slices (presentatieslices, komen overeen met de top-level slices uit RMM) en m-slices (komen overeen met de slices op lager niveau uit RMM). Een andere nieuwigheid is de introductie van enkele domeinprimitieven (slices) om interactieve pagina's met queries of "input-forms" te modelleren.
Een volledig nieuw concept, om toegangsrestricties tot informatie van bepaalde pagina's te kunnen definiëren, is toegevoegd aan de methode.

### 3.4.1 Het RMM+ design proces

Het design proces wordt binnen RMM+ beschreven als een aantal te volgen stappen, maar het is belangrijk om op te merken dat in de praktijk verschillende iteraties van deze stappen (of een deel ervan) moeten uitgevoerd worden om tot een goede en onderhoudbare hypermedia applicatie te komen.




Figuur 3.4: de verschillende stappen binnen de RMM+ design methode.

De verschillende stappen houden in:

**Stap 1:** Analyse van de vereisten

- definitie van het applicatiedomein
- identificatie van de mogelijke gebruikers

- specificatie van systeemfunctionaliteit en gebruik

Deze stap kwam in vorige methoden reeds aan bod en zal daarom hier niet verder besproken worden

**Stap 2:** E-R design

Een E-R model die de objecten en relaties uit de reële wereld van het applicatiedomein weergeeft.
Ook deze stap werd reeds eerder besproken (bij RMM) en hoeft ook hier geen verdere uitleg.

**Stap 3:** Conceptueel design

Een conceptueel design gebaseerd om het E-R design waarbij aspecten zoals “information-clustering” (welke informatie moet op welke pagina’s terechtkomen) en navigatie bepaald worden.

**Stap 4:** Autorisatie design

In deze stap wordt de toegang naar de verschillende delen van de applicatie gelimiteerd en kunnen rollen gedefinieerd worden om adaptieve pagina’s te voorzien.

**Stap 5:** Logisch design

The orde van de elementen op de web site en de presentatie van attributen en links worden gespecificeerd.

**Stap 6:** Lay-out design

In de laatste stap worden alle aspecten met betrekking tot de layout van de HTML-pagina’s uitgewerkt. Deze stap kan afhankelijk zijn van de implementatieomgeving en de tools die kunnen gebruikt worden om de applicatie uit te werken.

### 3.4.2 Conceptueel Design

Zoals eerder reeds werd aangegeven bevat deze stap twee grote aandachtspunten:

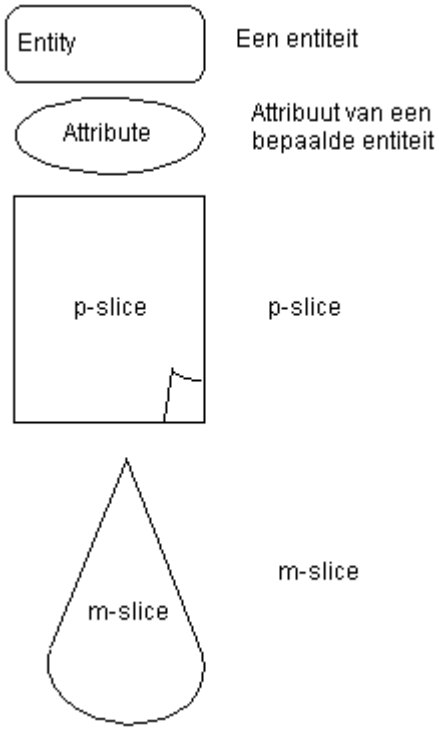
- Informatiegroepering is het proces dat relevante en gerelateerde stukken informatie op één plaats samenbrengt.
- Navigatiedesign bepaalt de toegangswegen waarlangs de gebruiker zal kunnen navigeren.

In RMM werd reeds uitgelegd hoe slices werden gebruikt om attributen, toegangsstructuren en andere elementen van entiteiten in eenheden samen te brengen. Ook was het mogelijk om de slices in een geneste structuur onder te brengen.
Hoewel heel krachtig en flexibel bleken deze slices een aantal onoverkomelijke nadelen te hebben, vooral doordat er geen onderscheid werd gemaakt tussen de top-level slices (die overeenkomen met de HTML-pagina's) en de low-level slices (die slechts onderdelen van pagina's zijn).
Enkele van die nadelen:

- verlies van overzicht: bij grote applicaties wordt het moeilijk om uit te maken welke slices bedoeld zijn voor de effectieve pagina's en welke voor onderdelen van pagina's.
- conceptuele fouten: doordat de twee soorten slices verschillende concepten voorstellen en later op een andere manier moeten behandeld worden, zouden ze ook in deze stap op een verschillende wijze gemodelleerd moeten worden.
- inconsistenties: doordat nesting met slices mogelijk is, zou het ook mogelijk zijn om volledige pagina's in elkaar te nesten, waardoor inconsistenties optreden.

Het RMM+ Data Model is in feite een uitbreiding van het model dat bij RMM gebruikt werd en is daardoor dus ook gebaseerd op RMDM. In het volgende stuk is echter te zien dat RMM+ een grote uitbreiding heeft gemaakt op dit data model.

Er zijn een aantal basisprimitieven zoals entiteiten, attributen en slices. De slices worden bij RMM+, zoals eerder reeds vermeld, opgedeeld in p-slices en m-slices. De p-slices vertegenwoordigen HTML-pagina's terwijl m-slices voor componenten van die pagina's staan.
Iedere slice behoort altijd tot één entiteit. Een slice kan zowel attributen van zijn entiteit bevatten als gegroepeerde attributen gedefinieerd in een m-slice, zodat het mogelijk wordt om m-slices in een geneste structuur te verwerken.




Figuur 3.5: de bijkomende notaties binnen RMM+ ten opzichte van RMM.

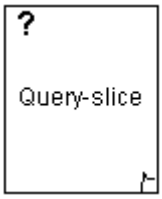
Om de mogelijkheid te geven relaties tussen de attributen van een entiteit weer te geven, kunnen slices ook één of meerdere toegangsstructuren zoals hyperlinks, indexen of “guided tours” bevatten.
Volgende punten zijn hierbij belangrijk:

- iedere hyperlink moet zowel een start m-slice als een doel slice hebben;
- een index is een lijst van instanties van een andere entiteit die in relatie is met de entiteit van de slice;
- een “guided tour” is dan weer een verbonden lijst van instanties van dezelfde entiteit.

Een nieuw en belangrijk onderdeel van RMM+ zijn de getypeerde slices. Deze slices verbeteren namelijk de expressieve kracht van RMM+ en laten toe om niet alleen passieve representatiepagina’s te modelleren, maar ook bijvoorbeeld query-interfaces die opzoekingen toelaten op de onderliggende databank(en). Daarnaast zijn er ook mogelijkheden om de resultaten van de uitgevoerde queries te presenteren en wordt de mogelijkheid voorzien om web interfaces te modelleren die informatie aan de databank toevoegen.

De verschillende types slices die in RMM+ gedefinieerd worden:

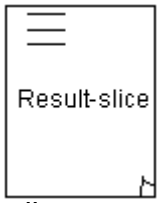
- query-slices:




Figuur 3.6: een query-slice.

Deze slices stellen pagina's voor die HTML-forms bevatten om queries op de onderliggende databank uit te voeren. Attributen, hyperlinks en indexes kunnen tot deze slices behoren.
Attributen: dit kunnen verschillende types invoervelden zijn.
Hyperlinks: kunnen verwijzen naar andere query-slices binnen dezelfde entiteit.
Indexes: laten toe om zoekoperaties uit te voeren op andere entiteiten die een relatie hebben met de entiteit waar we van uitgaan.
Het uitvoeren van de query in de HTML-form kan gezien worden als een navigatie van de query-pagina naar een nieuwe pagina die de resultaten bevat van de query. Het overgaan van de ene naar de andere pagina wordt binnen de query-slice aangeduid door middel van een submitter-symbool.

- result-slices:



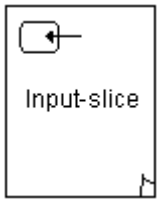

Figuur 3.7: een result-slice.

Hoewel het presenteren van de resultaten van een query overeenkomt met de voorstelling van een detailpagina, is een extra domeinprimitief toegevoegd voor deze taak.
Een detail-slice met een index kan namelijk niet gebruikt worden om query-resultaten weer te geven, omdat een index een toegangsstructuur is en de slice waarbij deze hoort heeft altijd een andere entiteit dan de slice waarbinnen de toegangsstructuur geplaatst is.

Een result p-slice kan alle elementen van de oorspronkelijke detail-slice bevatten. Het probleem bij result-slices is dat niet op voorhand kan bepaald worden hoeveel resultaten de query zal opleveren. Dit

kan opgelost worden door de index te verdelen over verschillende pagina's, maar dit moet dan in latere design stappen uitgewerkt worden.
Een andere oplossing is de queries te verfijnen tot een degelijk aantal resultaten is bekomen. Dit kan binnen RMM+ in deze stap gerealiseerd worden door in een result p-slice query m-slices toe te laten, die een verdere verfijning van de oorspronkelijke query als resultaat kunnen hebben.

- input-slices:



Figuur 3.8: een input-slice.

Naast data representatie is het tegenwoordig ook van groot belang om data manipualtie via het web mogelijk te maken. Dit blijkt vooral cruciaal te zijn bij web sites met een groot aantal verschillende partners in verwerkt.
Daarom wordt in RMM+ het concept van input p-slices geïntroduceerd.
De input slices kunnen volgende attributen bevatten:

- attributen: voor ieder attribuut aanwezig in de slice moet een input-form van een bepaald type voorzien zijn;
- hyperlinks: een link refereert naar andere input-slices van dezelfde entiteit;
- index: indexes in een input-slice laten toe om instanties van andere entiteiten in relatie te brengen met de entiteit dat gecreëerd wordt.

### 3.4.3 Autorisatie design

Meer en meer web sites bevatten naast de publieke informatie ook informatie die slechts voor bepaalde groepen van gebruikers mag zichtbaar zijn.
Dit betekent dat er voor die stukken informatie een toegangscontrole moet zijn die moet uitmaken of gebruikers deze informatie al dan niet mogen zien.
Via het internet is deze toegangscontrole over het algemeen nog altijd verwerkt in een standaard HTTP-autorisatie die plaats vindt via de web server, wat een groot aantal nadelen met zich meebrengt.

Een aantal nadelen worden hier meegegeven:

- de toegangscontrole is enkel mogelijk voor statische HTML-pagina's;
- enkele volledige pagina's kunnen geautoriseerd worden, een grotere specificatie ligt niet binnen de mogelijkheden;
- er is geen samenwerking met andere applicaties, zoals een databanksysteem, mogelijk;
- er is een afhankelijkheid tussen de toegangscontrole en de fysieke structuur van de web site.

Deze nadelen geven aan dat de toegangscontrole moet verwerkt worden in een conceptueel design van de web site, waardoor later duidelijk wordt welke controlesysteem gebruikt zou moeten worden.

Er bestaan een aantal traditionele toegangscontroletechnieken zoals "DAC" en "MAC", die de permissies onmiddellijk associeren met de gebruikers. Een andere techniek, "RBAC (Role Based Access Control)", dat toegepast wordt binnen RMM+, associeert de permissies met bepaalde rollen en stelt deze rollen dan in verband met de gebruikers.

- RBAC voor p-slices:

  Dit komt overeen met de traditionele toegangscontrole. Hierbij kunnen dus toegangsregels op volledige HTML-pagina's gedefinieerd worden.

  Alhoewel een restrictie op een p-slice altijd tot gevolg heeft dat alle m-slices op die p-slice beperkt worden, betekent dit nog niet automatisch dat die m-slices dezelfde restrictie overnemen.

  Dit is in de eerste plaats niet nodig, omdat een opgelegde restrictie op de p-slice er altijd voor zorgt dat de m-slices ook niet kunnen weergegeven worden indien de toegang tot de p-slice verboden is.

  In de tweede plaats kan het overnemen van restricties op p-slices door m-slices leiden tot inconsistenties doordat de m-slices ook binnen andere p-slices kunnen gebruikt worden waar dergelijke restricties niet van toepassing zijn.

- RBAC voor m-slices:

  Indien de toegangsrestricties op m-slices gelegd worden, krijgen we een geheel ander effect dan bij p-slices het geval was. Hier worden alle onderdelen, waarop geen restricties van toepassing zijn, weergegeven, samen met de onderdelen waarvoor de gebruiker toegang heeft. Enkel de m-slices waarop dus geen toegang gegeven is voor die specifieke gebruiker worden niet weergegeven.

Door gebruik te maken van deze techniek is het zodoende mogelijk om adaptieve pagina's op te bouwen die afhankelijk van de gebruikers en de gedefinieerde rollen, verschillende informatie (of een verschillende hoeveelheid aan informatie) zullen weergeven.

### 3.4.4 Logisch Design

In dit onderdeel wordt de logische representatie van de inhoud van de HTML-pagina's beschreven.
Deze beschrijving houdt volgende aspecten in:

- de orde van de elementen;
- de betekenis van de elementen;
- de logische representatie van elementen (door middel van metatypes);
- individuele logische aspecten van bepaalde slice types.

Die meta-types komen overeen met de "stylesheets" uit HTML, maar hebben een aantal bijkomende eigenschappen zoals:

- afhankelijkheid van de logische representatie van het attribuut binnen de databank;
- onafhankelijk van eventuele wijzigingen binnen HTML.

### 3.4.5 Lay-out Design

Deze laatste stap is het minst ondersteund door het overgrote deel van de methoden door het feit dat een goede formalisatie niet altijd evident is.
Er zijn binnen RMM+ toch een aantal punten die kunnen beschreven worden. Doordat p-slices na deze stap uitgewerkte HTML-pagina's moeten voorstellen, kan men aan die p-slices bepaalde gegevens zoals titel, achtergrond, lettertype en kleuren meegeven. Ook globale "headers" en "footers" kunnen voor die p-slices reeds beschreven worden.

### 3.4.6 Besluit

RMM+ is de enige methode die rekening houdt met applicatie type web sites. Ook minder bekende methoden zoals W3DT [BN 96] en anderen houden enkel rekening met statische pagina's.

In het volgend hoofdstuk wordt onderzocht of de oplossing die RMM+ aanbrengt ook bruikbaar is binnen WSDM, eventueel met kleine aanpassingen.

# Hoofdstuk 4:

# Een mogelijke oplossing

Er zijn in de loop van het onderzoek een aantal mogelijkheden uitgetest die eventueel een oplossing konden bieden om dynamische web sites te kunnen modelleren binnen WSDM.

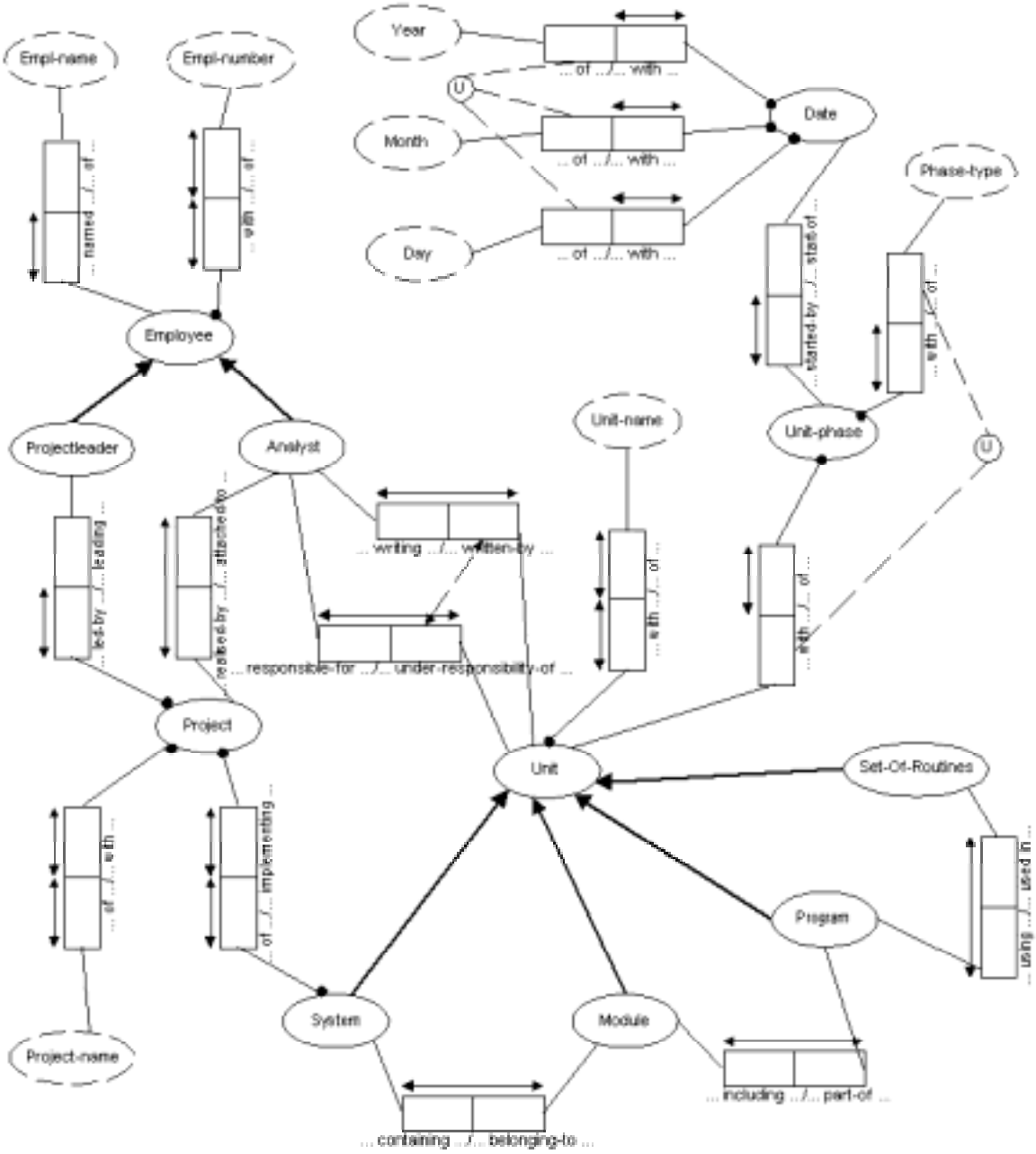
Die mogelijkheden bespreken we in dit hoofdstuk, waarbij ook verklaard wordt waarom die mogelijkheden al dan niet effectief bruikbaar zijn. De oplossingen zullen ook geïllustreerd worden door voorbeelden en achteraf komen ook de voor- en nadelen van die oplossing aan bod.

## 4.1 Een voorbeeld

Bij het voorstellen van de verschillende mogelijkheden zal gebruik gemaakt worden van onderstaand voorbeeld, zodat het snel duidelijk wordt hoe de oplossing eventueel verder zou moeten gebruikt worden.
Alhoewel dit een relatief klein voorbeeld is, komen praktisch alle situaties voor die men ook in uitgebreidere situaties kan ontmoeten.

### 4.1.1 Universe Of Discourse

De situatie in dit voorbeeld is dit van een informaticadepartement binnen een bedrijf. Het departement werkt op verschillende projecten.
De projecten worden geïmplementeerd door analisten en worden geleid door een projectleider.
Het departement kan werknemers hebben die noch analisten noch projectleiders zijn.
Alle werknemers van het departement hebben een identificatienummer.
De implementatie van een systeem is altijd het resultaat van een project.
Een systeem bevat modules, de modules bevatten op hun beurt programma’s en de programma’s maken gebruik van routinesets.
Systemen, modules, programma’s en routinesets zijn allemaal units.
Een unit kan geschreven worden door een analist en is ook onder de verantwoordelijkheid van een analist. Deze twee hoeven niet dezelfde te zijn. Een unit wordt ontwikkeld in verschillende fasen, die van een bepaald type zijn. Een startdatum kan gegeven worden aan iedere unit fase in de implementatie.
De werknemers, units, projecten en fasen hebben elk een naam.
Datums worden weergegeven door een jaar, maand en dag.




Figuur 4.1: Het conceptueel schema horend bij het voorbeeld.

## 4.2 Gebruik maken van RMM+, een eerste mogelijkheid

In RMM+, dat in het vorige hoofdstuk reeds uitvoerig werd besproken, is er rekening gehouden met de design van dynamische web sites.
Het was dan ook in eerste instantie de bedoeling om de oplossing die RMM+ hanteert binnen WSDM over te nemen en te gebruiken.

In het eerste hoofdstuk werd reeds een volledige beschrijving gegeven van de WSDM methode. Deze zal hier nu niet herhaald worden, maar er zal enkel aangegeven worden op welke plaatsen de wijzigingen aangebracht zouden worden door de invloed van de oplossing met RMM+.

### 4.2.1 Beschrijving van de oplossing

- Mission Statement en Audience Modelling

  Het bepalen van het “Mission Statement” en de “Audience Modelling” ondergaan geen wijzigingen. Het doel van WSDM is vooral om gebruikersgericht te werken en daarbij zijn deze stappen van groot belang waardoor het niet verstandig lijkt om wijzigingen aan te brengen binnen deze stapen.

- Conceptual Design

  In dit deel moeten de wijzigingen uitgevoerd worden om de design van dynamische web pagina’s mogelijk te maken.
  Het is echter niet volledig duidelijk waar we de oplossing van RMM+ binnen deze stap zouden moeten situeren.

  Indien we bij WSDM nagaan hoe we te werk gaan, is het duidelijk dat er eerst en vooral een informatiemodel wordt opgesteld voor de verschillende gebruikersklassen.
  Deze informatiemodellen worden aan elkaar verbonden zodat we daarna een volledig overzicht krijgen van de algemene datastructuur voor de web site.
  Daarna komt een structuur voor de verschillende navigatiemogelijkheden voor alle gebruikersklassen. Ook hier wordt eerst voor iedere gebruikersklasse een aparte structuur aangemaakt die dan achteraf allemaal aan elkaar verbonden worden.

  Bij RMM+ zelf wordt eveneens een informatiemodel opgesteld, zij het met een andere methode (Entity-Relationship), maar de rest van het conceptuele design is volledig anders uitgewerkt.
  Er wordt binnen RMM+ immers een introductie gemaakt (net zoals dat eigenlijk reeds bij RMM het geval was) van een model die volledig los staat van andere designontwerpen.

  Dit Relationship Management Data Model (RMDM), dat reeds in vorig hoofdstuk uitvoerig werd besproken (zie ook [ISB 95,

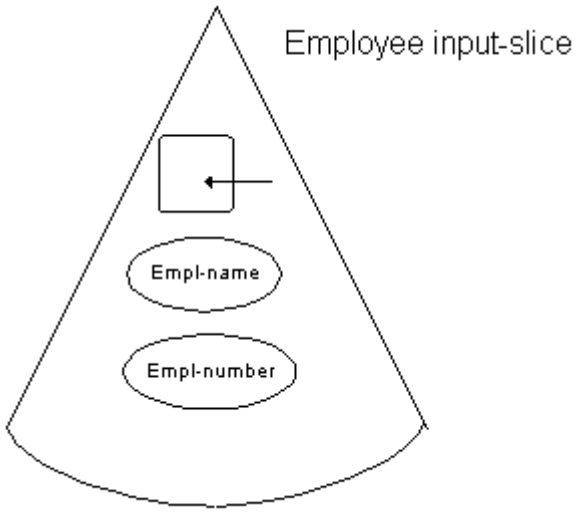
Zol]), wordt gebruikt om de verschillende pagina's van de web site uit te werken.

In dit model worden de onderdelen vooral opgebouwd uit m-slices (die delen van pagina's voorstellen) en p-slices (komen overeen met volledige web pagina's). Die slices bevatten dan zelf entiteiten (en hun attributen).

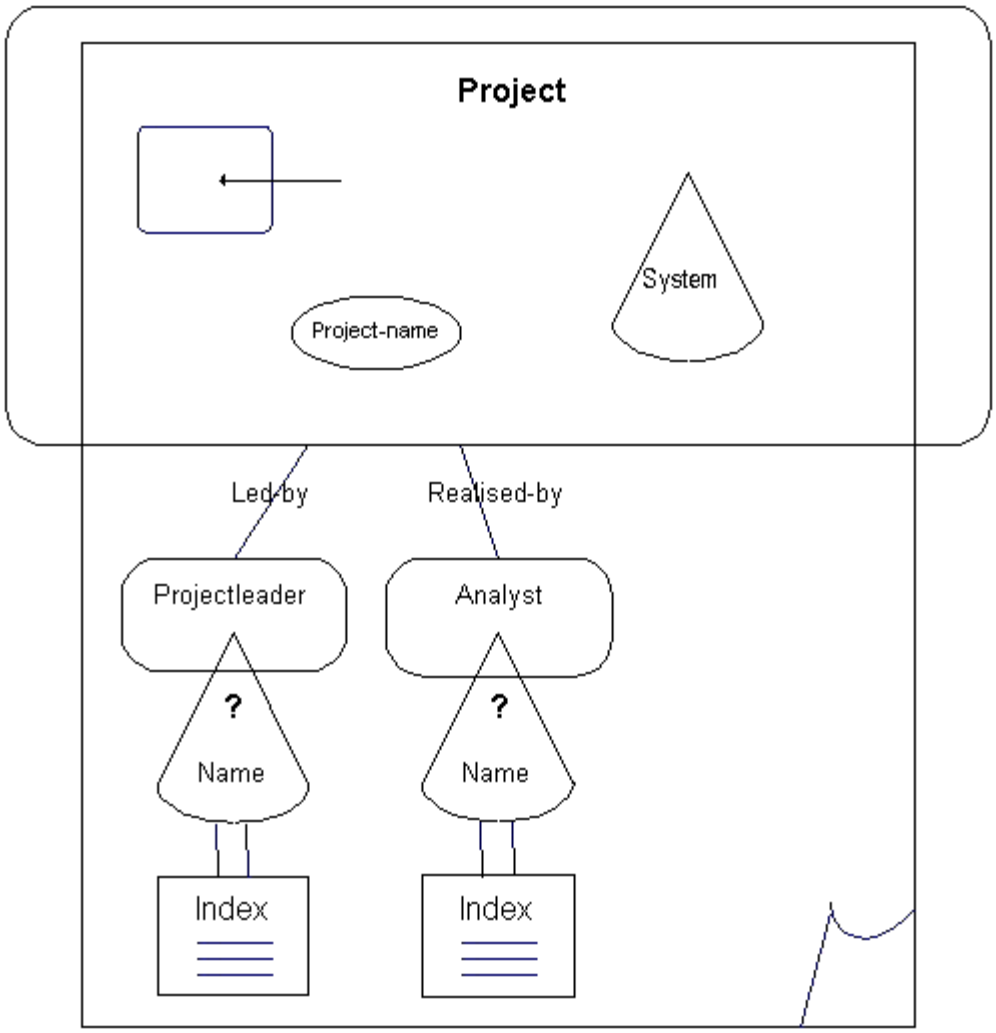
Door gebruik te maken van voorgedefinieerde getypeerde slices (p- of m-slices) is het dan ook mogelijk om pagina's te creëren die gebruikt kunnen worden om data manipulaties uit te voeren.

Binnen dit model worden de links naar en afkomstig van de andere pagina's ook meegegeven waardoor men automatisch een navigatiestructuur opbouwt.
Het opbouwen van het navigatiemodel uit WSDM lijkt dan ook totaal overbodig te worden.

Indien we de oplossing van RMM+ willen introduceren binnen WSDM zijn we dus verplicht om het volledige RMDM-model van RMM+ over te nemen.
Net zoals we binnen WSDM het conceptueel datamodel voor de verschillende gebruikersgroepen opstellen, zal het ook nu de bedoeling zijn om voor die groepen een verschillend RMDM-model op te stellen.
In figuur 4.2 en figuur 4.3 hebben we enkele voorbeelden van input-slices.




Figuur 4.2: een input m-slice voor een Employee. Hierin zouden we in de corresponderende web pagina een Empl-name en een Empl-number moeten ingeven.




Figuur 4.3: een input p-slice voor een“Project. Daarbij is de Project-name een variabele en System een aparte m-slice. De indexes die gebruikt worden voor Projectleader en Analyst laten toe een relatie te maken met andere entiteiten van de instantie die gecreëerd worden.

- Na het Conceptueel Design

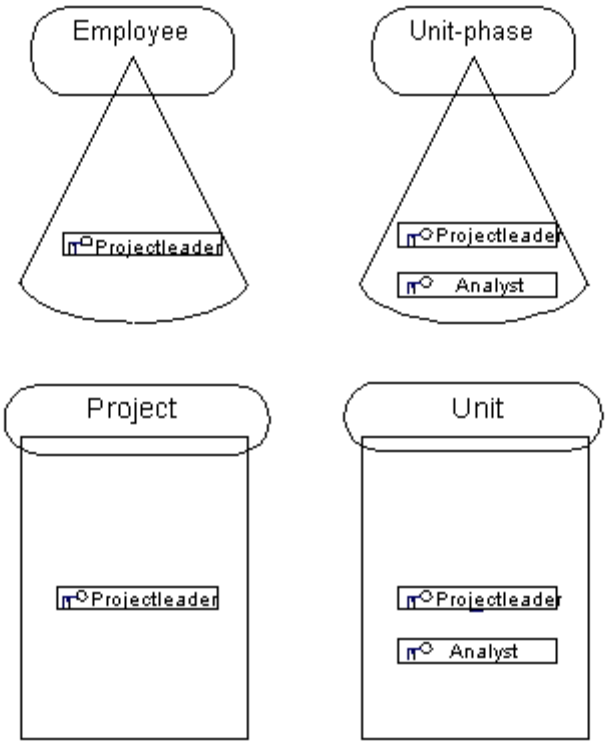
Het mogelijk maken van data manipulatie via het internet zorgt er onherroepelijk voor dat er een beveiliging nodig is. Niet iedereen mag immers de toegang krijgen om informatie te gaan wijzigen.
Ook tussen de gebruikersgroepen bestaat er een diversiteit van de mogelijkheden om data manipulatie uit te voeren.

Een mogelijk voorbeeld zou zijn dat Analysten nieuwe Units en Unit-phases mogen toevoegen, maar geen nieuwe projecten kunnen creëren. De Projectleaders krijgen de toegang tot alles, dus

ook nieuwe werknemers en nieuwe projecten aan de databank toevoegen.

In de huidige versie van WSDM is daarover nog geen sprake, te meer omdat er nog geen sprake was van data manipulatie en dergelijke beveiliging dan niet echt hoeft.
Vermits dit nu gaat veranderen is er dan ook behoefte aan de design van een goede beveiliging.
Bij RMM+ is er wel een beveiliging uitgewerkt. Er wordt in die methode gebruik gemaakt van de RBAC-techniek (Role Based Access Control) [Zol].
Vermits deze nauwkeurig toegepast is op het RMDM-model kan ook deze techniek worden overgenomen. Door middel van de RBAC-techniek kan aangegeven worden welke pagina's van de web site voor welke gebruikersklassen toegankelijk zijn.

Bij het gebruiken van RBAC kan men de toegang beperken bij de p-slices, waarmee we een traditionele toegangscontrole verkrijgen waarbij bepaalde pagina's voor gebruikers al dan niet toegankelijk zullen zijn.
Een tweede mogelijkheid bestaat er in de beperkingen te leggen op de m-slices. Dit resulteert dan in adaptieve pagina's. Op die manier worden alle m-slices waar geen restricties op liggen, samen met de slices waar de gebruiker toegang tot heeft, weergegeven. Dit resulteert voor de verschillende gebruikersgroepen dan ook in verschillende pagina's.




Figuur 4.4: enkele voorbeelden van mogelijke slices met autorisatie. Enkel de groepen die een key hebben op de slice mogen deze pagina's zien.

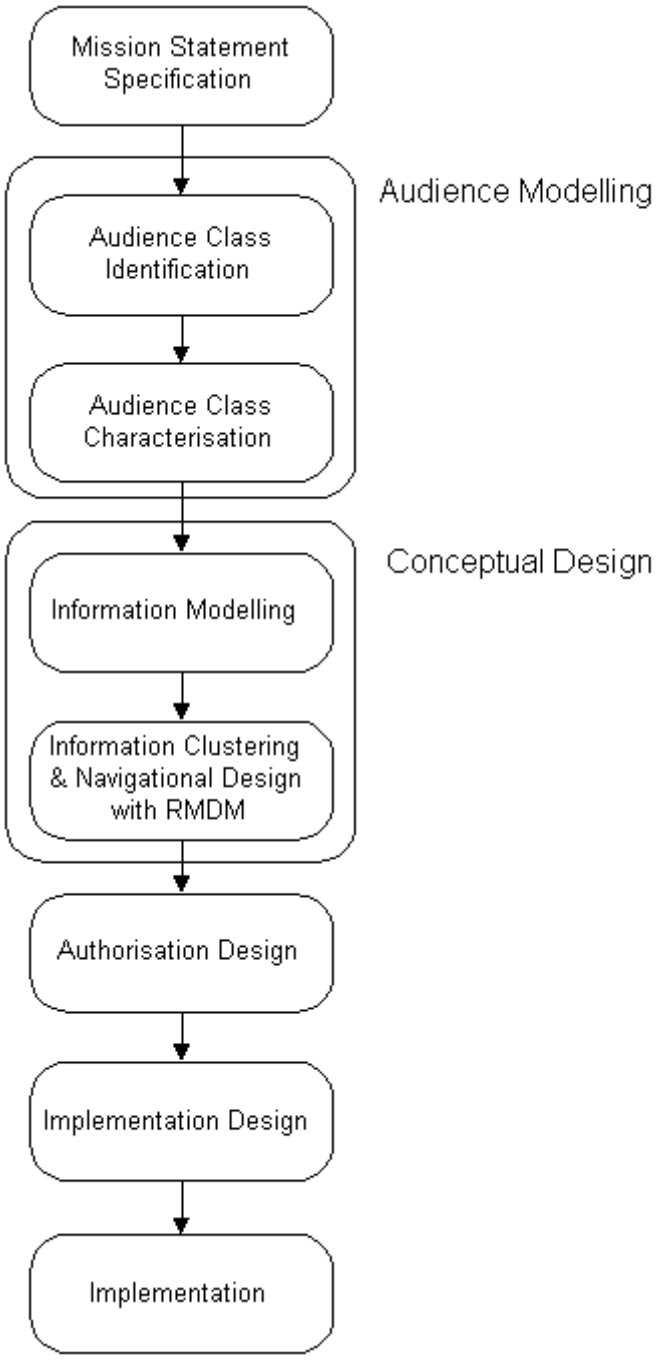
- vervolg van WSDM

Na het autorisatiedesign resten er enkel nog de "Implementation Design" en dan de uiteindelijke implementatie zelf van de web site. Deze stappen, die reeds volledig uitgewerkt zijn binnen WSDM hoeven hier geen verdere uitleg.

De design van de implementatie en de implementatie worden niet beïnvloed door het gebruik van RMM+ binnen de WSDM-methode.

### 4.2.2 Vernieuwde structuur van WSDM

De structuur van WSDM zou door het invoegen van het RMDM-model uit RMM+ en het bijhorende autorisatieconcept een kleine wijziging ondergaan. De vernieuwde structuur zou er dan als volgt uitzien:




Figuur 4.5: De vernieuwde structuur van WSDM indien we gebruik maken van de oplossing uit RMM+.

### 4.2.3 RMM+, een bruikbare oplossing?

Het invoegen van de RMM+ oplossing binnen WSDM heeft als voordeel dat de methode van WSDM niet volledig hoeft gewijzigd te worden.

Er is enkel het nieuwe onderdeel dat wordt toegevoegd, waardoor het navigatiedesign van WSDM zelf wegvalt.

Er zijn echter een aantal nadelen aan RMM+ verbonden die een groter overwicht hebben dan de voordelen om deze oplossing niet door te voeren. Er zal dus naar een andere oplossing moeten gezocht worden.
De belangrijkste nadelen zijn:

- RMM+ maakt gebruik van Entity-Relationship diagrammen om zijn datastructuren op te bouwen. Het RMDM-model wordt eveneens beïnvloed door het gebruik van deze diagrammen.
  WSDM maakt echter gebruik van ORM (Object Role Modelling) voor de datamodellen, wat veel krachtiger is dan E-R. Het gebruik van RMDM zou er voor zorgen dat er veel van de kracht van ORM verloren gaat.

- RMM+ heeft wel voorgedefinieerde slices om nieuwe informatie toe te voegen, maar bij dynamische web sites is het ook dikwijls de bedoeling dat de informatie op de web site kan gewijzigd of verwijderd worden. Bij RMM+ is daarover nog geen sprake.
  En zelfs indien dit nu over het algemeen niet zo is, dan is het toch beter om dit reeds te voorzien, zodat dit later niet moet gedaan worden.

- Bij het toevoegen of wijzigen van informatie kan het soms van belang zijn dat de gebruiker de mogelijkheid krijgt zowel nieuwe informatie toe te voegen of reeds bestaande te selecteren. Afhankelijk van de relaties tussen de informatie kan de gebruiker maximum één of in het andere geval meerdere gegevens invoeren.
  Dit weerspiegelt zich in verschillende “input-forms” die in de implementatiedesign en de implementatie aan bod komen
  In RMDM is er geen rekening gehouden met deze mogelijkheden.

- Het is altijd de bedoeling om de design zo eenvoudig mogelijk te houden met als doel de gebruiksvriendelijkheid van de methode zo groot mogelijk te houden.
  Enkel in het geval van een eenvoudig te hanteren methode zal de ontwerper bereid zijn deze methode ook effectief te gebruiken.
  Door het toevoegen van het RMDM-model moet de ontwerper naast ORM nog een nieuwe technologie kennen.
  Bij het gebruiken van de vernieuwde WSDM zou de gebruiker een eerste datamodel moeten opstellen en daarna die structuur

nog eens moeten hergebruiken, maar dan in een andere versie en met andere functionaliteiten die RMDM te bieden heeft.

Mede door deze redenen werd overwogen om op zoek te gaan naar een andere oplossing. Een vereiste waarmee we rekening zouden houden is het feit dat de oplossing moet gebruik kunnen maken van ORM.
Op die manier moet het mogelijk zijn om die oplossing gemakkelijker binnen WSDM te integreren.

In het volgende onderdeel zullen we dan ook een oplossing krijgen die nog niet optimaal is, maar die wel aanleiding zal geven tot een goede en definitieve oplossing.

## 4.3 Een tweede mogelijke oplossing dankzij RIDL

Het tweede onderzoek dat werd uitgevoerd naar een oplossing heeft een heel andere bron als startpunt genomen. Ditmaal was RIDL [DMP 83] een uitgangspunt van waaruit verder gewerkt werd.

In dit deel wordt een uitleg gegeven over de RIDL-taal, waarom deze taal een mogelijke bron kan zijn voor een uitstekende oplossing, de vereiste uitbreiding van RIDL (al dan niet effectief doorgevoerd binnen RIDL) om tot een mogelijke oplossing te komen en waarom deze oplossing uiteindelijk toch niet doorgevoerd werd.

### 4.3.1 Wat is de RIDL-taal?

De taal RIDL is het resultaat van een poging om de diepere semantische betekenis te geven van een syntactisch “netwerk”, zoals men kan terugvinden in de beschrijving van een Conceptueel Schema.
RIDL staat voor “Reference and Idea Language”

RIDL is uitgewerkt tot een geïntegreerd, formeel, syntactisch hulpmiddel voor informatie- en procesanalyse, semantische specificatie, constraint-definitie en een query/update-taal.
Deze query/update-taal is wel eerder op conceptueel dan op dataniveau uitgewerkt.

De taal manipuleert, definieert en legt restricties op informaticastructuren en informatiestromen die beschreven zijn in conceptuele schema’s van het type ORM.
De taal is praktisch volledig uitgewerkt om de beschrijving en communicatie rond deze informaticastructuren mogelijk te maken. De syntax om te verwijzen naar de informatie is in relatie met de grafische representatie die gebruikt wordt in de conceptuele ORM schema’s.

De kracht van de RIDL-taal kan het best uitgedrukt worden in “conceptuele kracht”: de hoeveelheid semantiek of betekenis is evenredig met de hoeveelheid “denkwerk” nodig om tot een resultaat te komen. Immers, een goed model voor een conceptueel schema moet sterk overeenkomen met de menselijke redenering, waarbij de semantiek dan ook op een efficiënte manier in een netwerk kan voorgesteld worden.
Indien de syntax in relatie is met deze redenering kan men krachtige talen verkrijgen op conceptueel niveau.

Het gebruik van de RIDL-taal was in eerste instantie ontworpen voor de interactieve gebruiker. Vermits RIDL nauw samenhangt met het concept van het conceptueel schema is het dan ook een vereiste dat, bij het uitwerken van zinnen binnen de taal, het conceptuele schema van waaruit men vertrekt voorhanden is.

Binnen RIDL zijn er geen globale variabelen die beschikbaar zijn voor alle “interne” procedures. De enige globale informatie is gedefinieerd in de Informatiebasis waarop RIDL werkt. Deze basis is ondergebracht in het conceptueel schema.

#### 4.3.1.1 RIDL syntax

Het hart van RIDL’s syntax is de referentie-expressie. Deze expressie evalueert één of meerdere voorkomens van de populaties waarvan de types gedefinieerd zijn in het conceptueel schema.
Sommige speciale gevallen van expressies kunnen ook aritmetische uitdrukkingen bevatten. Deze uitdrukkingen kunnen niet in het Conceptuele Schema verwerkt worden. Meer duidelijkheid daaromtrent wordt later verklaard in enkele voorbeelden.

De belangrijkste syntactische concepten binnen RIDL zijn:

- Set- of Verzamelingexpressies:
  Deze expressie evalueert tot één enkel voorkomen of een verzameling van voorkomens van een lexicaal of niet-lexicaal object type of van een abstract supertype van een lexicaal object type zoals een getal, een string of zelfs datum of tijd.
  Het evaluatieproces kan onder andere aritmetische of verzameling operaties bevatten, evenals interactieve invoer van gegevens.

- Predikaten:
  Deze stellen een relatie voor tussen twee verzamelingexpressies. Predikaten worden voornamelijk gebruikt om de keuze te bepalen zoals in “if”-statements (of andere waar door middel van een aantal mogelijkheden een keuze moet worden gemaakt).

- Procedures en functies:

  Procedures en functies worden gebruikt om processen te definiëren die een interactie maken met het informatiesysteem of om een structuur op te stellen voor deze interacties. Hierbij moet opgemerkt worden dat de gebruiker geen globale variabelen kan gebruiken, vermits RIDL dit niet toelaat.

- Tabellen:

  Tabellen zijn in feite beschrijvingen voor “n-ary” structuren. Ze zijn voornamelijk bedoeld om de communicatie mogelijk te maken met andere omgevingen dan RIDL. Tabelbeschrijvingen kunnen voorgesteld worden als macro’s.

- Verklaringen van Namen:

  Deze zijn onder andere toekenningen en “define”-statements. Dit concept laat toe binnen RIDL een gebruikersnaam toe te kennen aan een set-expressie of aan de gedeeltelijke uitwerking ervan. Ook gedefinieerde constructies kunnen gebruikt worden als macro’s in verdere toepassingen.

- Controlestatements:

  Om een richting aan te geven binnen de controlestroom van vertakkingen of lussen. Deze controlestatements komen overeen met de klassieke vertakkingen of lussen zoals bijvoorbeeld een “for each”-lus.

- Statements van Werkwoorden:

  Een voorbeeld hierbij is het “list” werkwoord die een actie uitvoert op set-expressies, tabellen en dergelijke. Meer hierover wordt duidelijk in de voorbeelden.

#### 4.3.1.2 Voorbeelden van het gebruik van RIDL

We gaan nu enkele voorbeelden bespreken die het gebruik van RIDL moeten verduidelijken. De voorbeelden hebben allemaal betrekking op het voorbeeld uit het begin van dit hoofdstuk. Daarbij is vooral het conceptueel schema van belang.
Deze en nog tal van andere voorbeelden kunnen teruggevonden worden in de gebruikershandleiding van RIDL [DMP 83].

Na deze voorbeelden zal een uitbreiding van RIDL gegeven worden die vereist is om applicatie type web sites te modelleren. Bij deze oplossing worden ook een aantal nieuwe definities voor die uitbreiding ingevoerd.

Enkel de voorbeelden die interessant zijn voor de verdere uitwerking en het gebruik van RIDL voor ons doel zullen aan bod komen. Het mag dan ook duidelijk zijn dat de mogelijkheden van RIDL veel uitgebreider zijn.

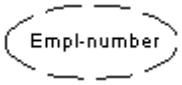
- voorbeeld 1:

“List Unit-Name of Unit written-by Analyst.”

Het resultaat van deze zin zijn alle namen van de units die geschreven worden door een analist. De units, waarvan er geen analist gekend is, worden niet aan het resultaat toegevoegd.

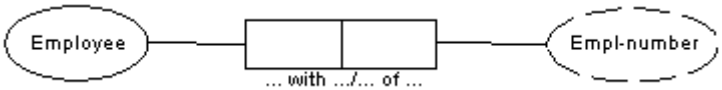
Het is mogelijk om zowel volledige populaties als delen ervan te selecteren. Populaties worden weerspiegeld in het conceptueel schema via een Object Type (= OT = LOT of NOLOT [Orm, Hal 99]).


Figuur 4.6: een NOLOT



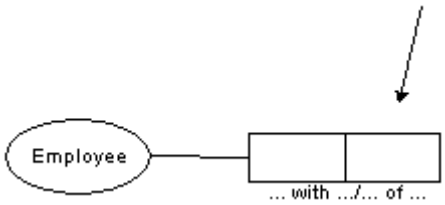

Figuur 4.7: een LOT

De geselecteerde resultaten zijn verbonden via een gegeven pad. Dit pad wordt geconstrueerd door een opeenvolging van Role-namen en OT-namen [Hal 99].




Figuur 4.8: een NOLOT in relatie met een LOT via Roles.

Een vereiste hierbij is dat de Roles en Object Types in relatie zijn met elkaar. De opeenvolging wordt een referentiepad genoemd.
Binnen een referentiepad moet iedere Role in verbinding staan met het Object Type die er aan vooraf gaat (volgens het conceptueel model).
De Companion Role (of CoRole) [Hal 99] van de Role moet dan in verbinding staan met het Object Type die volgt op de Role binnen het referentiepad. De CoRole wordt in het referentiepad niet weergegeven.



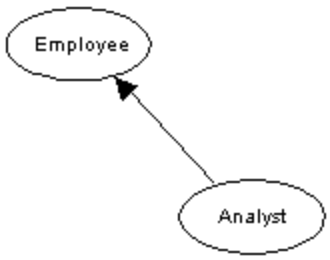

Figuur 4.9: De Companion Role "… of …" van de Role "… with …".

Een referentie kan dus gezien worden als een beschrijving van het doorlopen van het schema.

- voorbeeld 2:

"List Empl-name of Employee writing Unit."
"List Empl-name of Analyst writing Unit."

Deze twee queries geven hetzelfde resultaat. Een referentiepad werkt namelijk ook via Subtype-links [Hal 99].




Figuur 4.10: Analyst is een Subtype van Employee. Dit wordt aangeduid met een pijl van Analyst naar Employee.

Een Object Type dat in relatie staat met een Role mag dus vervangen worden door eender welk Subtype van dat Object Type.

De resultaten van de queries zijn bij gebruik van subtypes niet noodzakelijk gelijk. Zo kan het voorbeeld dat hierop volgt een ander resultaat geven dan één van beide voorgaande:
"List Empl-name of Projectleader writing Unit."

- voorbeeld 3:

Ook het gebruik van "Not" of "Any" is toegelaten in queries:

"List Empl-name of Analyst Not Writing Unit."
"List Unit-name of Any Unit."

Bij de “Not” geven we het negatieve resultaat weer van de normale query. In dit geval hier geven we een lijst van namen van Analysten die geen Unit schrijven.

Het gevolg van een “Any” in een zin is dat er maximaal één resultaat is, zelfs indien er meerdere oplossingen zijn. Het gebruik van “Any” is vooral interessant als men geïnteresseerd is in een bepaalde inhoud van het resultaat, en niet in het aantal resultaten.
Er kan ook niet bepaald worden welke van de positieve resultaten zal worden weergegeven.

- voorbeeld 4:

“List Unit-name of Unit with Unit-phase started-at Date with
Year = (The Year of Date start-of Unit-phase
(of Unit with Unit-name ‘IFD’ And with Phase-type ‘Design’
)
) – 1.”

Dit voorbeeld geeft aan dat het mogelijk is om complexe queries op te stellen, waarbij ook aritmetische vergelijkingen mogelijk zijn.

Zoals duidelijk blijkt uit de voorbeelden is het doorlopen van paden uit het conceptueel model van groot belang om queries of zinnen op te stellen binnen RIDL.
Nogmaals moeten we hierbij vermelden dat RIDL een krachtige taal is, waarbij niet alle mogelijkheden in dit stuk aan bod zijn gebracht. Een groter aantal voorbeelden is te vinden in [DMP 83]. Enkel de voorbeelden en dat onderdeel van RIDL dat interessant is om op verder te werken is uitgelegd.
De belangrijkste concepten van RIDL die nodig zijn om een uitbreiding te maken zijn het doorlopen van het conceptueel schema en zo een referentiepad opstellen en het weergeven van die resultaten.

### 4.3.2 Een uitbreiding van RIDL

Vertrekkend van de mogelijkheden binnen RIDL willen we nu ook zinnen kunnen definiëren die gelijkaardige queries als de reeds bestaande zinnen uitvoeren, maar waarbij het mogelijk moet zijn om informatie toe te voegen aan de databank. Deze databank moet dan natuurlijk ontstaan zijn uit het conceptueel schema waaruit we de queries halen. Met de huidige queries kan men enkel informatie halen uit deze databank.

Om de resultaten op te vragen binnen RIDL-zinnen wordt gebruik gemaakt van het werkwoord “List”. “List” geeft de lijst van resultaten terug na het uitvoeren van de query die erop volgt.

Bij het toevoegen van informatie kunnen we dan ook logischerwijze starten met het werkwoord “Insert”.

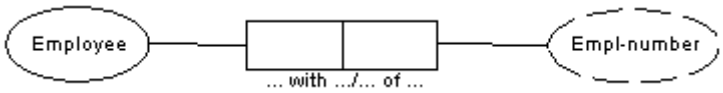
Een voorbeeld:

> “Insert Unit with Unit-name ‘a-name’.”
>
> Hierbij zou het de bedoeling zijn dat een nieuwe Unit met Unit-name ‘a-name’ aan de databank wordt toegevoegd.

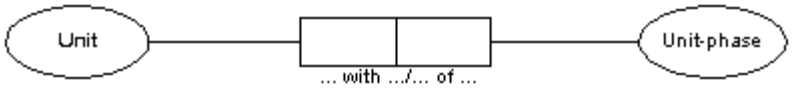
Net zoals in RIDL moeten er bepaalde regels worden opgelegd over de volgorde van Roles en Object Types:

- Na de Insert volgt altijd een Niet-Lexicaal Object Type (NOLOT).
- Een Role in een Insert wordt altijd voorafgegegaan door een Niet-Lexicaal Object Type waarmee het in relatie is.
- Een Role wordt altijd gevolgd door ofwel een Lexicaal Object Type (LOT) ofwel een procedure oproep waarbij de procedure een nieuwe Insert-definitie voorstelt. (zie verder voor meer informatie in verband met deze procedures).
- Een Lexicaal Object Type moet gevolgd worden door een variabele, die als nieuw element in de databank zal worden toegevoegd.

In figuur 4.11 en figuur 4.12 zien we wanneer het nodig is om een procedure-oproep te gebruiken en wanneer een variabele.




Figuur 4.11: Een relatie met een LOT. Bij een dergelijke situatie maken we gebruik van een variabele om in te voeren.




Figuur 4.12: Een relatie met een NOLOT. In dit geval gebruiken we, bij de invoer van een Unit, een procedure-oproep, om de invoer van Unit-phase op te roepen.

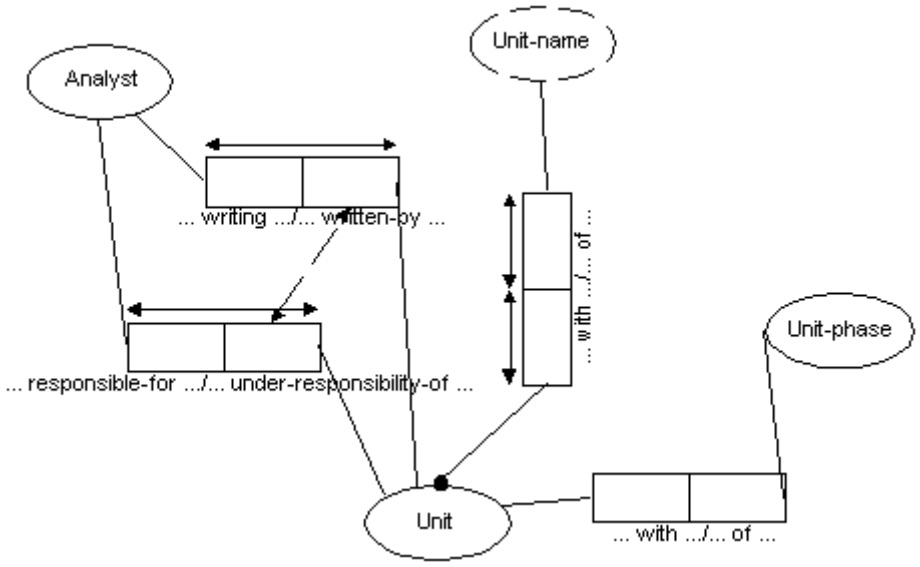
Het gebruiken van procedure oproepen binnen een Insert kunnen we zien als een soort verwijzing, waarbij de gegevens binnen de procedure in dit geval ook moeten ingevoerd worden in de databank.
Door middel van een voorbeeld zullen we de werkwijze hiervan verklaren:

“Insert Unit written-by Insert-Analyst.”

“Procedure Insert-Analyst
Insert Analyst named Empl-name ‘a-name’.”

De procedure Insert-Analyst zorgt ervoor dat er een nieuwe Analyst aan de databank wordt toevoegd met ‘a-name’ als Empl-name.
De query die de procedure oproept zorgt er dan voor dat de Insert binnen de procedure uitgevoerd wordt.

Bij het uitvoeren van de query in dit voorbeeld zouden we naast het invoeren van een nieuwe Unit ook een nieuwe Analyst invoeren.
Het moet natuurlijk ook mogelijk zijn om binnen een Insert meerdere elementen van een Object Type toe te voegen. Een voorwaarde hierbij is dat alle Roles in relatie moeten zijn met het Niet-Lexicaal Object Type dat er aan vooraf gaat. Informeel kunnen we zeggen dat we in een Insert meerdere informatie over éénzelfde Object Type willen toevoegen.
Indien we kijken naar het conceptueel schema zien we dat er veel elementen zijn waar we meerdere informatie in één keer willen invoeren. Zie ook figuur 4.13.




Figuur 4.13: een NOLOT (Unit) is in relatie met meerdere andere NOLOT’s (Analyst en Unit-phase) en/of meerdere LOT’s (Unit-name).

Tussen de verschillende Role-LOT of Role-Procedure-combinaties plaatsen we een “;” om aan te geven dat het om een nieuwe combinatie gaat.

Een voorbeeld omtrent het gebruik van meerdere combinaties:

“Insert Employee
with Empl-number ‘a-number’;
named Empl-name ‘a-name’.”

Hierbij voegen we een nieuwe Employee toe met zowel een name als een number als variabelen bij de Employee.

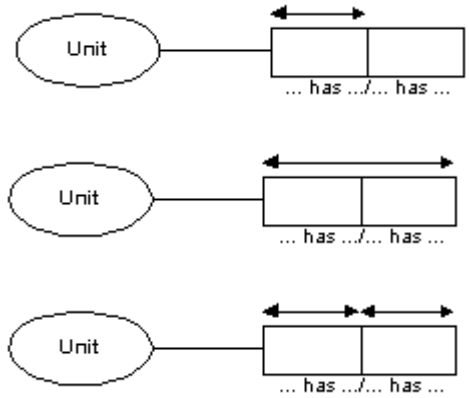
“Insert Unit
with Unit-name ‘a-name’;
written-by Insert-Analyst;
with Insert-Unit-phase.”

“Procedure Insert-Analyst
Insert Analyst named Empl-name ‘a-name’.”

“Procedure Insert-Unit-phase
Insert Unit-phase with Phase-type ‘a-type’.”

In deze query wordt een nieuwe Unit toegevoegd met als variabelen: een Unit-name, een Analyst en een Unit-phase die hierbij ingevoegd worden.

Bij het uitvoeren van queries moet het ook mogelijk zijn om bepaalde informatie meer dan eens in te voeren. We willen bijvoorbeeld meerdere Unit-phases invoeren die bij een bepaalde Unit horen tijdens één Insert.
Afhankelijk van het conceptueel schema mogen bepaalde relaties tussen Object Types meer dan eens voorkomen, terwijl dat bij andere maximaal één kan zijn.
Vermits dit in het conceptueel schema duidelijk bepaald is, moet dit ook binnen de Update-taal duidelijk aangegeven worden.
We kunnen makkelijk uit het conceptueel schema afleiden welke “uniqueness constraints” [Hal 99] op de Roles aanwezig zijn.
De verschillende mogelijkheden zijn weergegeven in figuur 4.14.




Figuur 4.14: Een aantal verschillende relatiesituaties. Op de bovenste figuur hebben we te maken met een “Uniqueness constraint”, zodat we een one-to-many relatie hebben.
Op de middelste figuur krijgen we dan een voorbeeld van een many-to-many relatie te zien.
De onderste figuur geeft de conceptuele constraints van een one-to-one relatie weer.

Hebben we te maken met een “uniqueness constraint” op de Role, zodat we een one-to-one of een one-to-many relatie hebben, dan moet de invoer SINGLE zijn.
In het andere geval hebben we een many-to-one of een many-to-many relatie waarbij we dan ook meerdere elementen kunnen invoeren door te MULTIPLE optie te nemen.

Dit gebeurt door na de variabelen of procedure oproepen aan te geven of het om een SINGLE of MULTIPLE Insertion gaat. Bij SINGLE is het aantal toevoegingen maximaal één terwijl dit bij MULTIPLE onbeperkt is.

Een voorbeeld om alles te verduidelijken:

“Insert Unit
with Unit-name ‘a-name’ SINGLE;
written-by Insert-Analyst MULTIPLE.”

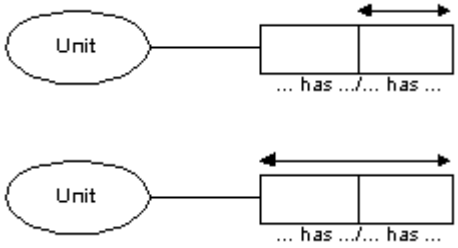
In dit voorbeeld kan Unit slechts één Unit-name hebben, maar kan ze wel door meerdere Analysten worden geschreven. Indien er geen specifieke aanduiding van SINGLE of MULTIPLE is gegeven, wordt SINGLE genomen. Dit zou dus ook correct zijn:

“Insert Unit
with Unit-name ‘a-name’;
written-by Insert-Analyst MULTIPLE.”

Deze schrijfwijze wordt echter afgeraden om misverstanden te vermijden.

Een volgend punt, waarmee rekening moet worden gehouden, is het feit dat er bij het toevoegen van bepaalde informatie ook gegevens gekozen moeten kunnen worden uit reeds bestaande informatie uit de databank.
Bij het toevoegen van een nieuw Project moet men niet noodzakelijk een nieuwe Projectleider hebben, maar moet men er één kunnen aanduiden die reeds bestaat.
Met andere woorden, we moeten ook gegevens kunnen selecteren die reeds in de databank aanwezig zijn.

Ook dit is duidelijk afleidbaar uit het conceptueel schema. Deze keer zijn de mogelijkheden afhankelijk van de “Companion Role”. Ook hier wordt alles verduidelijkt via figuur 4.15.




Figuur 4.15: Relatiesituaties afhankelijk van de CoRole. De constraints op de CoRole zijn van belang. In het eerste geval is er een “uniqueness constraint” aanwezig, in het tweede geval is dit niet zo.

Indien er een “uniqueness constraint” op de CoRole aanwezig is (waardoor we een many-to-one of een one-to-one relation krijgen) kan er enkel informatie toegevoegd worden, niet geselecteerd.
In het geval er geen “uniqueness constraint” aanwezig is moet de gebruiker zowel de keus krijgen om informatie toe te voegen als informatie te selecteren.

Om de verschillende keuzes mogelijk te maken kunnen we na de Role een “SELECT Object-Type” kiezen in plaats van een procedure oproep of een LOT. Daardoor geven we aan dat er een selectie in plaats van een toevoeging moet plaats vinden. Bij het effectief implementeren zal men dan een query doen op de databank om de gegevens op te vragen waaruit de gebruiker zal kunnen selecteren.
Belangrijk hierbij is dat het Object Type, net zoals bij een Insert, via de Role in relatie moet zijn met het Object Type dat voor de Role staat.

Een voorbeeld:

“Insert Unit
with Unit-name ‘a-name’ SINGLE;
written-by SELECT Analyst MULTIPLE.”

Hierbij kan de gebruiker bij het toevoegen van een Unit een nieuwe naam voor de Unit invoeren, maar kan hij geen nieuwe Analysten invoeren. Hij zal deze enkel kunnen selecteren uit de reeds bestaande Analysten. In de query staat ook dat we een MULTIPLE selectie hebben, dus kan de gebruikers meerdere Analysten selecteren.

In sommige gevallen zal de gebruiker niet enkel de mogelijkheid willen hebben ofwel om gegevens te selecteren ofwel om nieuwe gegevens in te voeren, maar beide. De gebruiker zal dus zowel nieuwe informatie willen toevoegen als andere, reeds bestaande informatie, willen selecteren.
Daarom moeten we een Select en een Insert ook kunnen combineren. Dit doen we door een “OR” tussen de twee te plaatsen.

Zo krijgen we bijvoorbeeld:

“Insert Unit
with Unit-name ‘a-name’ SINGLE;
written-by Insert-Analyst OR Select Analyst MULTIPLE;
under-responsibility-of Insert-Analyst
OR Select Analyst MULTIPLE.”

“Procedure Insert-Analyst
Insert Analyst
named Empl-name ‘a-name’ SINGLE;
with Empl-number ‘a-number’ SINGLE.”

Zoals duidelijk blijkt uit dit voorbeeld zal men bij het toevoegen van een Unit zowel bestaande Analysten kunnen selecteren als nieuwe Analysten kunnen toevoegen.

De MULTIPLE of SINGLE slaat, in het geval een “OR” gebruikt wordt, op de gehele lijn. Daarmee wordt het volgende bedoeld:

- Indien MULTIPLE kan de gebruiker zowel meerdere gegevens toevoegen als meerdere gegevens selecteren (zoals men in het voorbeeld meerdere Analysten kan invoeren en selecteren).
- Indien SINGLE kan de gebruiker maximaal één gegeven ofwel toevoegen ofwel selecteren. De “OR” in dit geval functioneert eigenlijk als een “XOR (exclusieve OR)”, waarbij men ofwel het ene, ofwel het andere kan kiezen, maar niet allebei. Dit is echter niet expliciet zo gegeven.

In het volgende voorbeeld krijgen we een situatie waarbij de gebruiker slechts de keuze mag hebben tussen het toevoegen of selecteren van informatie, maar niet allebei:

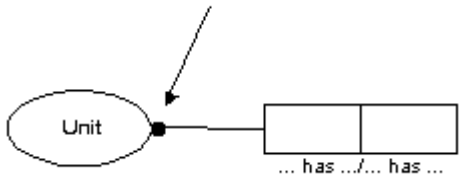
"Insert Project
with Project-name 'a-name' SINGLE;
led-by Insert-Projectleader OR Select Projectleader SINGLE;
realised-by Insert-Analyst OR Select Analyst MULTIPLE."

"Procedure Insert-Projectleader
named Empl-name 'a-name' SINGLE;
with Empl-number 'a-number' SINGLE."

De procedure Insert-Analyst is bij dit voorbeeld gelijk aan dezelfde procedure uit voorgaand voorbeeld. Verder mag de gebruiker slechts de mogelijkheid krijgen om een Projectleider toe te voegen of te selecteren.

Het laatste belangrijke punt in de uitwerking van de Insert is het feit dat bij het toevoegen van informatie sommige gegevens verplicht zijn terwijl andere informatie eerder optioneel is.
Bij het invoeren van een nieuwe Unit moeten we bijvoorbeeld een Unit-name aan die Unit geven, terwijl we niet onmiddellijk verplicht zijn om aan te geven welke Analysten de Unit schrijven. Dit kan dan later ingevuld worden.

Het is duidelijk af te leiden uit het conceptueel schema welke informatie al dan niet optioneel is, gezien dit gegeven is door een constraint op de Role te leggen. Dit attribuut, de "mandatory role constraint" (zie figuur 4.16), geeft aan of een Role al dan niet verplicht is. Indien de constraint niet aanwezig is, is de Role optioneel.




Figuur 4.16: een "mandatory constraint".

Bij het toevoegen van informatie zal de gebruiker bepaalde informatie moeten invullen om foutmeldingen te voorkomen.
Om dit nu duidelijk te maken binnen de uitbreiding van RIDL wordt er een "*" geplaatst voor iedere Role die "mandatory" of verplicht is.

Een voorbeeld uit het conceptueel schema:

“Insert Project
* with Project-name ‘a-name’ SINGLE;
* led-by Insert-Projectleader
OR Select Projectleader SINGLE;
* implementing Insert-System SINGLE;
realised-by Insert-Analyst
OR Select Analyst MULTIPLE.”

Hierbij zijn de eerste drie Roles verplicht terwijl de laatste slechts optioneel is. Indien we dit verder willen uitwerken, moeten we ook nog de procedures definiëren die in de query opgeroepen worden:

“Procedure Insert-Projectleader
Insert Projectleader
named Empl-name ‘a-name’ SINGLE;
* with Empl-number ‘a-number’ SINGLE.”

“Procedure Insert-Analyst
Insert Analyst
named Empl-name ‘a-name’ SINGLE;
* with Empl-number ‘a-number’ SINGLE.”

“Procedure Insert-System
Insert System
* with Unit-name ‘a-name’ SINGLE;
written-by Insert-Analyst OR
Select Analyst MULTIPLE;
under-responsibility-of Insert-Analyst OR
Select Analyst MULTIPLE;
with Insert-Unit-phase MULTIPLE.”

“Procedure Insert-Unit-phase
Insert Unit-phase
* with Phase-type ‘a-type’ OR
SELECT Phase-type SINGLE;
Started-by Insert-Date SINGLE.”

“Procedure Insert-Date
Insert Date
* with Year ‘a-year’ SINGLE;
* with Month ‘a-month’ SINGLE;
* with Day ‘a-day’ SINGLE.”

### 4.3.3 Waarom is dit geen definitieve oplossing?

Alhoewel deze oplossing goed werkt zijn er een aantal redenen die er voor gezorgd hebben dat deze oplossing niet als definitief werd beschouwd.
Enkele van deze redenen zijn onder andere:

- Terwijl we tegenwoordig alles proberen om te zetten in grafische representaties, zouden we hier weer naar een volledig tekstuele oplossing gaan.
  Vermits het overgrote deel van WSDM reeds volledig grafisch is, willen we ook deze oplossing op een grafische wijze zien. Verder zullen ontwerpers bij het kiezen van een methode bijna altijd kiezen voor een grafische versie.
- Deze oplossing zal wellicht nooit binnen RIDL geïntegreerd worden, vooral omdat er een groot deel aan nieuwe syntax zou bijkomen. Er zouden dan ook heel wat regels moeten opgesteld worden om het gebruik van die syntax eventueel te kunnen uitsluiten binnen de gewone queries.

Om het verder af te werken zouden we ook nog update- en delete queries moeten opstellen. Daarnaast zouden we moeten aangeven dat we verschillende queries zouden kunnen opstellen voor alle gebruikersklassen, zodat we een onderscheid zouden kunnen maken welke gebruikers welke informatie zouden kunnen invoeren, wijzigen of verwijderen.
Dit is bewust niet verder uitgewerkt, omdat we na de invoer queries zagen dat er nog een derde en betere oplossing voorhanden is. Deze zal volledig worden uitgelegd in het hierop volgend deel.

## 4.4 Definitieve oplossing: uitbreiding van de ORM (Object Role Modelling) notatie

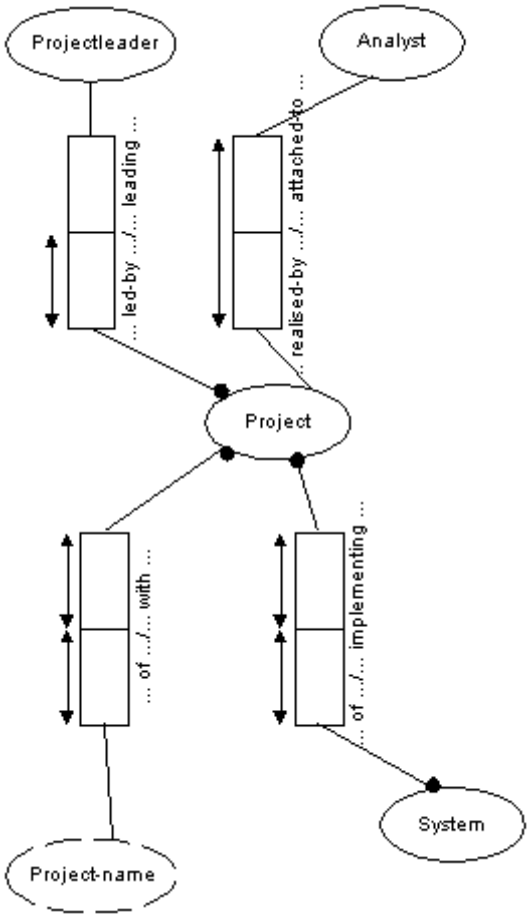
Na het uitbereiden van RIDL met invoerqueries, waar we tot de conclusie kwamen dat het nog beter kan, geven we hier een oplossing die enkel gebaseerd is op ORM [Orm, Hal 99].

Ditmaal zijn we gestart vanuit voorgaande oplossing, waarin het al duidelijk bleek dat alle opgestelde regels in de RIDL-uitbreiding rekening moesten houden met de opgelegde constraints uit het conceptueel schema. We willen nu, in plaats van iets af te leiden uit het schema, het schema hergebruiken.

### 4.4.1 Een omzetting van de RIDL-uitbreiding naar ORM

Beschouwen we volgende invoerquery uit RIDL samen met het bijhorende conceptuele schema:

"Insert Project
* with Project-name 'a-name' SINGLE;
* led-by Insert-Projectleader
OR Select Projectleader SINGLE;
* implementing Insert-System SINGLE;
realised-by Insert-Analyst
OR Select Analyst MULTIPLE."




Figuur 4.17: het conceptuele schema horend bij de RIDL-query.

Dan merken we dat we de RIDL invoerquery grotendeels kunnen afleiden uit het schema. Zowel het invoeren van de "Project-name", de "Projectleader" en het "System" zijn verplicht in de invoerquery. Dit is afkomstig van de "mandatory constraints" die op de Roles in de relatie aanwezig zijn.

Hetzelfde geldt voor de “uniqueness constraints” op de Roles. Telkens er een “uniqueness constraint” aanwezig is op de Role, hebben we te maken met een SINGLE invoer. In het andere geval, zoals bij de Analyst te zien is, hebben we een MULTIPLE invoer.
Indien er dan nog een “uniqueness constraint” en een “mandatory constraint” op de CoRole aanwezig zijn kunnen we enkel informatie invoeren en niet selecteren. Dit is het geval bij de ‘Project-name’ en het ‘System’.
Indien er enkel een “uniqueness constraint” op de CoRole aanwezig is, mag er wel informatie ingevoerd worden maar moet er een controle plaatsvinden die nagaat of de “constraint” geldig blijft.

Hieruit blijkt duidelijk dat de RIDL-queries tekstuele representaties zijn van delen van het conceptuele schema waaraan informatie is toegevoegd die aangeeft hoe nieuwe gegevens kunnen ingevoerd worden.
We willen deze situatie echter omkeren. Door in het conceptuele schema een duidelijke notatie toe te voegen die aangeeft welke gegevens al dan niet ingevoerd moeten worden, kunnen we een grafische representatie weergeven van RIDL-invoerqueries.
Later zou het dan natuurlijk de bedoeling zijn dat we bij het modelleren van de informatie ook direct modellen toevoegen van de informatie die in de databank moet toegevoegd worden.

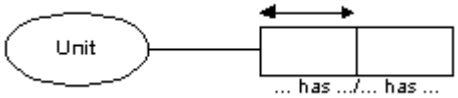
Net zoals we in de “Information Modelling” stap van WSDM conceptuele schema’s uitwerken voor de verschillende gebruikersklassen, zullen de gebruikersklassen nu ook verschillende updatemodellen hebben. Die verschillende modellen zullen dan afhankelijk zijn van de autorisaties die zij hebben.
Niet iedere gebruikersklasse zal bijvoorbeeld het recht hebben om informatie toe te voegen, te wijzigen of te verwijderen. Net zoals niet iedere gebruikersklasse het recht heeft om bepaalde informatie te zien.

Indien we dergelijke oplossing kunnen creëren, die bovendien gebruiksvriendelijk is en makkelijk in de omgang, dan hoeven we ook niks te wijzigen in de werkwijze van WSDM. De oplossing hoeft gewoon toegevoegd te worden aan de "Information Modelling Step” binnen WSDM. Ook het autorisatieprobleem zou op die wijze onmiddellijk van de baan zijn.

In het volgende onderdeel geven we een uitleg over de toegevoegde notatie aan ORM. Deze notatie wordt gebruikt om alles duidelijk te maken voor de ontwerper.
Verder kan het ook misverstanden verhinderen tussen de ontwerper en andere ontwikkelaars van de web site. Door het plaatsen van enige notatie wordt het namelijk duidelijk dat we te maken zullen hebben met

invoermodellen in plaats van de gewone web modellen die enkel informatie weergeven.

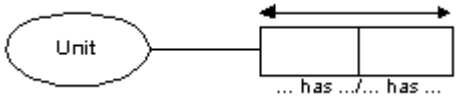
Een andere reden waarom de notatie wordt toegevoegd is de volgende:
De ontwerper kan via de notatie beslissingen nemen die niet volledig af te leiden zijn uit de constraints van het conceptueel schema. Het is wel zo dat de ontwerper in principe de constraints, die minimum en maximumregels opleggen, moet volgen, maar er zijn daarbinnen variaties mogelijk.
De variaties die hier uitgelegd worden, zijn eveneens geldig voor het wijzigen en verwijderen van informatie.
De mogelijkheden voor de ontwerper zijn de volgende:




Figuur 4.18: De “uniqueness constraint” op de linker Role.

Volgens de “uniqueness constraint” kan de gebruiker hier maximum één relatie toevoegen. Dit toevoegen kan gebeuren door nieuwe informatie toe te voegen of bestaande informatie te selecteren.
De ontwerper kan echter beslissen om enkel nieuwe informatie toe te voegen, of enkel informatie te selecteren, maar niet beide keuzes te laten. De ontwerper mag zelfs beslissen om van dit “Object Type” geen informatie te laten toevoegen op deze plaats (vermits er geen “mandatory constraint” aanwezig is), zodat deze informatie dan op een andere plaats zal toegevoegd worden.

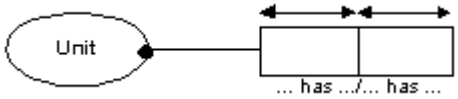
Over het algemeen kan de ontwerper dus mogelijkheden wijzigen indien hij volgens het conceptueel model de keuze zou moeten geven tussen selecteren van informatie en invoeren van nieuwe informatie. Daar kan de ontwerper dan de beslissing nemen om enkel één van beide mogelijkheden te gebruiken. Ook in het volgende voorbeeld hebben we met dezelfde situatie te maken:




Figuur 4.19: een relatie zonder “uniqueness constraints”.

Ook hier krijgen we dezelfde situatie als de vorige. De ontwerper kan beslissen om enkel informatie te selecteren of enkel nieuwe informatie toe te voegen, terwijl volgens het conceptueel model beide mogelijkheden beschikbaar zouden moeten zijn.

Een situatie waarin geen variaties mogelijk zijn is de volgende:



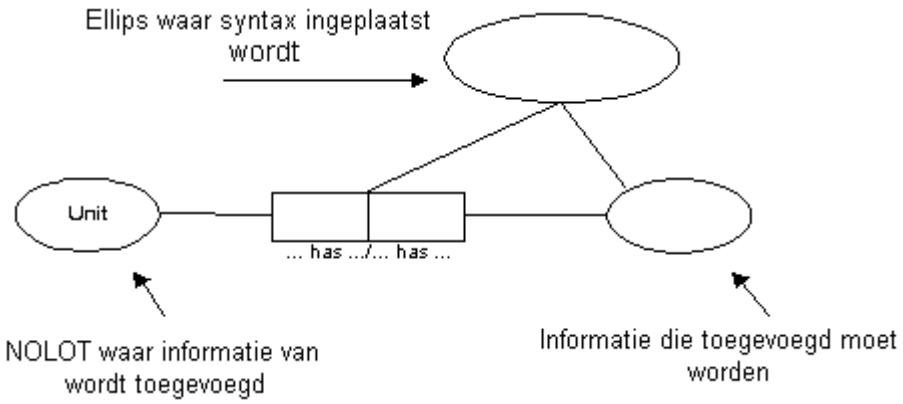

Figuur 4.20: een relatie met “uniqueness constraints” en “mandatory constraints”.

Hier zijn we verplicht om informatie in te voeren of te selecteren wegens de “mandatory constraint”.
Daarnaast mogen we maximaal één gegeven invoeren of selecteren wegens de “uniqueness constraint” op de Role (links).
We mogen dus juist één element kiezen of selecteren. Bij het selecteren van een element zal er enkel een controle moeten gebeuren die nagaat of de geselecteerde instantie nog niet in relatie is met een andere instantie.

### 4.4.2 De aan ORM toegevoegde notatie

De notatie die we toevoegen aan ORM is dus om de gebruiker een duidelijk zicht te kunnen geven in de mogelijkheden die er zijn. Door de correcte notatie op de juiste plaatsen toe te voegen zal de communicatie met de andere ontwikkelaars van de web site optimaal zijn.
De personen die de web site effectief moeten implementeren zullen via grafische representaties een duidelijk zicht krijgen op de te creëren “input-forms” en andere forms, die op de web pagina’s moeten komen.

Bij het invoeren, wijzigen en verwijderen van informatie zullen we, net zoals bij RIDL, starten vanuit een Niet-Lexicaal Object Type (NOLOT). De informatie de we willen invoegen moet in relatie zijn met het NOLOT.
We verbinden dan een aparte ellips met de Role en het Object Type waarvan we informatie willen toevoegen. In die ellips zullen we dan de geschikte notatie plaatsen die moet aangeven wat de mogelijkheden zijn voor het invoeren van de informatie. Figuur 4.21 geeft aan wat er juist bedoeld wordt met deze ellips.




Figuur 4.21: grafische weergave van een conceptueel schema, aangevuld met een ellips om regels, voor het toevoegen van informatie, in te plaatsen.

Om aan te geven welke functies er kunnen gebruikt worden, geven we hier de verschillende notaties op een rijtje:

- **?** ⫞ Single Insertion (enkelvoudige invoer)
- **??** ⫞ Multiple Insertion (meervoudige invoer)
- **!** ⫞ Single Selection (enkelvoudige selectie)
- **!!** ⫞ Multiple Selection (meervoudige selectie)
- **X** ⫞ Single Deletion van relatie en Object Type (enkelvoudige verwijdering)
- **XX** ⫞ Multiple Deletion van relatie en Object Type (meervoudige verwijdering)
- **x** ⫞ Single Deletion van relatie (enkelvoudige verwijdering)
- **xx** ⫞ Multiple Deletion van relatie (meervoudige verwijdering)
- **O** ⫞ No Insertion, Selection or Deletion (geen invoer, selectie of verwijdering)
- **[schema]**
  ⫞ oproep van ander schema

De verschillende mogelijkheden zijn waarschijnlijk duidelijk, maar toch geven we hier nog een korte beschrijving van de notaties:

- Single Insertion: het is mogelijk om maximaal één informatie-element toe te voegen.
- Multiple Insertion: meerdere toevoegingen zijn in dit geval mogelijk.
- Single Selection: in dit geval zal de gebruiker juist één element uit de reeds bestaande informatie kunnen selecteren.
- Multiple Selection: hier is het dan de bedoeling dat meerdere elementen kunnen geselecteerd worden.

- Single Deletion (relatie en object type): ook bij het deleten van informatie hebben we het verschil tussen één element en meerdere elementen verwijderen. In dit geval kunnen we slechts één element verwijderen. Hierbij wordt zowel de relatie als de instantie van het Object Type, waarmee het te verwijderen Object Type in relatie is, verwijderd.
- Multiple Deletion (relatie en Object Type): dit is dan de andere mogelijkheid waarbij meerdere elementen kunnen verwijderd worden. Ook hier worden relatie en instantie van Object Type verwijderd.
- Single Deletion (relatie): het verwijderen van exact één relatie. De instantie van het Object Type wordt hierbij niet verwijderd omdat die instantie nog in relatie kan zijn met andere elementen of omdat dit op dit moment niet gepast is.
- Multiple Deletion (relatie): het verwijderen van meerdere relaties. Ook hier enkel de relatie wegens dezelfde reden als hiervoor.
- Nothing: deze allerlaatste mogelijkheid is er omdat de gebruiker in sommige gevallen geen informatie zal willen toevoegen, wijzigen of verwijderen.
- [Schema]: Door dit toe te voegen binnen de ellips wordt aangegeven dat dit schema moet gebruikt worden om een bepaalde instantie toe te voegen. Dit kan dus enkel gebruikt worden bij het toevoegen van informatie. Daarbij moet het schema dat wordt opgeroepen dit zijn dat de instantie invoert die de ontwerper in het originele schema wil invoeren en kan deze oproep enkel plaatsvinden indien het om een Niet-Lexicaal Object Type gaat (voor een Lexicaal Object Type wordt trouwens geen apart schema opgesteld).

Bij het enkelvoudig of meervoudig invoeren, wijzigen of verwijderen van informatie is het ook zo dat er minimaal één element respectievelijk moet ingevoerd, gewijzigd of verwijderd worden. Indien de ontwerper wil dat de gebruiker ook de mogelijkheid moet hebben om geen enkel element in te voeren, te wijzigen of te verwijderen, dan kan hij gebruik maken van de laatste notatie.

Bij het verwijderen van informatie is het ook van belang om te weten dat ook relaties tussen elementen in een databank worden bijgehouden. Het verwijderen van een object uit een databank houdt dan natuurlijk in dat al de relaties uit de databank verwijderd worden.
Afhankelijk van de “constraints” op de Roles kan er nagegaan worden of zowel de relatie als de instantie moet verwijderd worden of enkel de relatie. Door dan de juiste notatie te gebruiken kan exact aangegeven worden wat er in de implementatie zal moeten gebeuren.

Het is ook vanzelfsprekend dat, indien we een instantie willen verwijderen uit de databank, naast het verwijderen van alle relaties van die instantie (eventueel met instanties van andere Object Types), die instantie zelf ook verwijderd wordt. Dit wordt in de modellen niet expliciet weergegeven omdat dit zo evident is.

Ook bij RIDL hadden we het probleem dat we de mogelijkheden moeten kunnen combineren, simpelweg omdat de constraints binnen ORM dit toelaten en omdat we de combinaties zullen nodig hebben bij het wijzigen van informatie.
Het wijzigen komt namelijk overeen met het verwijderen van informatie en andere (nieuwe of geselecteerde) informatie in de plaats zetten.

Combineren van notaties kan dan op zich ook weer op verschillende wijzen:

1. de combinaties moeten samen uitgevoerd worden (bijvoorbeeld bij het wijzigen van informatie: er moet tegelijk verwijderd en ingevoerd of tegelijk verwijderd en geselecteerd worden);
2. de combinaties mogen (maar moeten niet) samen uitgevoerd worden (bijvoorbeeld de mogelijkheid om nieuwe informatie toe te voegen en informatie te selecteren, maar ze moeten niet allebei);
3. één van beide combinaties moet uitgevoerd worden maar niet allebei samen (dit komt dan weer overeen met de “exclusive or”. Bij een “uniqueness constraint” mag er bijvoorbeeld informatie toegevoegd of geselecteerd worden, maar beide mogelijkheden samen zijn uitgesloten).

De verschillende wijzen om combinaties te vormen worden door gebruik van volgende syntax van elkaar onderscheiden:

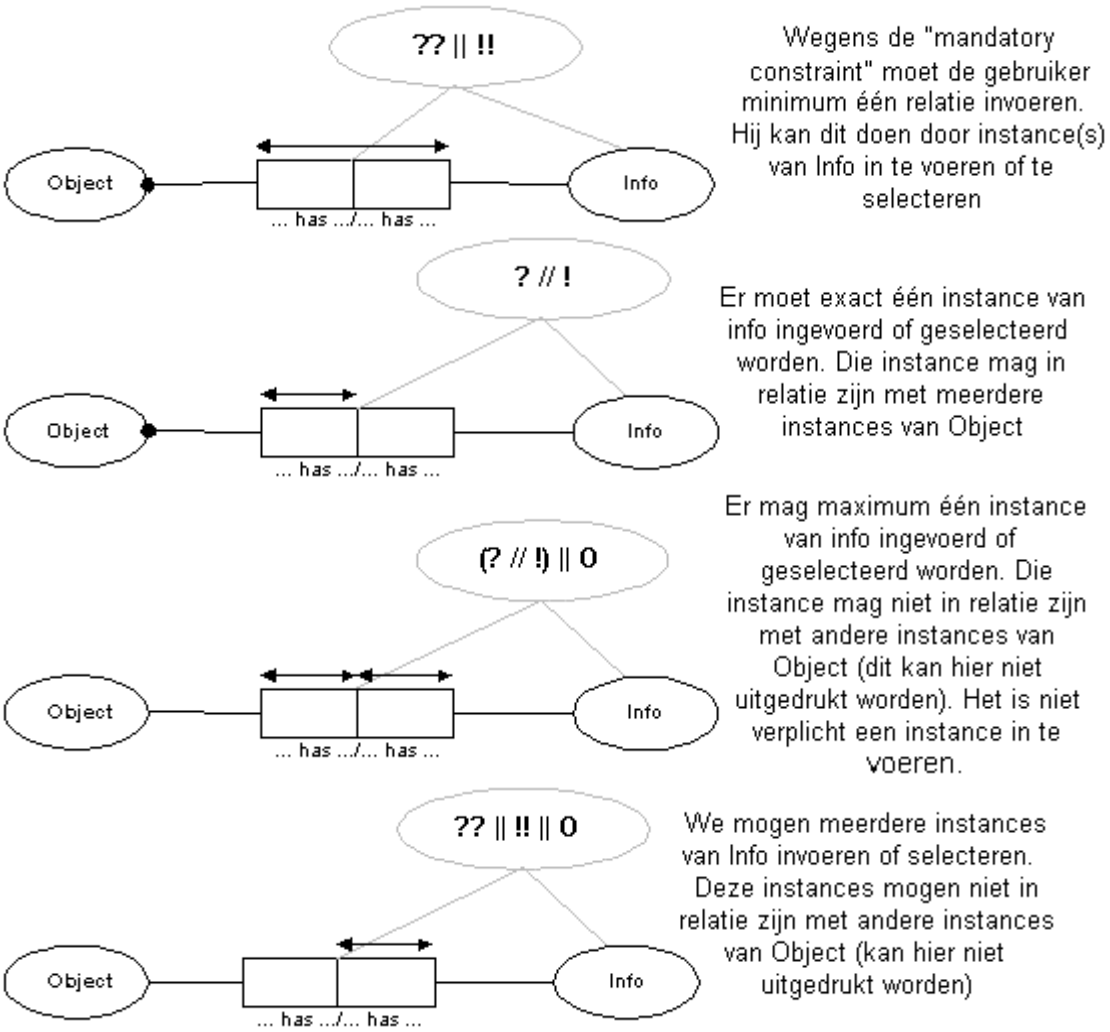
- **&&** ⫲ de combinaties verplicht samen uitvoeren;
- **||** ⫲ minimaal één van beide combinaties moet uitgevoerd worden;
- ***//*** ⫲ maximaal één van beide combinaties mag uitgevoerd worden, maar niet allebei samen.

Door middel van verschillende combinaties op te stellen aan de hand van de keuzemogelijkheden en de combinatiemogelijkheden, kunnen alle nodige instructies opgesteld worden om informatie toe te voegen, te wijzigen of te verwijderen. Er kan immers rekening worden gehouden met alle “constraints” van het conceptueel schema en de ontwerper is vrij om eventuele verdere beperkingen (zie eerder) op die “constraints” te leggen, zodat de gebruiker van de web site het juiste aantal mogelijkheden zou krijgen om iets met de informatie uit te voeren.

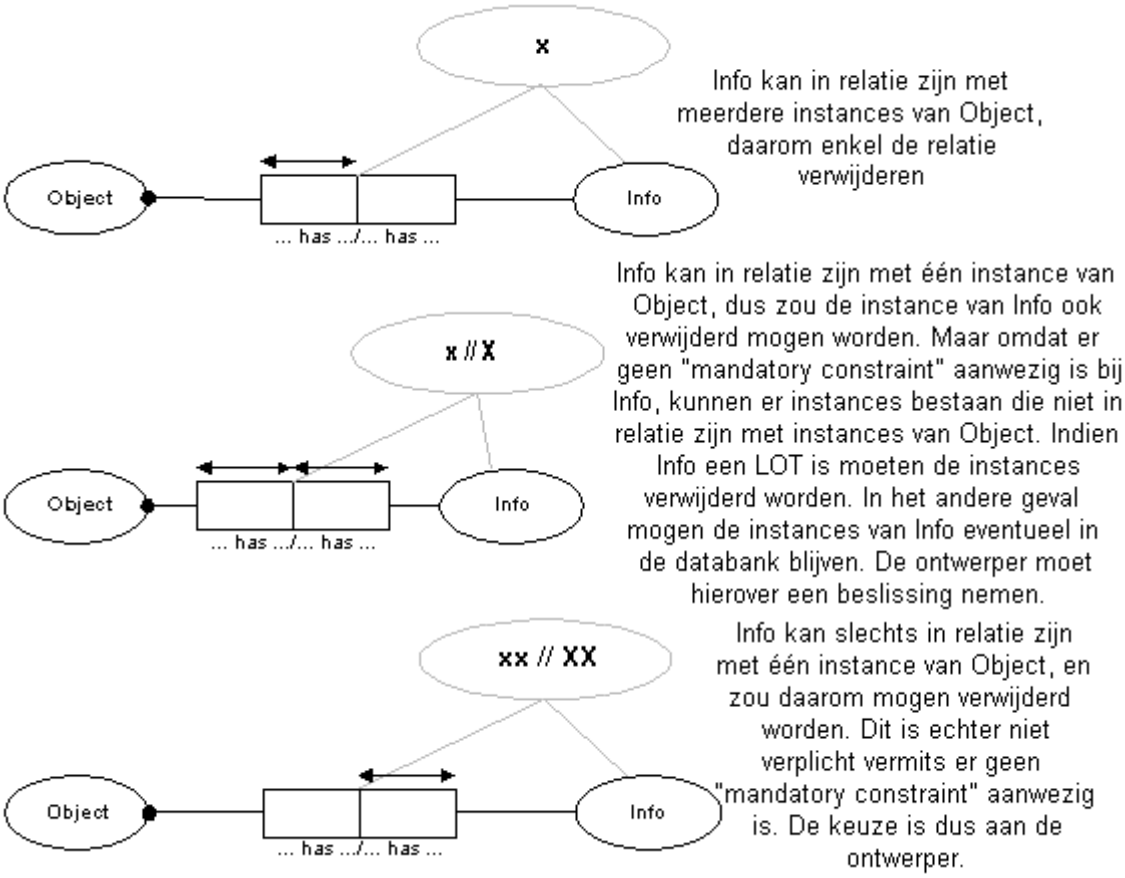
Daarbij is het ook toegelaten om haakjes te plaatsen, waar het nodig is, om de duidelijkheid te verbeteren en de verschillende combinaties goed van elkaar te kunnen onderscheiden.

De volgende voorbeelden zijn deze die standaard kunnen toegepast worden, afhankelijk van de “constraints” die op de Roles aanwezig zijn.

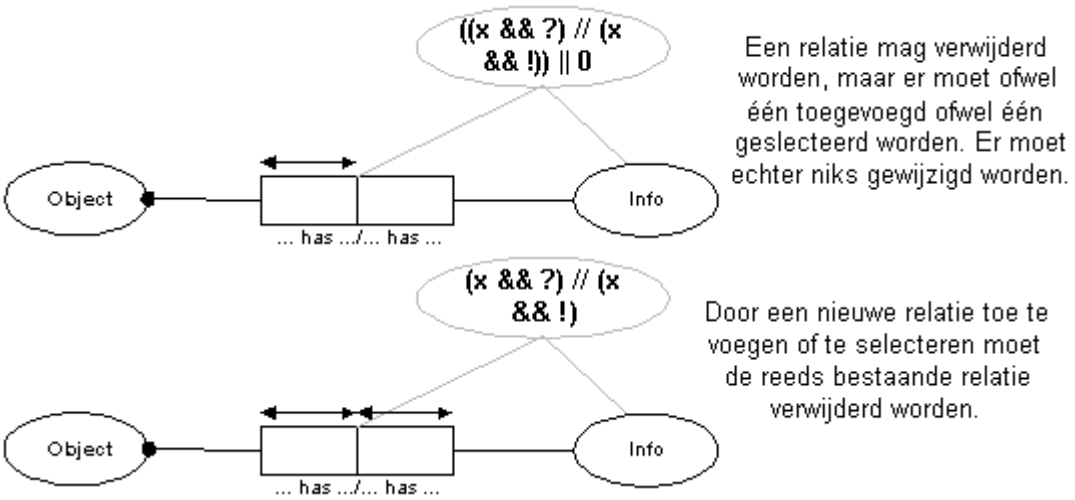
In al deze voorbeelden is “Object” het element waarover we een instantie willen toevoegen, wijzigen of verwijderen, en is “Info” die expliciete informatie.

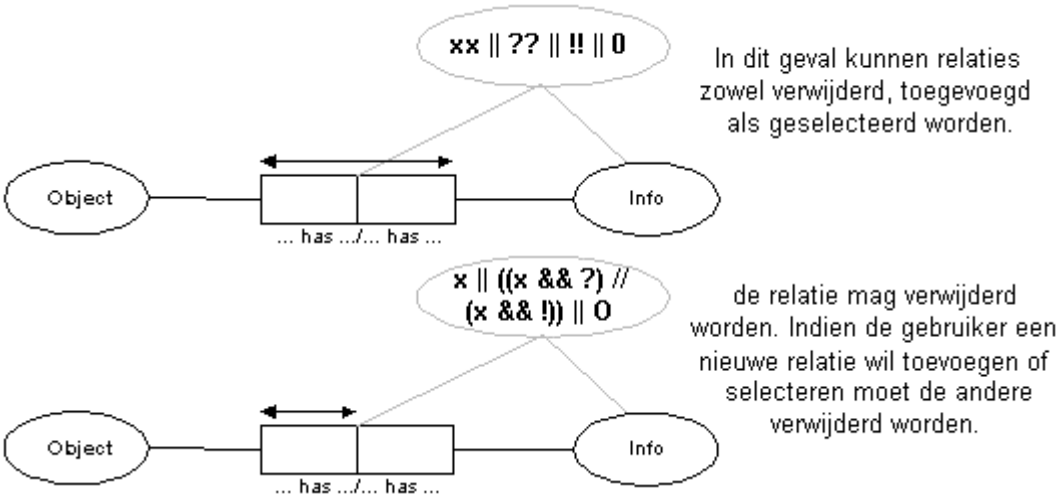



Figuur 4.22: een aantal relatiesituaties en de oplossingen voor het invoeren van informatie. Bij andere situaties krijgen we gelijkaardige resultaten.




Figuur 4.23: een aantal relatiesituaties en de oplossingen voor het verwijderen van informatie. Andere situaties kunnen gelijkaardige resultaten geven, vermits de constraints niks wijzigen aan de regels.






Figuur 4.24: een aantal relatiesituaties en de oplossingen voor het wijzigen van informatie. Ook hier bestaan er nog een aantal andere situaties die over het algemeen dezelfde resultaten hebben.

Er bestaan nog heel wat andere mogelijkheden die hier niet besproken zijn, maar de ontwerper kan deze voornamelijk afleiden uit deze die hier gegeven worden, samen met het logische redeneren en het afleiden uit het conceptueel schema.

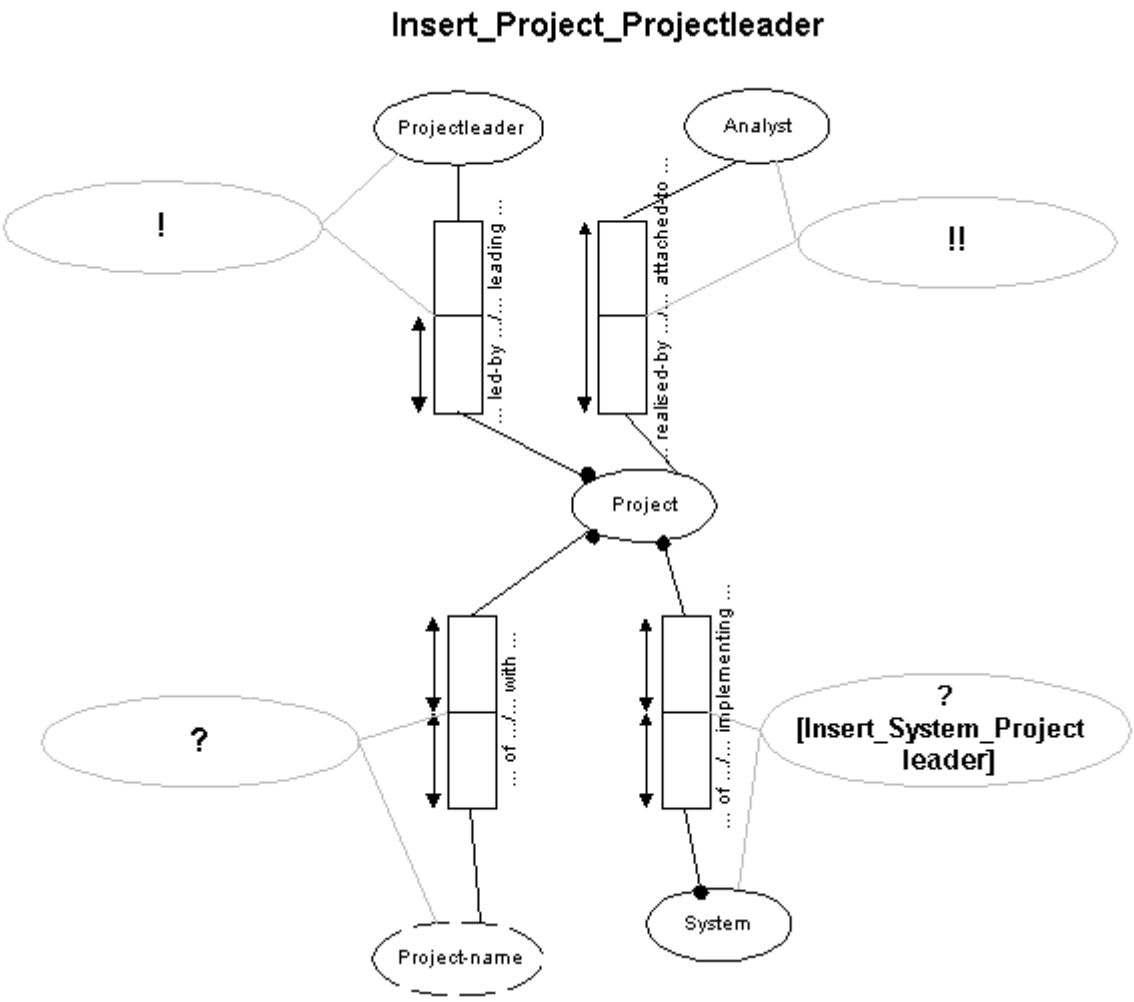
### 4.4.3 Voorbeelden

In dit deel zullen we nu, aan de hand van het voorbeeld uit het begin van dit hoofdstuk, de effectiviteit van de oplossing aangeven. Ook zal duidelijk worden hoe het opstellen van de combinaties in zijn werk gaat.

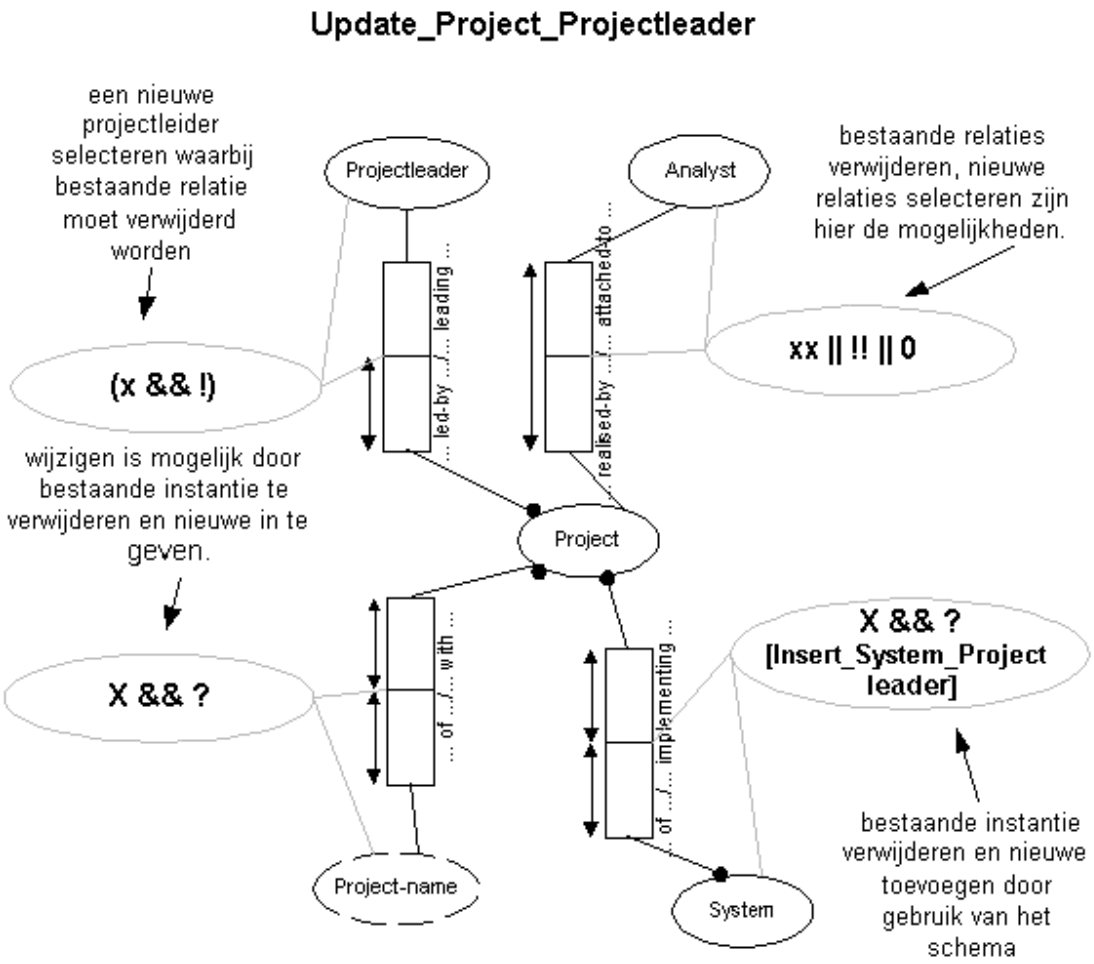
De grafische voorbeelden zullen dan ook de duidelijkheid weergeven voor de andere ontwikkelaars van de web site en meteen zal ook de gebruiksvriendelijkheid aangetoond worden.

We zullen een aantal situaties geven en de grafische weergaven ervan.

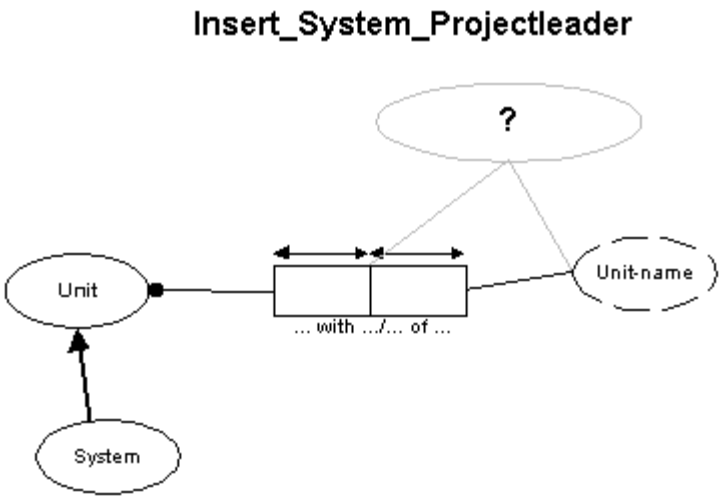
- Situatie 1: enkel de projectleiders mogen informatie toevoegen, wijzigen of verwijderen van projecten. Bij het invoeren of wijzigen van informatie van projecten mag men geen nieuwe Analysten toevoegen, enkel bestaande selecteren. De gebruiker is verplicht om minstens één Analyst te selecteren. Ook de Projectleader moet geselecteerd worden.
Bij het invoeren moet de projectleider een naam geven aan het project en moet hij een systeem invoeren zoals een Unit.
Bij het verwijderen van een project moeten ook de Project-name en het System dat in relatie is met het project effectief verwijderd worden. Bij de Projectleader en de Analysten worden enkel de relaties verwijderd.



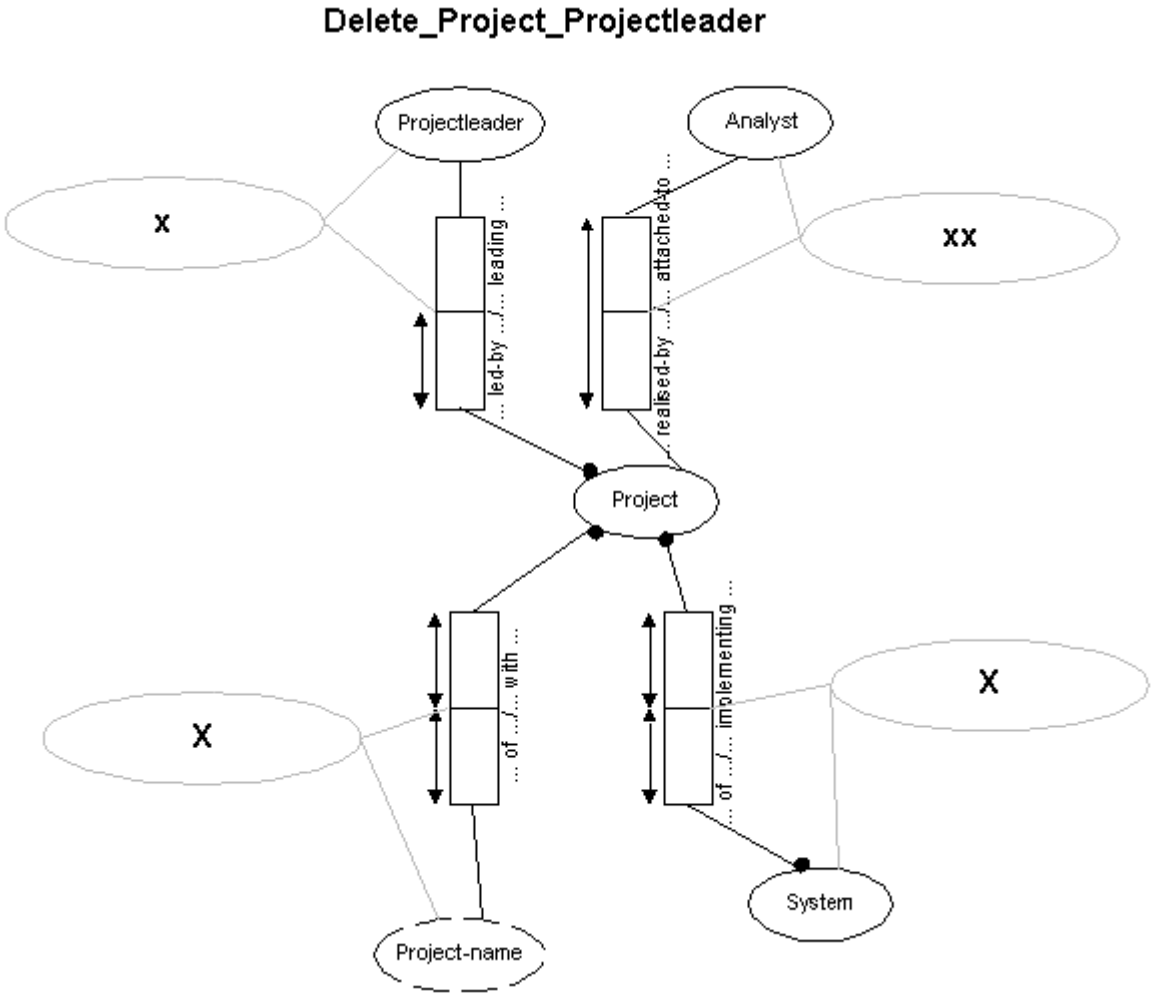

Figuur 4.25: het invoeren van een “Project” door een Projectleider.




Figuur 4.26: het wijzigen van informatie over een “Project” door een Projectleider.

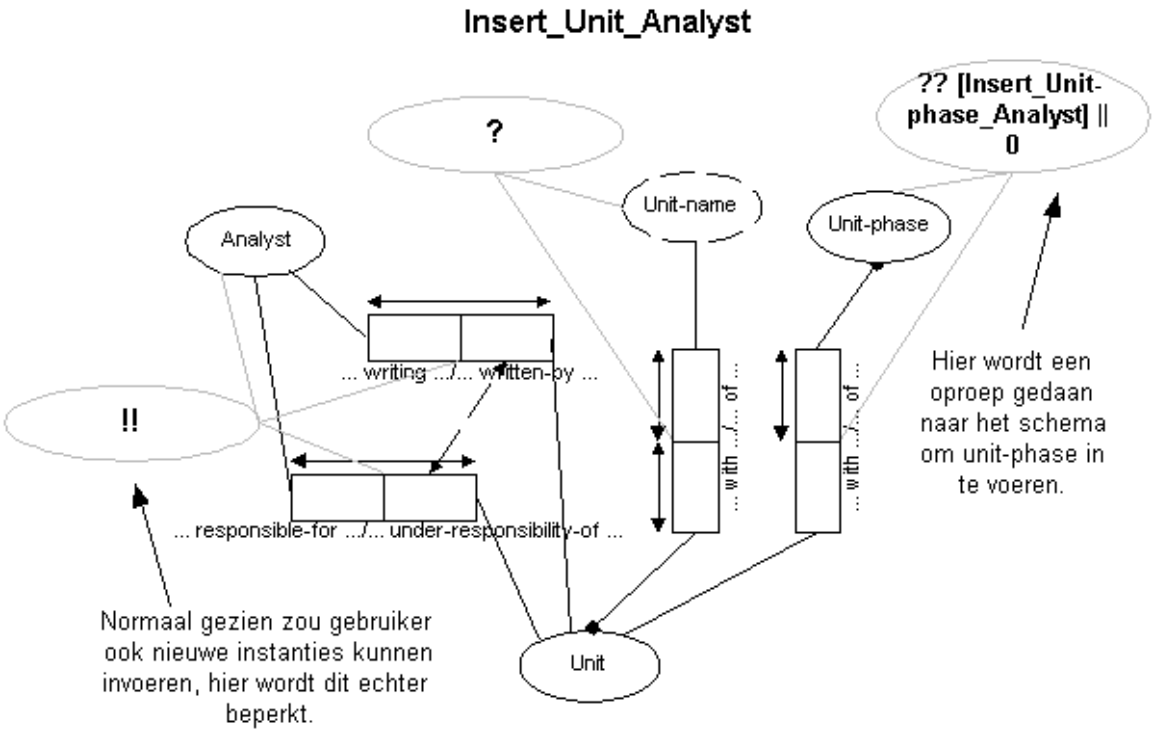



Figuur 4.27: het invoeren van een “System” door een Projectleider. Hierbij wordt enkel een naam ingevoerd voor “System”.

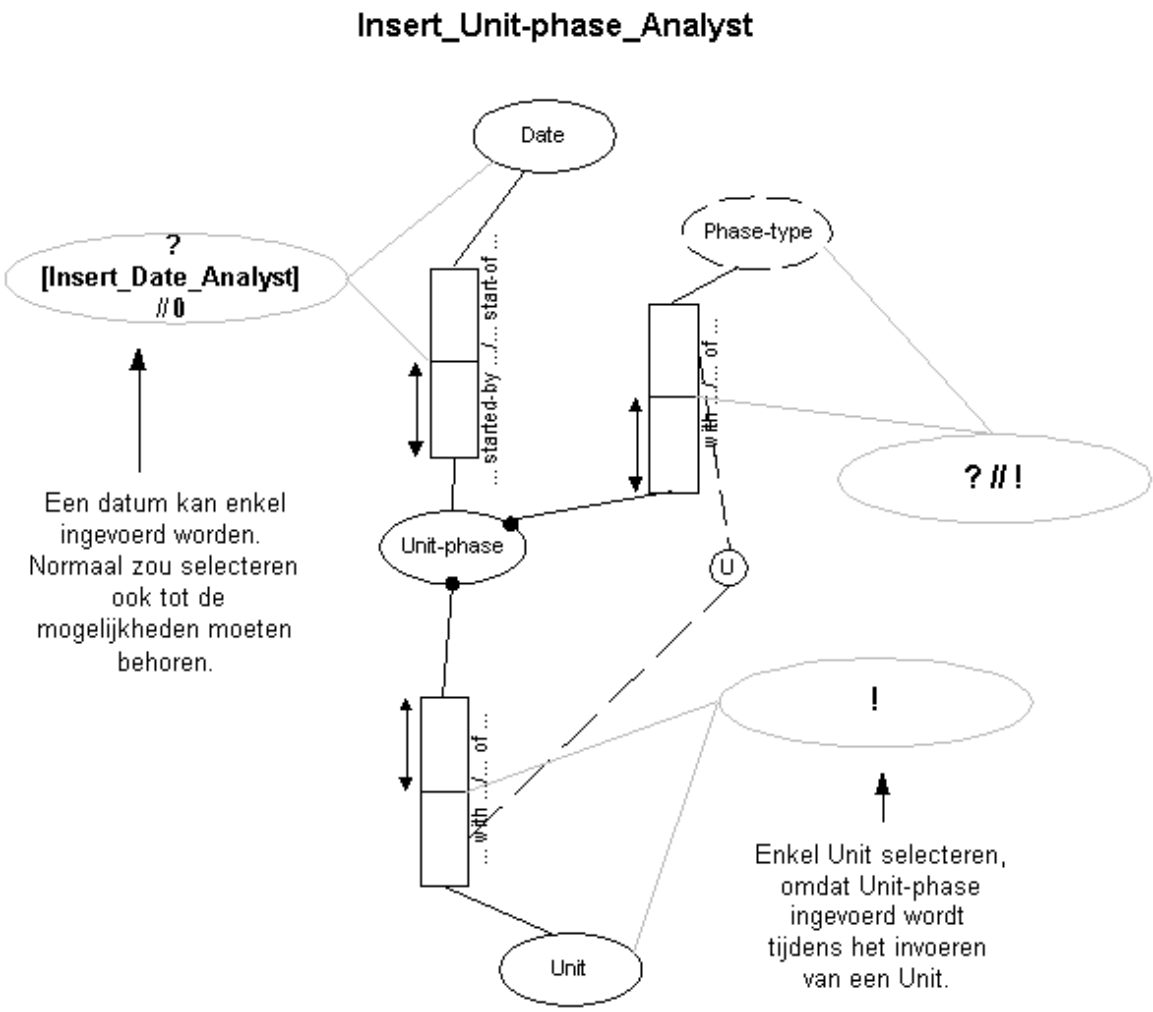



Figuur 4.28: het verwijderen van een “Project” door Projectleider.

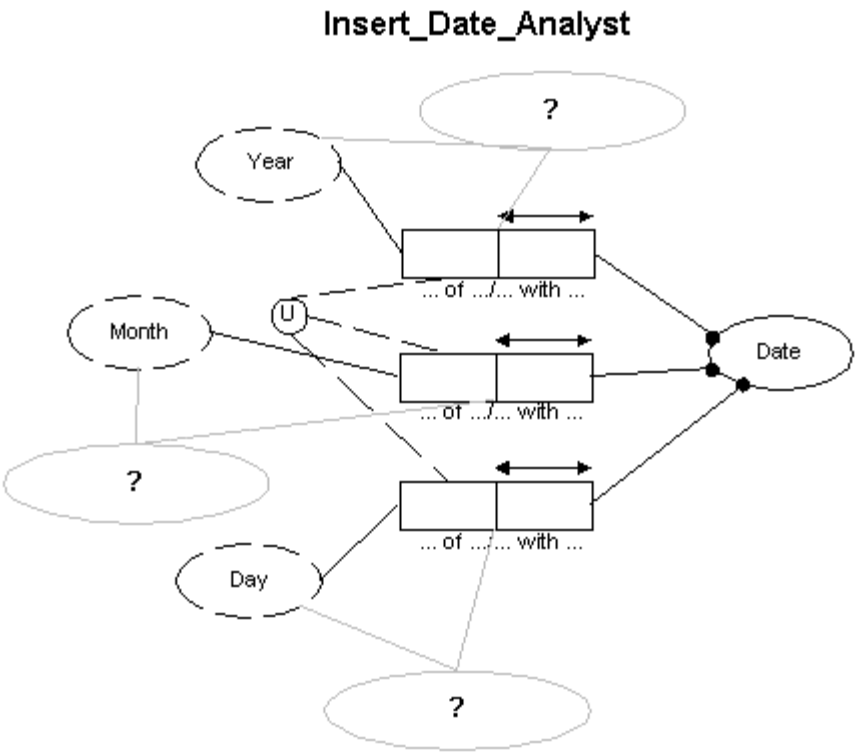
- Situatie 2: het invoeren, wijzigen en verwijderen van informatie in verband met Units gebeurd door Analysten.
Bij het invoeren of wijzigen zijn enkel relaties met bestaande Analysten mogelijk.
Bij het verwijderen van een Unit moet ook de Unit-name en de Unit-phases effectief verwijderd worden.



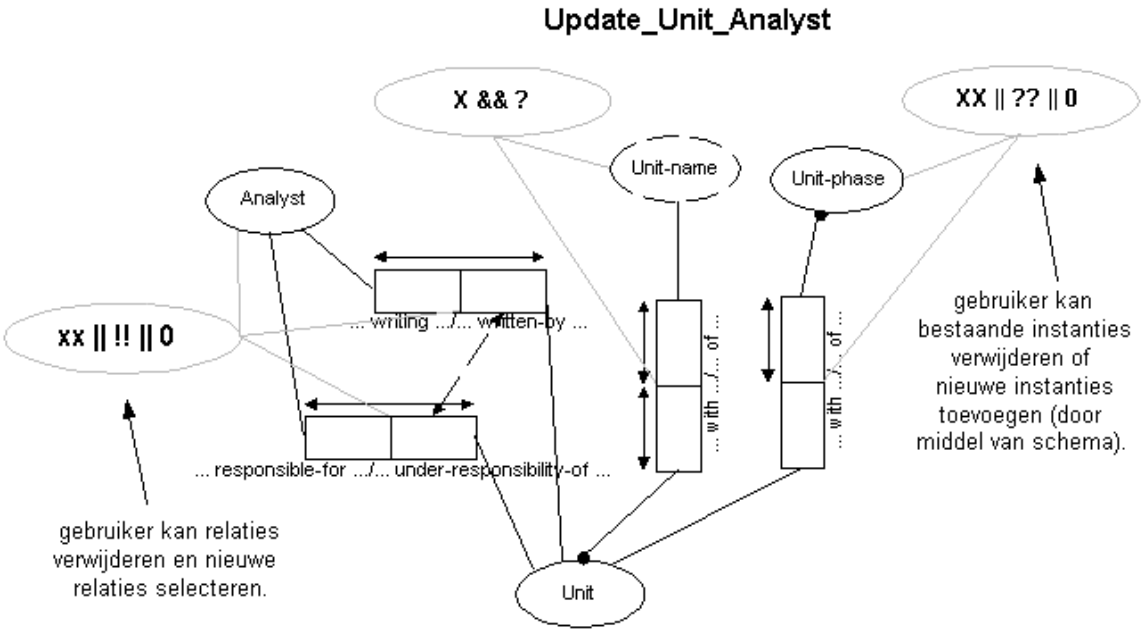

Figuur 4.29: het invoeren van informatie over “Unit”. Hierbij is ook het schema nodig om een “Unit-phase” in te voeren. Dit schema wordt in figuur 4.30 weergegeven.



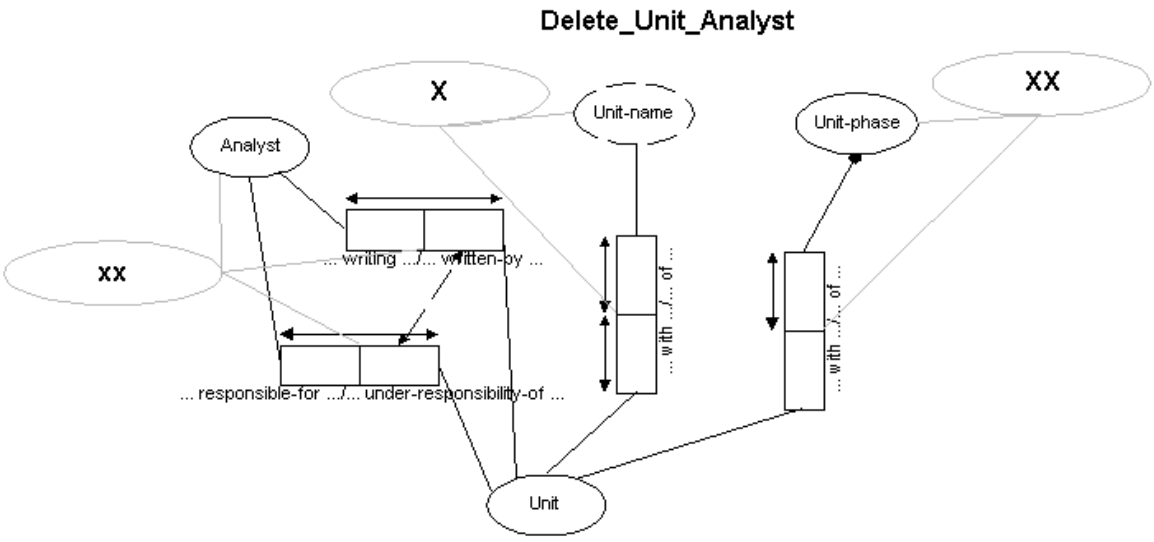

Figuur 4.30: het invoeren van een “Unit-phase” door een Analyst. Ook hier wordt een schema opgeroepen om gebruikt te worden binnen dit schema, namelijk dat om een datum in te voeren. Dit schema wordt weergegeven in figuur 4.31.




Figuur 4.31: het invoeren van een “Date” door Analyst.




Figuur 4.32: het wijzigen van informatie over een “Unit”.




Figuur 4.33: het verwijderen van informatie over een “Unit” door een Analyst.

### 4.4.4 De voor- en nadelen van deze oplossing

Een bepaalde oplossing zal altijd voor- en nadelen hebben, zo ook deze. Ditmaal wegen de nadelen echter niet op ten opzichte van de voordelen.

De nadelen van deze oplossing zijn:

- Het is afhankelijk van ORM. De oorspronkelijk doelstelling was om een oplossing te bedenken die totaal onafhankelijk zou zijn. Het probleem is dat men daarbij een volledige nieuwe oplossing zou moeten bedenken, waarbij we dan ook een extra grafische representatie zouden maken, naast het conceptueel model dat reeds in WSDM geconstrueerd wordt. De andere mogelijkheid is een reeds bestaande presentatie te hergebruiken, zoals in deze oplossing. Maar, indien een ontwerper een andere methode wil gebruiken om conceptuele schema’s op te stellen, kan deze oplossing misschien op een eenvoudige wijze aangepast en gebruikt worden binnen deze andere methode.

- De uitbreiding wordt niet ondersteund door grafische tools voor ORM.
  Dit betekent dat de ontwerper problemen zal hebben indien hij software zal gebruiken dat specifiek ontworpen is om ORM-schema’s op te stellen, zoals Infomodeler [Inf]. Deze software laat niet altijd toe om extra tekeningen (die niets met de oorspronkelijke schema’s te maken hebben) aan deze schema’s toe te voegen. Gelukkig is de meeste software aangepast, zie Visio [Vis] dat Infomodeler overgenomen heeft, zodat de ontwerper de extra’s gemakkelijk kan toevoegen.

De belangrijkste voordelen bij deze oplossing zijn de volgende:

- De verschillende mogelijkheden om aan te geven hoe informatie toe te voegen, te wijzigen of te verwijderen zijn aanwezig. Het probleem dat we hadden in RMM+, waarin enkel het invoeren mogelijk was, is hier van de baan.

- Net zoals bij het opstellen van het conceptueel schema, kunnen ook hier verschillende versies opgesteld worden voor de verschillende gebruikersklassen. Bij het toevoegen van een element zullen bepaalde gebruikers dan bijvoorbeeld slechts een deel van de informatie kunnen invoeren, terwijl andere gebruikers de volledige informatie kunnen toevoegen. Dit is natuurlijk volledig afhankelijk van de eisen die er door de applicatie op dit gebied worden gesteld, maar de mogelijkheden zijn er in ieder geval.

- Samen met vorig voordeel kunnen we stellen dat hiermee ook het autorisatieprobleem opgelost is. Voor de verschillende soorten van gebruikers kunnen we immers ook verschillende schema's opstellen die perfect weergeven wat de gebruikers zullen mogen. Daarbij moet natuurlijk nog altijd rekening worden gehouden met de "constraints" die aanwezig zijn in het conceptueel schema, maar dit is iets waarmee we steeds rekening moeten houden.

- Het is niet nodig om extra stappen aan WSDM toe te voegen. De oplossing kan samen met het opstellen van de conceptuele schema's binnen de "Information Modelling Step" geplaatst worden.
  Enkel de nieuwe notatie moet toegevoegd worden met een duidelijke handleiding zodat de gebruiker gemakkelijk de methode kan gebruiken.

### 4.4.5 Verder werk

- Het is mogelijk dat de schema's duidelijker zijn indien de ellipsen niet verbonden worden met de Roles en de Object Types, maar enkel met de Roles. Eventueel kan men het gebruik van beide mogelijkheden toelaten, afhankelijk van de situatie (bijvoorbeeld bij het verwijderen van informatie op deze manier aangeven wat effectief moet verwijderd worden – enkel de relatie of relatie en instantie).
- Indien een tool voor WSDM uitgewerkt wordt, zou ook eerst een tool kunnen opgesteld worden die controleert of de schema's die de ontwerper heeft aangemaakt wel consistent zijn met de constraints die

op de relaties aanwezig zijn. Er zou zelfs een tool uitgebouwd kunnen worden die, afhankelijk van de constraints, de mogelijkheden zou aangeven voor het invoeren, wijzigen en verwijderen van informatie.

# Hoofdstuk 5:

# Conclusie

In deze thesis werd WSDM besproken, een methode om web sites te modelleren, waarbij de aandacht vooral gaat naar de toekomstige gebruikers van de web sites. Ook een aantal kleine tekortkomingen van WSDM werden daarbij besproken.

Eén van deze tekortkomingen, het modelleren van dynamische web sites, kwam daarna in meer detail aan bod. Zo zagen we dat deze dynamische web sites steeds belangrijker worden binnen het hedendaagse internet en dat het opstellen van modellen daarom onmisbaar zijn geworden.
In deze thesis hebben we dan ook gezocht naar een oplossing om modellen op te stellen waarbij het voor gebruikers van een web site mogelijk zou moeten zijn om informatie aan de web site toe te voegen, te wijzigen en te verwijderen.

Niet onbelangrijk hierbij was dat er rekening moest gehouden worden met het autorisatieconcept, waarbij er moet kunnen aangegeven worden welke gebruikers al dan niet bepaalde informatie kunnen manipuleren.

In de eerste plaats hebben we daarna een bespreking gegeven van de belangrijkste methoden op gebied van web site design. Speciale aandacht ging daarbij uit naar hun eventuele kijk op de problematiek die we in deze thesis wilden oplossen.

Het laatste onderdeel van de thesis stelt dan een aantal mogelijkheden voor die tot een oplossing zouden kunnen leiden. Daarbij werd aangetoond of werden voldoende voor- en nadelen gegeven waarom bepaalde oplossingen al dan niet binnen WSDM verwerkt konden worden.
De mogelijke oplossingen zijn:

1. **Methode van RMM+:** het leek interessant om een reeds bestaande oplossing, namelijk die aangeboden in RMM+, over te nemen. Al vlug werd echter ingezien dat deze oplossing niet echt een goede keuze was, omdat ze te veel wijzigingen binnen WSDM met zich zou meebrengen. Ook de onvolledigheid ervan, ten opzichte van onze normen, was een reden om naar een andere oplossing te zoeken.

2. **Een tekstuele oplossing dankzij RIDL:** doordat RIDL een query-taal is, die volledig gebaseerd is op conceptuele schema's, kan men binnen RIDL queries opstellen die afleidbaar zijn uit een conceptueel schema. De taal is volledig uitgewerkt om resultaten weer te geven, dus was er extra syntax nodig om informatie toe te voegen, te wijzigen en te verwijderen.
   Deze extra syntax werd uitgewerkt voor het toevoegen van informatie. Het resultaat was dat alle mogelijkheden aanwezig waren, zodat een goede oplossing in de maak leek.
   Er werd echter ingezien dat het beter zou zijn om een grafische oplossing te zoeken, vermits de communicatie tussen ontwerpers daardoor veel beter verloopt.

3. **Uitbreiding van de ORM notatie:** door de oplossing met RIDL werd er ingezien dat het mogelijk moest zijn om dankzij ORM tot een geschikte oplossing te komen. Ook hier was het onoverkomelijk dat er extra notatie moest toegevoegd worden. Deze notatie zou dan moeten aangeven hoe en welke informatie in bepaalde situaties zou moeten toegevoegd, gewijzigd of verwijderd worden. Daarbij is het dan een voordeel dat de bestaande notatie van ORM reeds aanleiding geeft tot enkele beperkingen.
   Door dezelfde methode toe te passen die in het "conceptual design" van WSDM gebruikt werd (waarbij modellen opgesteld worden voor de verschillende gebruikersklassen), is het autorisatieprobleem ook opgelost. De mogelijkheden die de gebruikers op de web site krijgen kunnen immers op dezelfde wijze gemodelleerd worden.

Dankzij deze oplossing kan WSDM nu ook gebruikt worden om een design op te stellen van dynamische web sites, een niet onbelangrijk deel naar de toekomst van het internet toe. Deze vervollediging van de methode is een onmisbare aanvulling indien er later beslist wordt om een tool te bouwen voor WSDM.

# Bibliografie


[BN 96] M. Bichler, S. Nusser, *W3DT – The Structural Way of Developing WWW-sites,* Proceedings of ECIS'96, 1996.

[DeT 98] O. De Troyer, *Designing Well-Structured Websites: Lessons to be Learned from Database Schema Methodology,* in Proceedings of the ER'98 Conference. Nota's in Computer Science (LNCS), Springer-Verslag, 1998, http://infolab.kub.nl/prj/past/wsdm/

[DeT] O. De Troyer, *Audience-driven Web Design*, In: Information modelling in the new millennium, IDEA Group, in voorbereiding.

[DL 98] O. De Troyer, C. Leune*, WSDM: A User Centered Design Method for Web Sites*, Computer Networks and ISDN systems, Proceedings of the 7th International World Wide Web Conference, eds. H. Ashman, P. Thistlewaite, Elsevier (pp. 85 - 94), 1998.

[DMP 83] O. De Troyer, R. Meersman, F. Ponsaert, *RIDL User Guide,* Research Report, Katholieke Universiteit Brabant, 1983.

[GMD 98] W. Goedefroy, R. Meersman, O. De Troyer, *UR-WSDM: Adding User Requirement Granularity to Model Web Based Information Systems,* Proceedings of 1st Workshop on Hypermedia Development, Pittsburgh, USA, June 20-24, 1998, http://infolab.kub.nl/prj/past/wsdm/

[GPS 93] F. Garzotto, P. Paolini, D. Schwabe, *HDM – A Model-Based Approach to Hypertext Application Design*, ACM Transactions on Information Systems, Vol 11, No 1, pp. 1-26, 1993.

[GRS 97] A. Garrido, G. Rossi, D. Schwabe, *Pattern Systems for Hypermedia,* Proceedings of PloP97 of Program, http://st-www.cs.uiuc.edu/users/hanmer/PloP-97.html, 1997.

[Hal 99] T. Halpin*, Conceptual Schema and Relational Database Design*, WytLytPub USA, second edition, 1999.

[Ibi] http://ibis.nott.ac.uk/guidelines/ch2/chap2-A-11.html

[Inf] *Infomodeler, nu onderdeel van Visio Enterprise Modeler*, Visio Corporation, http://www.visio.com.

[Int] http://www.intersites.com/

[ISB 95] T. Isakowitz, E. A. Stohr, P. Balasubramanian*, RMM: A Methodoloy for Structured Hypermedia Design*, Communications of the ACM, Vol. 38, No 8, pp. 34-43, 1995.

[Ise] http://ise.ee.uts.edu.au/ise/homepages/hmpgs97a/kendall/1_essay.htm

[Net] http://www.netcraft.com/Survey/
http://www.netcraft.com/Survey/Reports/200007/graphs.html

[Orm] http://www.orm.net

[Php] http://www.php.net
http://www.php.net/usage.php

[Pii] http://www.pi.informatik.tu-darmstadt.de/~neuss/www1/w4-main.html

[SRB 96] D. Schwabe, G. Rossi, S. Barbosa*, Systematic Hypermedia Application Design with OOHDM*, http://www.cs.unc.edu/barman/HT96/P52/section1.html.

[RSL 98] G. Rossi, D. Schwabe, F. Lyardet, *Improving Web Information Systems with Navigational Patterns,* http://www8.org/w8-papers/5b-hypertext-media/improving/improving.html, 1998.

[Vis] *Visio 2000*, Visio Corporation, http://www.visio.com.

[Zol] P. Zoller, *RMM+: Modeling Interactive and Adaptive Hypermedia Applications*, In: Information modelling in the new millennium, IDEA Group, in voorbereiding.