ADIRA : adaptive information retrieval application

Beens, R.; Hoogendoorn, A.F.

Award date:
2001

TECHNISCHE UNIVERSITEIT EINDHOVEN

Faculteit Wiskunde en Informatica

AFSTUDEERVERSLAG

ADIRA:
Adaptive Information Retrieval Application

door

Richard Beens en Teun Hoogendoorn

Afstudeercommissie
dr. ir. G.J.P.M. Houben
prof. dr. P.M.E. De Bra
mr. E.P.G. Kamps

Juni 2001

Voorwoord

Dit verslag is een rapportage van de afstudeeropdracht die is uitgevoerd bij Kamps/Goyaerts/Putters (KGP), een maatschap van advocaten en belastingadviseurs gevestigd in 's-Hertogenbosch. De afstudeeropdracht vormt voor de auteurs de laatste fase van de studie Technische Informatica aan de Technische Universiteit Eindhoven (TUE).

Dit verslag is interessant voor lezers die geïnteresseerd zijn in adaptieve hypermedia systemen. Lezers die bekend zijn met deze systemen worden verwezen naar hoofdstuk 3, 4 en 5. Lezers die vooral geïnteresseerd zijn in het resultaat worden verwezen naar hoofdstuk 7, hoofdstuk 9, bijlage B en bijlage C.

Onze dank gaat uit naar mr. E.P.G. Kamps van KGP en dr.ir. G.J. Houben van de TUE voor hun begeleiding. Prof. dr. P.M.E. de Bra willen we bedanken voor zijn deelname aan de examencommissie en de adviezen die hij ons heeft gegeven. Binnen KGP gaat onze dank uit naar Bert de Laat voor de technische ondersteuning met betrekking tot het systeembeheer. De overige medewerkers van Kamps/Goyearts/Putters bedanken we voor hun medewerking en het aangename verblijf tijdens de afstudeeropdracht. Als laatste gaat onze dank uit naar onze ouders, partners, familie en vrienden die ons gedurende de hele studie hebben gesteund.

Eindhoven, juni 2001

Richard Beens
Teun Hoogendoorn

Samenvatting

Door de auteurs van dit verslag is voor Kamps/Goyaerts/Putters (KGP) een afstudeeropdracht uitgevoerd. Deze opdracht is als volgt geformuleerd: "Maak een technisch ontwerp van een systeem dat een input- en outputproces realiseert. Het inputproces dient ervoor te zorgen dat informatie van verschillende auteurs wordt verzameld. Het outputproces dient ervoor te zorgen dat deze informatie op behoefte wordt aangeboden aan gebruikers. Implementeer vervolgens het systeem aan de hand van het ontwerp."

Uit analyse is gebleken dat het inputproces relatief eenvoudig te realiseren is door een Database Management System (DBMS) met een replicatiemechanisme. De focus van dit verslag ligt op het outputproces dat gerealiseerd is d.m.v. een adaptief hypermedia systeem, genaamd Adaptive Information Retrieval Application (ADIRA). Er is voor het Adaptive Hypermedia Application Model (AHAM) gekozen als referentiemodel voor de architectuur van ADIRA. De belangrijkste onderdelen van AHAM zijn het Domain Model, User Model en het Adaptation Model. Het Domain Model is een abstracte representatie van de informatie. Het User Model houdt een toestand van de gebruiker in relatie tot het Domain Model bij. In het Adaptation Model zijn regels gedefinieerd die het adaptatiegedrag specificeren.

Er is een architectuur gepresenteerd, waarvan de belangrijkste component de AdaptiveEngine is. Deze zorgt ervoor dat de opgevraagde pagina wordt geadapteerd aan de behoefte van de gebruiker en vervolgens wordt gepresenteerd. De belangrijkste techniek die gebruikt wordt voor de adaptatie is het aanbieden van meest relevante links naar informatie waar de gebruiker behoefte aan heeft.

Het ontwerp, dat een gedetailleerde uitwerking is van de architectuur, is volledig geïmplementeerd en ingevoerd op het intranet van KGP.

Het applicatiedomein van ADIRA is geïdentificeerd als informatie retrieval systemen. Deze identificatie hangt samen met de regels die gedefinieerd zijn in het AM. Door een aantal kleine wijzigingen kan ADIRA tevens dienst doen als educatief of on-line help systeem.

Toekomstig werk zal bestaan uit het ontwikkelen van een tool, die de auteur met weinig informatica-ervaring ondersteunt bij het onderhouden van ADIRA. Tevens zal het toekomstig werk bestaan uit het vergroten van het applicatiedomein.

Inhoudsopgave

1 Inleiding

Opdrachtgever Kamps/Goyaerts/Putters (KGP) is een maatschap van advocaten en belastingadviseurs met het ondernemingsrecht als specialisme. Met name het midden- en kleinbedrijf en de particuliere sector doen een beroep op de dienstverlening van het kantoor, waarbij het accent op de advisering ligt.

1.1 Probleem

Specifiek
KGP wil haar cliënten voorzien van een totaal advisering. Het specialisme van KGP beperkt zich echter tot advocatuur en belastingadvies. KGP wil haar kennis combineren met de kennis van andere specialisten om een totaal advisering te bereiken. Een onderdeel hiervan is een website waar de cliënten de informatie van de verschillende specialisten kunnen vinden. De informatie voor deze website wordt op één punt, genaamd Exceldus [K00], verzameld.
Door het verzamelen van de informatie van de specialisten zal er een grote bron van informatie ontstaan. Om uit deze bron de juiste informatie aan de cliënt te bieden, is het noodzakelijk te weten welke informatie hij relevant vindt. De opdrachtgever wil dat de cliënt zijn behoefte opgeeft om, aan de hand van deze behoefte, relevante informatie aan te bieden.

Generiek
Het zojuist geschetste probleem is als volgt te generaliseren. Er is een inputproces nodig dat informatie van verschillende auteurs verzamelt. Een auteur is de generalisatie van de specialist in het specifieke probleem. Er is een outputproces nodig dat deze informatie aanbiedt aan gebruikers. De aangeboden informatie dient te worden afgestemd op de behoefte van de individuele gebruiker. Een gebruiker is de generalisatie van de cliënt in het specifieke geval.

De opdracht is als volgt geformuleerd.
" Maak een technisch ontwerp van een systeem dat een input- en outputproces realiseert. Het inputproces dient ervoor te zorgen dat informatie van verschillende auteurs wordt verzameld. Het outputproces dient ervoor te zorgen dat deze informatie op behoefte wordt aangeboden aan gebruikers. Implementeer vervolgens het systeem aan de hand van het ontwerp."

1.2 Aanpak

Om de opdracht te realiseren zijn drie fasen onderscheiden:
1. Analyseren van het probleem.
2. Ontwerpen van een systeemarchitectuur op basis van de analyse.
3. Implementeren van het ontwerp.

1.3 Inhoud verslag

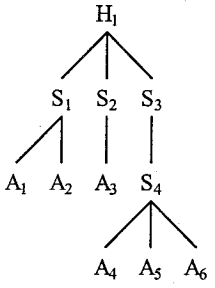
Het verslag volgt de aanpak beschreven in paragraaf 1.2. De eerste fase wordt besproken in de hoofdstukken 2 en 3. De tweede fase wordt besproken in de hoofdstukken 4 en 5. De derde fase wordt besproken in de hoofdstukken 6 en 7. Hoofdstuk 8 behandelt het applicatiedomein. In hoofdstuk 9 wordt de conclusie gepresenteerd.

2 Probleemanalyse

De analyse van het probleem (zie paragraaf 1.1) wordt in dit hoofdstuk beschreven. In paragraaf 2.1 wordt de structuur van de informatie besproken. In paragraaf 2.2 wordt het input- en outputproces geanalyseerd. Paragraaf 2.3 bespreekt twee typen hypermedia systemen die van toepassing kunnen zijn. Paragraaf 2.4 presenteert de keuzes die gemaakt zijn op basis van de analyses.

2.1 Informatie

De opdrachtgever beschikt over een database, waarin de informatie op een hiërarchische manier is gestructureerd. Een hoofdonderwerp wordt de wortel van de hiërarchie genoemd en een artikel een blad. Tussen een hoofdonderwerp en een artikel kunnen zich een variabel aantal knopen bevinden die subonderwerpen worden genoemd. Een artikel representeert de daadwerkelijke informatie, terwijl de onderwerpen alleen meta-informatie bevatten om het artikel te kunnen rubriceren. Figuur 1 geeft weer hoe de structuur gezien kan worden.



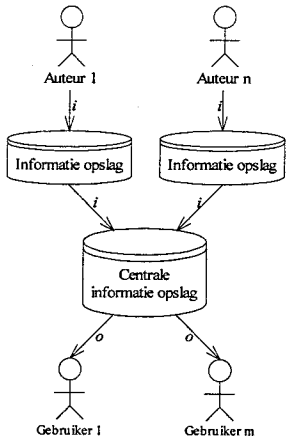
H_i = Hoofdonderwerp i
S_i = Subonderwerp i
A_i = Artikel i

Figuur 1: Informatiestructuur

Een artikel bestaat uit een aantal fragmenten zoals titel, auteur en inhoud. Een fragment kan bestaan uit tekst en links, die respectievelijk dienen voor de inhoud van en de navigatie tussen artikelen.

2.2 Processen

Er zijn zoals besproken in paragraaf 1.1 twee processen. Het inputproces wordt behandeld in paragraaf 2.2.1. Het outputproces wordt behandeld in paragraaf 2.2.2.



i = Inputproces
o = Outputproces

Figuur 2: Processen

2.2.1 Inputproces

Het inputproces zorgt ervoor dat de informatie die de verschillende auteurs aanbieden op één punt wordt verzameld. Er dient te worden voorkomen dat er redundantie ontstaat in de verzamelde informatie. Een organisatorische eis is nodig om dit te bewerkstelligen. In het specifieke probleem van KGP kan ervoor gekozen worden om aan iedere auteur een verzameling hoofdonderwerpen toe te kennen. Deze verzamelingen dienen disjunct te zijn. Aan de opdrachtgever c.q. auteur KGP kunnen bijvoorbeeld de hoofdonderwerpen advocatuur en belastingadvies worden toegekend.
Het verzamelen van informatie is eenvoudig te realiseren door gebruik te maken van een Database Management System (DBMS) met een replicatiemechanisme. Replicatie zorgt ervoor dat bij elkaar behorende gegevens, die op verschillende locaties zijn opgeslagen en bewerkt worden, consistent blijven. Er dient rekening gehouden te worden met de hoeveelheid verzamelde informatie, die groot kan worden en sterk kan variëren, omdat de auteurs voortdurend informatie kunnen toevoegen.

2.2.2 Outputproces

Het outputproces zorgt ervoor dat de verzamelde informatie aan de gebruiker wordt aangeboden. Dit is geen triviaal proces, omdat de aangeboden informatie geadapteerd dient te worden aan de behoefte van de gebruiker.

Definitie: Behoefte is een toestand die de wensen van een gebruiker aan informatie op een bepaald moment representeert.

Om relevante informatie aan de gebruiker aan te bieden is het van belang dat de behoefte van de gebruiker bekend is. Een gedeelte van de behoefte dient afgeleid te worden, zodat de gebruiker niet overstelpt wordt met vragen naar zijn behoefte.

2.3 Hypermedia systemen

Vanwege de ontsluiting van informatie via een website is er sprake van een web gebaseerd hypermedia systeem.

De Bra [D98] zegt over hypermedia het volgende:

> *"Hypermedia is een acroniem dat de woorden hypertext en multimedia combineert. De woorden hypermedia en hypertext worden vaak gebruikt als synoniemen. Hoewel hypertext suggereert dat alle informatie in de vorm van onopgemaakte tekst is, staan de meeste hypertext systemen het gebruik van informatie in andere vormen als graphics, geluid, animatie en/of video toe."*

Shneiderman [SK89] definieert hypertext als:

> *"Een database met actieve kruisverwijzingen, die de lezer in staat stelt naar wens te 'jumpen' naar andere delen van de database."*

2.3.1 Traditioneel hypermedia systeem

Een traditioneel hypermedia systeem bestaat uit een verzameling pagina's die onderling verbonden zijn d.m.v. links. Er zijn een aantal problemen die zich voor kunnen doen bij een traditioneel hypermedia systeem:

1. In het algemeen wordt een pagina voor iedere gebruiker op dezelfde wijze gepresenteerd. Het aanpassen van de presentatie van informatie voor een willekeurige gebruiker is moeilijk.
2. De navigatievrijheid is vaak te ruim, omdat een pagina voor iedere gebruiker op dezelfde manier gepresenteerd wordt. De gebruiker kan geen goede keuze voor een link maken, omdat er simpelweg teveel links gepresenteerd worden.

Een referentiemodel is een abstract conceptueel model van de architectuur van een hypermedia systeem. In de praktijk wordt vaak het Dexter-model [HS90] als referentiemodel voor een traditioneel hypermedia systeem gebruikt.

2.3.2 Adaptief hypermedia systeem

Een adaptief hypermedia systeem richt zich op het voorkomen van de problemen die kunnen ontstaan door het gebruik van een traditioneel hypermedia systeem. Brusilovsky [B96] definieert een adaptief hypermedia systeem (AHS) als volgt.

> *"Met een adaptief hypermedia systeem bedoelen we een hypermedia systeem dat een aantal kenmerken van de gebruiker reflecteert in een gebruikersmodel en dit model toepast om verschillende visuele aspecten van het systeem te adapteren aan de gebruiker. Anders gezegd, het systeem dient aan drie criteria te voldoen: het moet een hypertext of hypermedia systeem zijn; het moet een gebruikersmodel hebben; het moet in staat zijn de hypermedia te adapteren aan de hand van dit model."*

De problemen die zich voor kunnen doen in een traditioneel hypermedia systeem, zoals besproken in paragraaf 2.3.1, worden door een AHS opgelost door respectievelijk content- en linkadaptatie.
Met contentadaptatie wordt het presenteren van de informatie afhankelijk van de kennis van de gebruiker, doelen, voorkeuren of andere karakteristieken bedoeld. Een voorbeeld van contentadaptatie is het toevoegen van een reclamebanner van KGP indien de op te vragen informatie betrekking heeft op de advocatuur.
De gedachte achter linkadaptatie is het adapteren van de linkstructuur, zodat de gebruiker geleid wordt naar interessante en relevante informatie en weg wordt gehouden van onrelevante informatie.

In de literatuur wordt het Adaptive Hypermedia Application Model (AHAM) [WHD98, DHW99] voorgesteld als referentiemodel voor adaptieve hypermedia systemen. Een toepassing van AHAM is de Adaptive Hypermedia Architecture (AHA) [WHD98]. AHA wordt op de Technische Universiteit Eindhoven gebruikt voor twee educatieve systemen en een kiosk systeem.

2.4 Resultaat van de probleemanalyse

Voor de realisatie van het systeem is er een input- en outputproces nodig, zoals is beschreven in paragraaf 2.2. Het inputproces is redelijk eenvoudig te realiseren door een DBMS te kiezen met een replicatiemechanisme. Door de in paragraaf 2.2.1 besproken organisatorische eis toe te passen behoeft er geen rekening gehouden te worden met redundantie in de verzamelde informatie. De implementatie van het inputproces valt buiten de scope van dit verslag.
Het outputproces is te realiseren d.m.v. een hypermedia systeem. Een traditioneel hypermedia systeem voldoet niet, omdat de presentatie van de informatie niet kan worden afgestemd op karakteristieken van de individuele gebruiker. De oplossing dient gezocht te worden in een adaptief hypermedia systeem, omdat de presentatie afgestemd moet worden op de behoefte van de individuele gebruiker. Het adaptief hypermedia systeem dat in dit verslag gepresenteerd wordt, is de Adaptive Information Retrieval Application (ADIRA).
Voor het ontwerp van de architectuur van ADIRA is gekozen om gebruik te maken van AHAM. AHAM biedt voldoende mogelijkheden en vrijheden om het outputproces te realiseren. De mogelijkheid om de AHA-software te gebruiken is overwogen, maar verworpen om de volgende redenen:

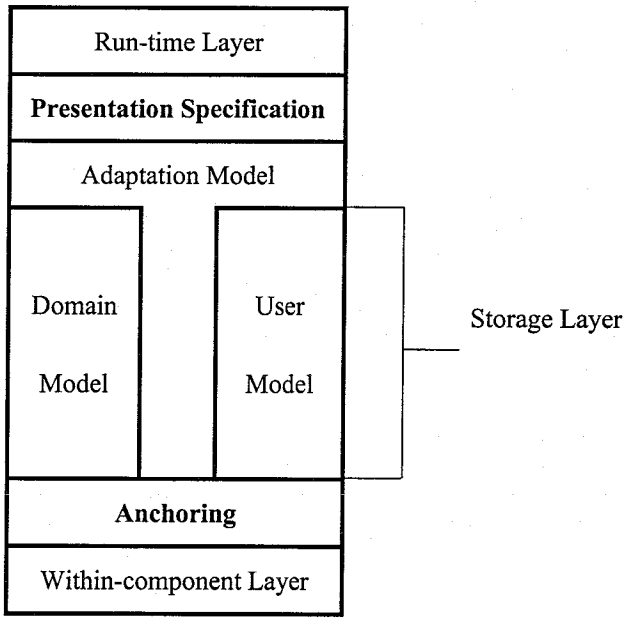
1. AHA wijkt op een aantal punten sterk af van het model [WHD98].
2. AHA werkt met een statische verzameling informatie. ADIRA dient rekening te houden met een dynamische verzameling informatie, omdat de auteurs voortdurend informatie aanleveren.
3. AHA werkt zonder een DBMS.

3 Referentiemodel (AHAM)

AHAM is gebaseerd op het Dexter Model van Halasz en Schwartz [HS90]. Voor een uitgebreide beschrijving van AHAM wordt verwezen naar [DHW99]. AHAM bestaat uit een architectuur van drie lagen:

1. Runtime Layer
   Deze laag verzorgt de presentatie aan de gebruiker.
2. Storage Layer
   Deze laag representeert de netwerkstructuur van knopen en links die de essentie van de hypertext is. De laag bestaat uit het Domain Model (DM), het User Model (UM) en het Adaptation Model (AM). De focus van AHAM ligt op deze laag, omdat de andere lagen van de implementatie afhankelijk zijn.
3. Within-component Layer
   Deze laag representeert de inhoud en structuur van elke knoop.

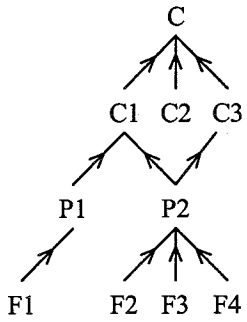
De AdaptiveEngine is de software omgeving die ervoor zorgt dat content en links worden geconstrueerd en geadapteerd. De AdaptiveEngine voert de adaptatie uit door het genereren van wat AHAM een Presentation Specification noemt. De Presentation Specification bevat informatie die specificeert hoe links en content worden gepresenteerd. De Runtime Layer maakt gebruik van de Presentation Specification om een pagina te presenteren. Anchoring representeert een mechanisme voor het adresseren van (of het refereren aan) locaties of items binnen de inhoud van een pagina.



Figuur 3: AHAM

3.1 Domain Model

Het DM bestaat uit concepten en conceptrelaties. Een concept is een abstracte representatie van een informatie item. Een concept kan atomair of samengesteld zijn. Een atomair concept correspondeert met een informatiefragment. De attributen van dit concept behoren tot de "Within-component Layer". Een samengesteld concept heeft een verzameling kinderen, die alle atomair of samengesteld zijn. Een samengesteld concept met alleen atomaire kinderen wordt een pagina genoemd.
Een conceptrelatie relateert twee of meer concepten aan elkaar. Er zijn verschillende soorten conceptrelaties, waarvan een link, een partof en een prerequisite voorbeelden zijn. Een partof relatie geeft aan dat een concept C1 deel is van een ander concept C. Door middel van deze relatie worden de kinderen van een concept gedefinieerd. Een prerequisite relatie geeft aan dat een pagina P1 gelezen dient te worden vóór pagina P2. De atomaire concepten, de samengestelde concepten en de conceptrelaties vormen samen het domain model.



C_i = Concept i (samengesteld abstract concept i)
P_i = Pagina i (samengesteld pagina concept i)
F_i = Fragment i (atomair concept i)

Figuur 4: Concepthiërarchie

De concepthiërarchie is een gerichte acyclische graaf, waarbij elk atomair concept bevat moet zijn in een samengesteld concept. In figuur 4 wordt een voorbeeld gegeven van een concepthiërarchie.

3.2 User Model

Het UM associeert voor iedere gebruiker een aantal attributen met concepten in het DM. Voor iedere gebruiker houdt het UM een tabelachtige structuur bij, waarin voor elk concept de attribuutwaarden voor dat concept zijn opgeslagen. Een voorbeeld van een attribuutwaarde is de kenniswaarde. De kenniswaarde representeert de kennis die de gebruiker van een concept heeft.

3.3 Adaptation Model

In het AM wordt d.m.v. adaptatieregels gedefinieerd hoe het DM en UM gecombineerd moeten worden om de adaptatie uit te voeren. In de regels staat beschreven hoe de kennis van de gebruiker de presentatie van informatie beïnvloedt. Een regel bestaat uit de volgende onderdelen:

1. Conditiegedeelte
   Het conditiegedeelte geeft aan of de regel uitgevoerd moet worden. Het conditiegedeelte wordt opgesplitst in een event, een fase, een set van concepten en een boolean expressie. Een event wordt bijvoorbeeld gegenereerd op het moment dat een gebruiker een pagina opvraagt (access event). Door gebruik te maken van de fase is het mogelijk adaptatie uit te voeren die gebaseerd is op de toestand voordat (pre-fase) en nadat (post-fase) de pagina is gelezen. Een fase die tussen de twee genoemde fasen gebruikt kan worden is de gen-fase die de generatie van de pagina voorstelt. De set van concepten geeft de concepten aan waarop de regel van toepassing is. Indien de regel op alle concepten van toepassing is spreekt men van een generieke regel en anders van een specifieke regel. De boolean expressie refereert aan attribuutwaarden uit het DM of UM.
2. Actiegedeelte
   Het actiegedeelte betreft een assignment aan een attribuutwaarde in het UM of een assignment aan een Presentation Specification. Een actie wordt alleen uitgevoerd als aan het conditiegedeelte is voldaan.
3. Boolean veld propagate
   Het boolean veld propagate geeft aan of de regel andere regels mag aanroepen (triggeren), waardoor een sequentie van uit te voeren regels kan ontstaan.

3.4 Wijzigingen aan het model

Er zijn een aantal aanpassingen aan de definitie van de adaptatieregels van AHAM gemaakt om een duidelijker beeld te krijgen welke regels worden uitgevoerd.

Triggeren van regels
In AHAM kan een adaptatieregel andere regels triggeren d.m.v. het boolean veld propagate. De variabelen in het actiegedeelte van de regel bepalen welke regels getriggerd dienen te worden. Als in het actiegedeelte de variabele X gewijzigd wordt en propagate true is, worden alle regels met variabele X in het conditiegedeelte

getriggerd. De volgende nadelen zijn door de auteurs geïdentificeerd bij hantering van deze wijze van triggeren:

1. Er is altijd een toekenning aan een variabele noodzakelijk om een trigger te veroorzaken.
2. Het overzicht van elkaar triggerende regels verdwijnt bij een redelijk aantal regels. Actiegedeelten en conditiegedeelten moeten op overeenkomstige variabelen vergeleken worden.
3. Het is niet mogelijk waarden door te geven bij aanroep van een regel.
4. Er wordt gebruikt gemaakt van conceptrelaties in het conditiegedeelte. De consequenties, die een actie heeft voor concepten waarmee bepaalde conceptrelaties bestaan, worden pas duidelijk bij bestudering van de overige regels.

De eerste drie problemen worden opgelost door de introductie van het veld eventcall dat in de plaats van het veld propagate komt. Het veld eventcall is een tuple <N, C, V> waar N staat voor de naam van het event, C staat voor het concept waarop de event van toepassing is en V een waarde is die meegegeven kan worden. Door het event bij zijn naam aan te roepen is duidelijk welke regels in aanmerking komen voor uitvoer. Er is dus geen toekenning aan een variabele meer nodig om een getriggerde regel te identificeren. Het veld eventcall vormt dus de eerste stap in het selectieproces van de uit te voeren regels. De waarde V is optioneel en daardoor alleen van toepassing in bijzondere gevallen.
Het vierde probleem wordt opgelost door het aanroepen van een event op (alle) concepten waarmee een bepaalde conceptrelatie bestaat. Een voorbeeld hiervan is het aanroepen van het event updateRead op alle ouders p van concept C.

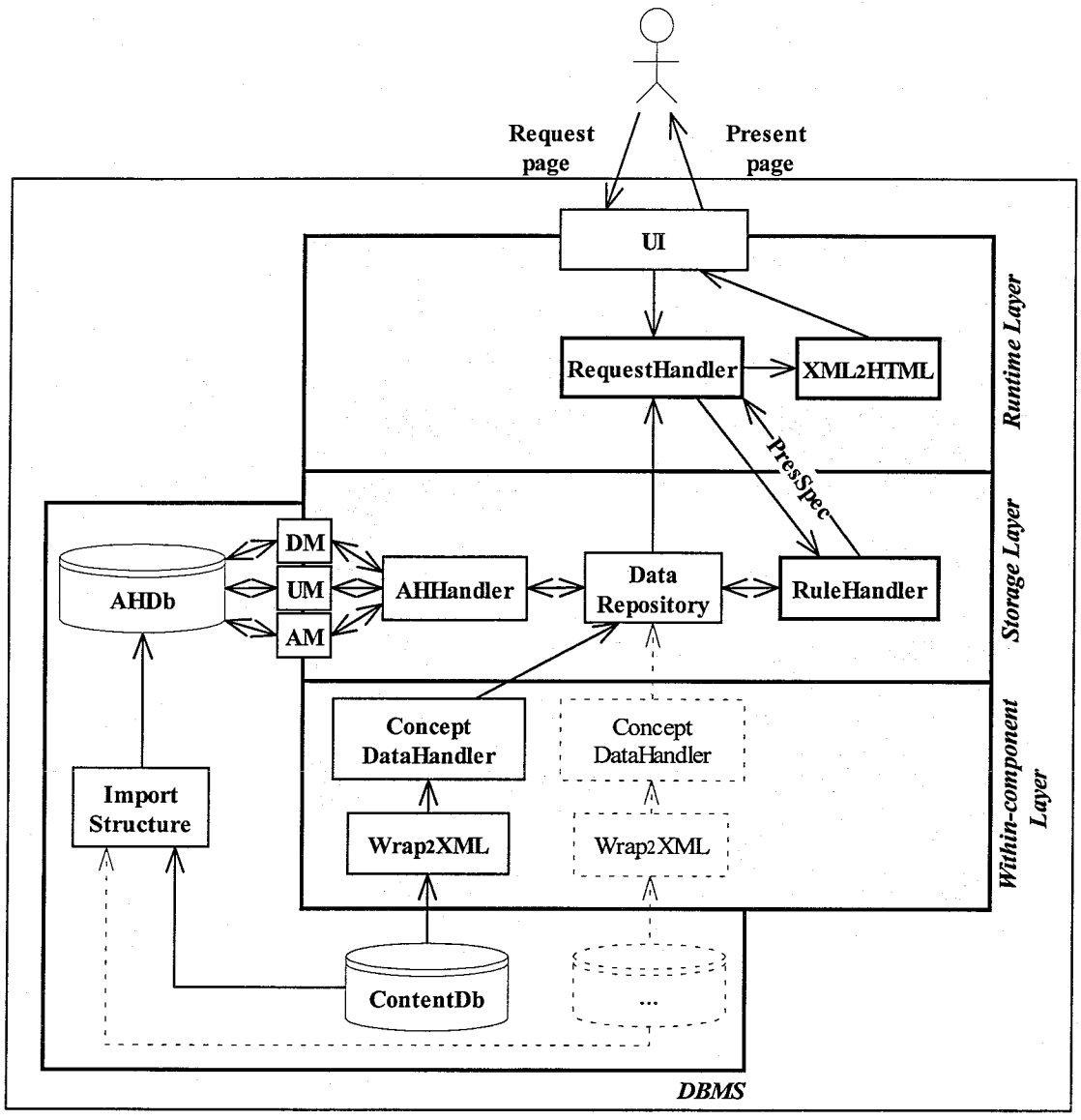
```
forall p in C.parents do updateRead(p) od
```

Interne en externe events

Door de introductie van het veld eventcall en het daarmee aanroepen van een event bij zijn naam ontstaan in het AM regels die alleen door triggering van een andere regel het gewenste effect hebben. Om te voorkomen dat ongewenste regels getriggerd kunnen worden, is er een onderscheid gemaakt tussen interne en externe events. Externe events treden op als een concept opgevraagd wordt door de gebruiker. Interne events kunnen alleen door andere regels gegenereerd worden, door het gebruik van het veld eventcall in de regel. Dit onderscheid in events wordt gemaakt in een extra veld, genaamd subphase. Dit veld kan de waarde internal of external aannemen.

4 Architectuur

Dit hoofdstuk presenteert de architectuur van ADIRA. In de architectuur zijn de lagen die worden voorgesteld door AHAM terug te vinden. De AdaptiveEngine genereert een Presentation Specification, bouwt een pagina op gebruikmakend van deze specificatie en retourneert een pagina in HTML (HyperText Markup Language) formaat. In de architectuur van ADIRA wordt deze functionaliteit gescheiden in respectievelijk de componenten RuleHandler, RequestHandler en XML2HTML. De componenten RequestHandler en XML2HTML zijn in de Runtime Layer geplaatst, omdat deze componenten de presentatie verzorgen van de opgevraagde pagina. De RuleHandler is in de Storage Layer geplaatst, omdat deze component de Presentation Specification genereert die volgens AHAM de interface is tussen de Runtime Layer en Storage Layer.



Figuur 5: Architectuur

4.1 Database Management System

Het DBMS verzorgt de fysieke opslag van gegevens. Gegevens die nodig zijn voor het DM, UM en AM worden opgeslagen in de AHDb. De inhoud van de artikelen wordt opgeslagen in één of meerdere ContentDb's. De component Import Structure is geïntroduceerd om automatisch één abstracte representatie van de informatie uit de verschillende ContentDb's te genereren. De auteur heeft de mogelijkheid deze representatie aan te passen.

4.2 Within-component Layer

De Within-component Layer zorgt ervoor dat de fragmenten, die door de Storage Layer opgevraagd zijn, worden opgehaald uit een ContentDb. De DataRepository roept de juiste ConceptDataHandler aan, die het fragment ophaalt uit de bijbehorende ContentDb.
Om ervoor te zorgen dat ADIRA niet afhankelijk is van de implementatie van een ContentDb is het noodzakelijk dat fragmenten in hetzelfde dataformaat worden aangeboden aan de Storage Layer. De component Wrap2XML transformeert data uit een ContentDb naar XML formaat.

XML
De eXtensible Markup Language (XML) is een generieke syntax voor het beschrijven van hiërarchische data, maar beschrijft niet hoe de data zijn gerepresenteerd. De metataal XML is bedoeld om een uniforme structuur aan te brengen aan de inhoud van één of meerdere documenten. De splitsing van definitie (het meta-deel) en inhoud wordt bereikt door het gebruik van een Document Type Definition (DTD) waarin op meta-niveau de inhoud van een document en zijn eigenschappen beschreven staan.

Er is voor XML als dataformaat gekozen om de volgende redenen:
1. XML is een standaard formaat voor het uitwisselen van data.
2. XML scheidt opmaak van data.

4.3 Storage Layer

De DataRepository verschaft toegang tot alle gegevens die opgevraagd kunnen worden uit een database. Via de AHHandler heeft de DataRepository toegang tot DM, UM en AM. Door een ConceptDataHandler aan te roepen krijgt de DataRepository toegang tot de bijbehorende ContentDb.

4.3.1 User Model

Het UM houdt permanent van iedere gebruiker een gebruikersprofiel bij, waarin staat hoeveel behoefte hij aan een concept heeft. Voor de vastlegging van de behoefte houdt het gebruikersprofiel een aantal waarden bij. Deze waarden worden gebruikt voor het aanbieden van de meest relevante pagina's en het kleuren van links, zie paragraaf 5.4.3.

Initiële behoefte
De BI-waarde (behoefte initieel) is geïntroduceerd, die per concept de initiële behoefte van een gebruiker representeert. Voor de BI-waarde zijn er de volgende mogelijkheden:
1. Een percentage dat aangeeft in welke mate de gebruiker behoefte heeft aan een concept. Het is echter moeilijk voor de gebruiker om exact het percentage aan te geven.
2. Een boolean waarde die aangeeft of de gebruiker wel of geen behoefte aan een concept heeft.

Er is gekozen voor een boolean waarde, zodat de gebruiker alleen de keuze heeft tussen wel of geen behoefte aan een concept. De concepten, waarvoor geldt dat er initieel behoefte aan is, beschrijven een gerichte graaf. Als de gebruiker heeft aangegeven dat hij initieel behoefte heeft aan een concept, heeft hij initieel behoefte aan alle ouders van het concept. Er is voor gekozen dat de gebruiker de initiële behoefte op twee manieren aan kan geven:
1. Hij kan zijn behoefte op een top-down manier aangeven d.m.v. het gebruikersprofiel. Als een gebruiker zijn gebruikersprofiel opvraagt, krijgt hij de keuze uit de kinderen van de root van de conceptthiërarchie.
2. Hij kan zijn behoefte op een bottum-up manier aangeven door bij een pagina aan te geven dat hij behoefte heeft aan pagina's van hetzelfde onderwerp.

Voldane behoefte
Voor het afleiden van de behoefte is de BVp-waarde (behoefte voldaan percentage) geïntroduceerd, die per concept de procentuele mate voor de behoeftevoldoening van de gebruiker representeert. De BVp-waarde wordt verhoogd naar mate er meer verschillende pagina's worden gelezen die deel uitmaken van het betreffende samengesteld abstract concept. Voor de berekening van de BVp-waarde zijn de BVa-waarde (behoefte voldaan absoluut) en de MB-waarde (maximale waarde voor BVa) geïntroduceerd. De BVa-waarde is een absolute maat voor de behoeftevoldoening van een concept. De MB-waarde stelt de maximale waarde voor die de BVa-waarde aan kan nemen.

De BVp-waarde van C wordt als volgt berekend.

```
C.BVp = (C.BVa / C.MB) * 100
```

Behoeftetoestand
De verzameling van alle BI- en BVp-waarden die betrekking hebben op een gebruiker vormt de behoeftetoestand van die gebruiker. De behoeftetoestand dient bijgewerkt te worden nadat een gebruiker een nog niet gelezen pagina heeft opgevraagd. Voor het bijwerken van de BVp-waarden dienen de BVa-waarden berekend te worden.
De toename van de BVa-waarde van een pagina concept P is als volgt gedefinieerd.

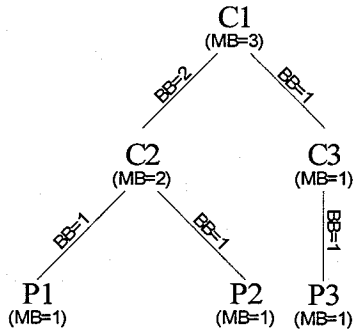
```
ΔP.BVa = 1
```

De toename van de BVa-waarde van een abstract concept C1 op een ouder C2 is als volgt gedefinieerd.

```
ΔC2.BVa = (ΔC1.BVa / C1.MB) * BB(C1, C2)
```

De BB-waarde stelt de behoeftebijdrage van een concept C1 op een concept C2 voor. Een hogere BB-waarde drukt uit dat een concept interessanter is voor de gebruiker en daarom meer bijdraagt in de behoeftevoldoening. In het AM zijn regels geformuleerd, die ervoor zorgen dat bij elke request van een concept de behoeftetoestand wordt bijgewerkt.

Voorbeeld



Figuur 6: Voorbeeld concepthiërarchie voor bijwerken behoeftetoestand

Nadat pagina P2 is gelezen, wordt de toename van de BVa-waarden als volgt berekend.
1. ΔP2.BVa = 1
2. ΔC2.BVa = (ΔP2.BVa / P2.MB) * BB(P2, C2) = (1/1) * 1 = 1
3. ΔC1.BVa = (ΔC1.BVa / C2.MB) * BB(C2, C1) = (1/2) * 2 = 1

Stel dat de BVa-waarden initieel 0 zijn, dan worden de BVa-waarden na uitvoering van de bovenstaande berekening gelijk aan de berekende toename. De BVp-waarden worden als volgt berekend.
1. P2.BVp = (P2.BVa / P2.MB) * 100 = (1/1) * 100 = 100
2. C2.BVp = (C2.BVa / C2.MB) * 100 = (1/2) * 100 = 50
3. C1.BVp = (C1.BVa / C1.MB) * 100 = (1/3) * 100 = 33

4.3.2 Domain Model

Om vast te leggen in welke mate een concept C1 in de behoeftevoldoening bijdraagt aan een concept C2 is het attribuut behoeftebijdrage (BB) geïntroduceerd (zie paragraaf 4.3.1). Tevens is de MB-waarde geïntroduceerd als grens voor de BVa-waarde (zie paragraaf 4.3.1). De BB- en MB-waarden worden in het DM gerepresenteerd.
Standaard komt de MB-waarde van een concept C overeen met het aantal pagina's dat deel uitmaakt van C. De BB-waarde van concept C1 op concept C2 komt standaard overeen met de MB-waarde van C1. Deze standaard waarden bewerkstelligen dat de BVp-waarde het percentage aangeeft van het aantal pagina's, deel uitmakende van een concept C, dat gelezen is.

4.3.3 Adaptation Model

Voor de specificatie van de taal die een adaptatieregel specificeert, wordt de EBNF (Extended Backus Naur Form) notatie [II96] gehanteerd.

EBNF
EBNF wordt gebruikt om de syntax van een context vrije grammatica te beschrijven in een aantal produktieregels. EBNF beschrijft de syntax door gebruik te maken van terminal en non-terminal symbolen. Terminals (één of meerdere karakters) worden omsloten door aanhalingstekens. Non-terminals worden omsloten door de metasymbolen "<" en ">". Een productieregel bestaat uit een linkerzijde en een rechterzijde gescheiden door het metasymbool "::=". De linkerzijde is een non-terminal en wordt gedefinieerd door de rechterzijde. De rechterzijde is een sequentie van terminals en non-terminals. Voor iedere non-terminal die voorkomt in de rechterzijde bestaat een produktieregel met die non-terminal aan de linkerzijde.

Specificatie
Een adaptatieregel wordt opgesplitst in drie non-terminals die de conditie, actie en eventueel de eventcall specificeren. De conditie wordt verder opgesplitst in non-terminals die het event, de fase, de subfase, de set van concepten en de boolean expressie specificeren. De fase wordt opgesplitst in terminals die de pre-, gen- en post-fase specificeren.

```
<RULE> ::= <CONDITION><ACTION>(<EVENTCALL>)?
<CONDITION> ::= <EVENT><PHASE><SUBPHASE><SOC><BOOLEAN EXPRESSION>
<PHASE> ::= 'pre' | 'gen' | 'post'
```

Voor de definitie van de overige non-terminals en terminals wordt verwezen naar bijlage A.

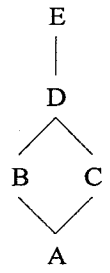
4.3.4 RuleHandler

De RuleHandler voert de regels die in het AM gedefinieerd zijn uit en retourneert een Presentation Specification aan de RequestHandler in de Runtime Layer. Bij de uitvoering van de regels in het AM dient rekening gehouden te worden met de problemen omtrent terminatie en determinisme [WDAH00]. Voor deze problemen zijn echter (nog) geen universele oplossingen beschikbaar. In paragraaf 4.3.4.1 en 4.3.4.2 worden respectievelijk de oplossingen gepresenteerd die ADIRA hanteert om terminatie en determinisme te garanderen.

4.3.4.1 Terminatie

Door het triggeren van regels kan er een situatie ontstaan waarin de uitvoering van de regels niet eindigt. Het is namelijk mogelijk dat er een cycle ontstaat tijdens het triggeren van de regels. Aangezien dit een onwenselijke situatie is dient er speciale aandacht naar dit probleem uit te gaan. Er zijn verschillende mogelijkheden om terminatie te garanderen [WDAH00]:

1. Het systeem mag een attribuutwaarde van een concept ten hoogste één keer veranderen per uitvoering van de regels. In sommige gevallen is het echter wenselijk om één attribuutwaarde meerdere malen te veranderen. Als concept A een behoefteverandering bewerkstelligt voor concepten B en C, en B en C beiden een (mogelijk andere) behoefteverandering bewerkstelligen bij concept D, dan wordt deze verandering maar één keer doorgevoerd (zie figuur 7).
2. Iedere instantie van een regel wordt ten hoogste één keer uitgevoerd. Een regel kan meerdere malen uitgevoerd worden, maar heeft dan betrekking op andere concepten. Het probleem van de vorige mogelijkheid is hiermee opgelost. Maar als D in het vorige voorbeeld een behoefteverandering bewerkstelligt bij concept E, dan wordt deze behoefteverandering maar één keer doorgevoerd i.p.v. twee keer (zie figuur 7).
3. Het alleen toestaan van monotone updates. Indien er alleen een verhoging (of verlaging) van een attribuutwaarde wordt toegestaan en er vastgestelde grenzen bestaan voor de attribuutwaarde, dan termineert de uitvoering van de regels bij het bereiken van een grens.



Figuur 7: Voorbeeld concepthiërarchie

De eerste en derde mogelijkheid vallen af, omdat door de introductie van de eventcall in paragraaf 3.4 de toekenning van een waarde aan een variabele geen trigger meer is voor een regel. De tweede mogelijkheid heeft betrekking op instanties van regels en kan wel gebruikt worden om terminatie af te dwingen. In ADIRA is voor deze mogelijkheid gekozen om terminatie te garanderen.

4.3.4.2 Determinisme

De volgorde waarin de regels worden uitgevoerd kan het uiteindelijke resultaat beïnvloeden, dit wordt non-deterministisch gedrag genoemd. Non-determinisme is niet wenselijk en dient opgelost te worden zodat het resultaat deterministisch bepaald wordt.

Ambiguïteit tussen regels
Regels die aan dezelfde attribuutwaarde een toekenning doen mogen geen ambigue conditiegedeelte hebben. Een voorbeeld van deze vorm van ambiguïteit is het volgende.

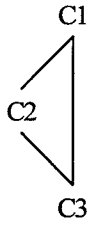
```
access(C) → C.X = 1
access(C) → C.X = 0
```

Bovenstaand voorbeeld geeft een situatie weer waarbij de volgorde van uitvoering het resultaat van de attribuutwaarde C.X bepaalt. Het invoeren van soortgelijke regels in het AM wordt als foutief auteurswerk beschouwd.

Uitvoering van de regels
Er is naast de ambiguïteit nog een probleem dat het uiteindelijke resultaat kan beïnvloeden. Dit probleem is eenvoudig te demonstreren aan de hand van het volgende voorbeeld:

```
access(C)                  → C.X = C.X + 1
access(C) ∧ C.X == 0       → C.Y = 1
```

Als initieel geldt dat de waarde van C.X gelijk is aan 0, dienen de bovenstaande twee regels beiden uitgevoerd te worden. Echter bij uitvoering van de eerste regel verandert de waarde van C.X (deze wordt namelijk 1) en de tweede regel wordt niet meer uitgevoerd. Dit is een onwenselijke situatie die de auteur niet valt aan te rekenen.
De oplossing die in ADIRA wordt toegepast ter voorkoming van dit probleem is het identificeren welke regels getriggerd worden alvorens ze uit te voeren. In bovenstaand voorbeeld worden beide regels door ADIRA geïdentificeerd als uitvoerbaar en vervolgens uitgevoerd. Deze oplossing is echter niet geheel toereikend, hetgeen duidelijk wordt gemaakt aan de hand van het volgende voorbeeld:



Figuur 8: Potentieel determinisme probleem

Figuur 8 geeft een situatie weer van een DM waarin een concept meerdere ouders kan hebben. Als een regel een eventcall op alle ouders van C3 uitvoert, worden de regels voor concepten C1 en C2 geïdentificeerd en uitgevoerd. Wanneer de geïdentificeerde regels voor C1 en C2 een eventcall op alle ouders van C1 en C2 uitvoeren worden opnieuw regels voor C1 geïdentificeerd. Het voor de tweede keer identificeren van regels voor hetzelfde concept is een potentieel probleem. De actie die wordt uitgevoerd na de eerste identificatie van C1 kan de tweede identificatie uitsluiten. Dit probleem kan vermeden worden door het aantal ouders van een concept te beperken tot ten hoogste één.

4.4 Runtime Layer

De RequestHandler vormt de centrale component van de Runtime Layer. De component stelt een XML document samen aan de hand van een Presentation Specification die geretourneerd is door de RuleHandler. Het XML document wordt naar HTML getransformeerd d.m.v. de component XML2HTML en aan de gebruiker gepresenteerd.

4.4.1 RequestHandler

De RequestHandler handelt een request van de gebruiker af door bij de RuleHandler regels te triggeren. De triggering is verdeeld in drie fasen.

Pre-fase
Een request van de gebruiker resulteert in de triggering van de regels die gedefinieerd zijn voor de pre-fase. In de pre-fase kunnen attribuutwaarden in het UM veranderd worden, maar is altijd één of meerdere assignments aan een Presentation Specification noodzakelijk.
Een aanroep van de gebruiker bestaat uit een event en een concept waarop het event van toepassing is. Onderstaand voorbeeld geeft een request (een access event) van concept P1 weer.

```
access(P1)
```

Gen-fase
Na de pre-fase worden regels in de gen-fase getriggerd, die betrekking hebben op het presenteren van de pagina. In de gen-fase zijn alleen assignments aan een Presentation Specification toegestaan. Nadat alle regels in de gen-fase getriggerd zijn, gebruikt de RequestHandler de Presentation Specification om de informatie samen te stellen. Het samenstellen resulteert in een XML document, dat na transformatie naar HTML, aan de gebruiker gepresenteerd kan worden.

Post-fase
Nadat de request is afgehandeld en het resultaat aan de gebruiker is gepresenteerd worden de regels getriggerd die gedefinieerd zijn voor de post-fase. In de post-fase kunnen alleen assignments aan attribuutwaarden in het UM gedaan worden.

4.4.2 XML2HTML

De component XML2HTML transformeert het XML document dat de AdaptiveEngine samengesteld heeft naar een HTML document. Een vorm van transformatie is noodzakelijk omdat een XML document geen beschrijving geeft om de data te presenteren. Er is voor XSL als opmaaktaal gekozen om een XML document op te maken.

XSL
De eXtensible Stylesheet Language (XSL) is een taal voor het uitdrukken van stylesheets, waarmee de presentatie van data beschreven wordt. De taal bestaat uit twee onderdelen:
1. XSL Transformations (XSLT), een taal voor het transformeren van XML documenten.
2. Een XML vocabulaire voor het specificeren van een opmaak semantiek.

Een XSL stylesheet specificeert de presentatie van een klasse XML documenten door te beschrijven hoe een instantie van die klasse wordt getransformeerd in een XML document dat het voorgeschreven vocabulaire gebruikt.

5 Ontwerp

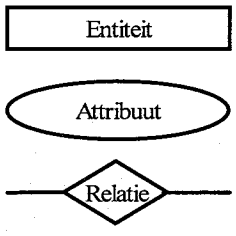
Dit hoofdstuk behandelt het ontwerp van ADIRA, dat een gedetailleerde uitwerking van de architectuur is. De modelleertechnieken die gebruikt zijn, worden in paragraaf 5.1 gepresenteerd. Paragraaf 5.2 geeft het ontwerp van het DBMS aan de hand van E-R diagrammen weer. Paragraaf 5.3, 5.4 en 5.5 behandelen respectievelijk de Within-component Layer, Storage Layer en Runtime Layer aan de hand van klassendiagrammen.

5.1 Modelleertechnieken

Voor het ontwerp zijn twee modelleertechnieken gebruikt die hieronder beschreven staan.

E-R diagram
De gebruikte datamodelleertaal is het E-R diagram [O94]. Een E-R diagram is een datamodel dat data classificeert in de volgende onderdelen:

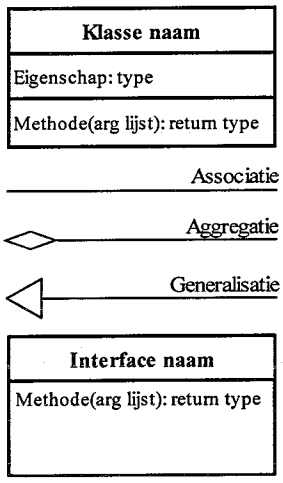
1. Entiteiten
   Een entiteit is een object met algemene eigenschappen.
2. Relaties
   Een relatie kan bestaan tussen twee of meerdere entiteiten.
3. Attributen
   Een attribuut is een data item dat een eigenschap van een entiteit of relatie beschrijft.



Figuur 9: Onderdelen van een E-R diagram

Klassendiagram
Een klassendiagram [EP98] geeft een statische weergave op het systeem in termen van klassen en relaties tussen de klassen. Een klassendiagram bestaat uit de volgende onderdelen:

1. Klassen
   Een klasse, bestaande uit methoden en eigenschappen, kan direct geïmplementeerd worden in een object georiënteerde programmeertaal.
2. Relaties
   Een relatie kan bestaan tussen twee of meerdere klassen. De volgende relaties zijn mogelijk:
   - Associatie
     Een associatie tussen klassen A en B representeert dat klasse A gebruik kan maken van de methoden en eigenschappen van klasse B.
   - Aggregatie
     Een aggregatie tussen klassen A en B representeert dat klasse B deel uitmaakt van klasse A.
   - Generalisatie
     Een generalisatie tussen klassen A en B representeert dat klasse B een specifiek geval is van klasse A.
3. Interfaces
   Een interface beschrijft een aantal gedragskenmerken die een klasse kan ondersteunen door de interface te implementeren.



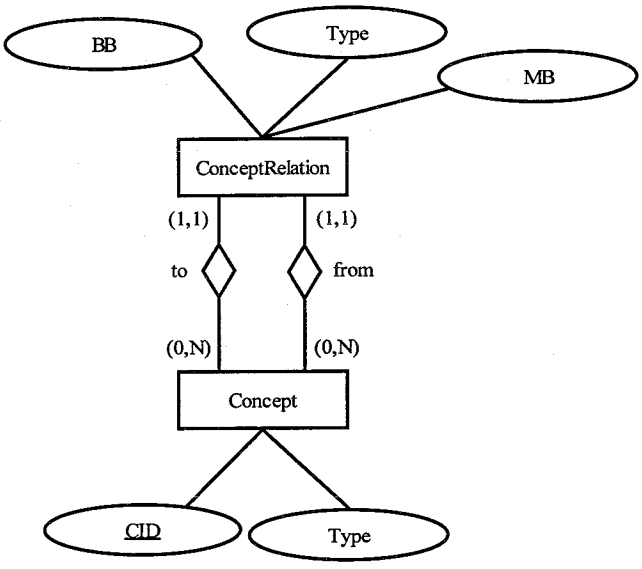
Figuur 10: Onderdelen van een klassendiagram

5.2 Database Management System

Het DM, UM en AM worden respectievelijk in de paragrafen 5.2.1, 5.2.2 en 5.2.3 besproken. In paragraaf 5.2.4 wordt de interface anchoring besproken. Paragraaf 5.2.5 geeft het volledige E-R diagram weer.

5.2.1 Domain Model

De entiteit Concept is de centrale entiteit in de vastlegging van het DM. Het Concept bestaat uit een unieke key (CID) en een type. Een Concept kan van het type "abstract", "page" of "fragment" zijn. De concepthiërarchie wordt vastgelegd door de entiteit ConceptRelation van het type "partof". Een ConceptRelation kan tevens van het type "link" zijn. Constraints 1 t/m 3 (zie figuur 11) geven respectievelijk aan dat een fragment een pagina als ouder heeft, een pagina een abstract concept en een abstract concept een abstract concept. Constraint 4 geeft informeel aan dat de concepthiërarchie geen cycles mag bevatten.



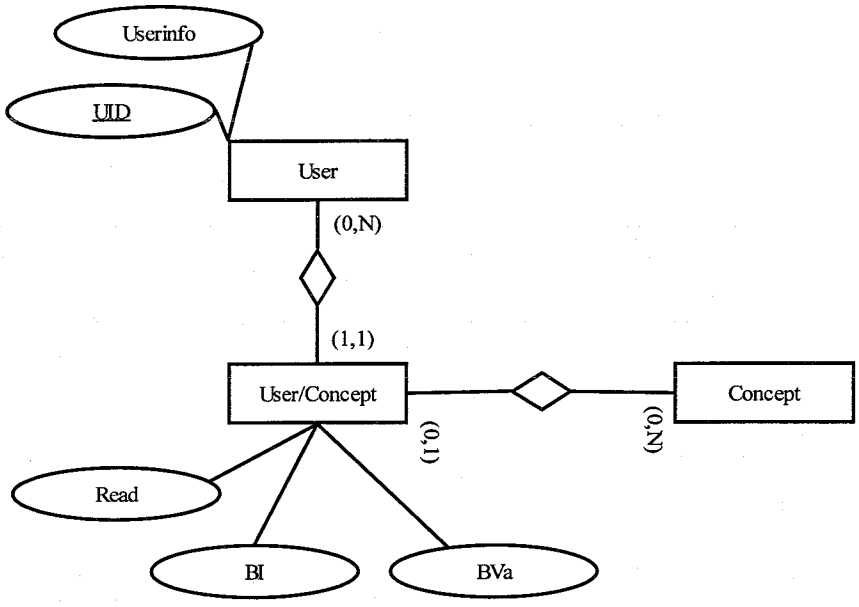
Constraints:

1. (∀ cr ∈ Concept Relation: cr.type = "partof" ∧ from(cr).Type = "fragment": to(cr).Type = "page")
2. (∀ cr ∈ Concept Relation: cr.type = "partof" ∧ from(cr).Type = "page": to(cr).Type = "abstract")
3. (∀ cr ∈ Concept Relation: cr.type = "partof" ∧ from(cr).Type = "abstract": to(cr).Type = "abstract")
4. "De concepthiërarchie mag geen cycles bevatten"

Figuur 11: DM

5.2.2 User Model

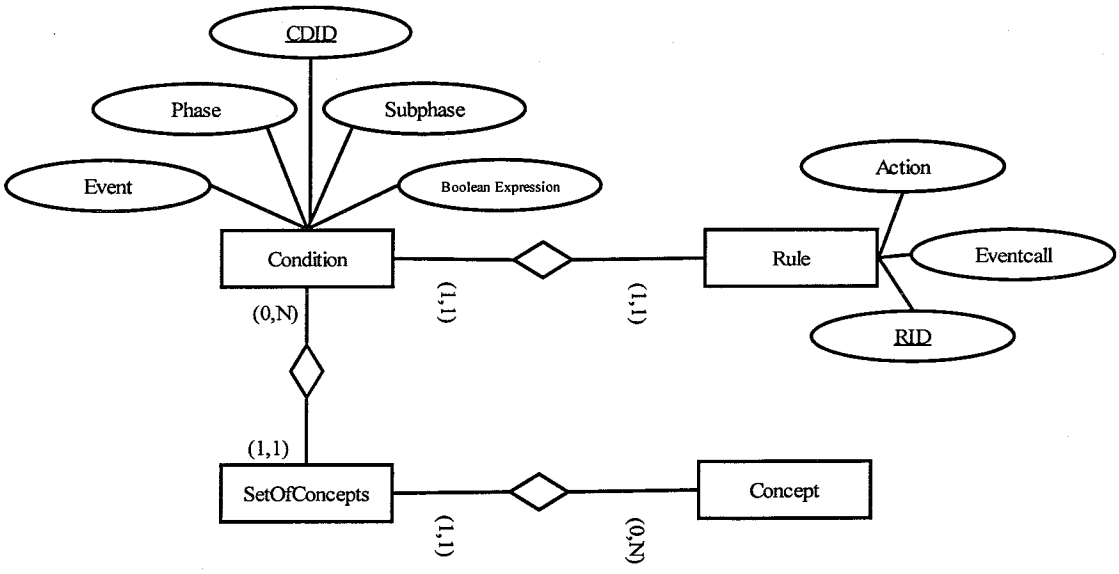
De entiteit User is in de vastlegging van het UM de centrale entiteit. Een gebruiker is uniek te identificeren aan de hand van een unieke key (UID). Persoonsgegevens als naam en e-mail kunnen worden opgeslagen, hetgeen gerepresenteerd wordt door het attribuut Userinfo.
De entiteit User/Concept legt het verband tussen de User en Concept vast. Voor ieder concept, dat wordt opgevraagd, wordt het aantal requests (Read), de initiële behoefte (BI) en de BVa-waarde (BVa) vastgelegd.



Figuur 12: UM

5.2.3 Adaptation Model

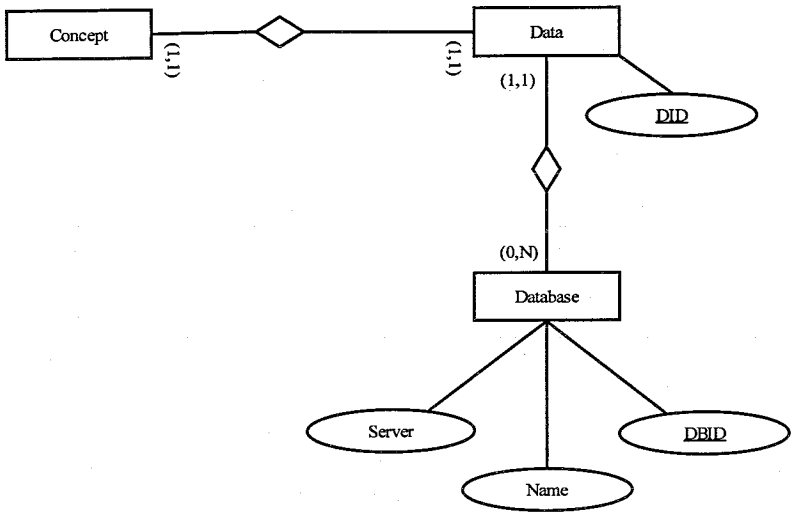
De entiteit Rule is de centrale entiteit in de vastlegging van het AM. Een regel bestaat uit een conditie, een actie en een eventcall. De actie en de eventcall worden gerepresenteerd door respectievelijk de attributen Action en Eventcall. Voor de conditie is een entiteit Condition geïntroduceerd. De onderdelen event, fase, subfase en boolean expressie van een conditie worden als attributen gerepresenteerd. Voor de set van concepten is een entiteit SetOfConcepts geïntroduceerd. Deze entiteit heeft relaties met de entiteiten Condition en Concept.



Figuur 13: AM

5.2.4 Anchoring

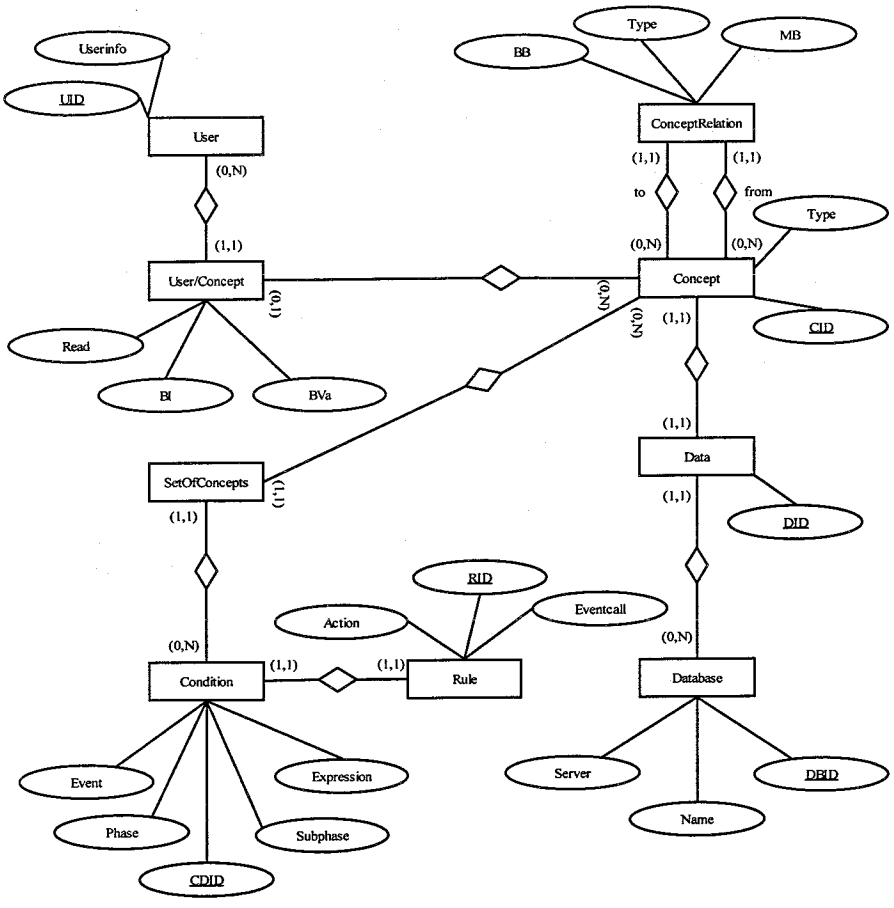
De inhoud van een concept wordt gerepresenteerd door de entiteit Data. Data heeft een relatie met Database en Concept. Van een Database worden de attributen Name, Server en een unieke key (DBID) vastgelegd.



Figuur 14: E-R diagram anchoring

5.2.5 E-R diagram

Figuur 15 geeft het complete E-R diagram weer, dat tot stand komt door samenvoeging van de diagrammen die gepresenteerd zijn in de voorgaande paragrafen.

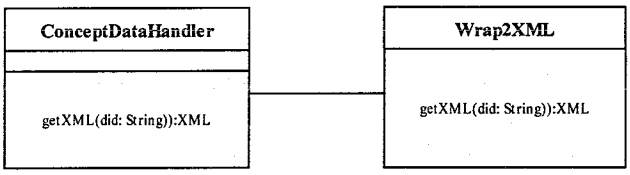


Figuur 15: E-R diagram

5.3 Within-component Layer

De klasse ConceptDataHandler implementeert de interface Wrap2XML. Wrap2XML maakt het mogelijk data uit verschillende ContentDb's in hetzelfde XML formaat te transformeren.

Objectmodel
De klasse ConceptDataHandler wordt aangeroepen d.m.v. de methode getXML. Deze methode retourneert de opgevraagde data in XML formaat.



Figuur 16: De klasse ConceptDataHandler

ContentDb
Iedere ContentDb heeft zijn eigen wrapper. De wrapper moet bij het transformeren van data items uit de database naar XML voldoen aan de volgende DTD. Deze DTD is generiek gehouden om de transformatie zo eenvoudig mogelijk te maken.

```
<!ELEMENT PAGE (FRAGMENT+)>
<!ELEMENT FRAGMENT ID (#PCDATA)>
```

Een voorbeeld van een XML document dat gecreëerd kan worden door de component Wrap2XML wordt hieronder weergegeven. Merk op dat er geen tag title wordt gecreëerd, maar een fragment tag met de ID title.

```
<PAGE>
  <FRAGMENT ID=title>
   levensverzekering
  </FRAGMENT>
  <FRAGMENT ID=author>
   mr. E.P.G. Kamps
  </FRAGMENT>
  <FRAGMENT ID=abstract>
   Overeenkomsten van levensverzekering vallen ook onder de ter
   beschikking gestelde vermogensbestanddelen...
  </FRAGMENT>
  .
</PAGE>
```

5.4 Storage Layer

Paragraaf 5.4.1 bespreekt de DataRepository. In paragraaf 5.4.2 worden de identificatie en de uitvoering van de regels besproken. Paragraaf 5.4.3 bespreekt de Presentation Specification.

5.4.1 DataRepository

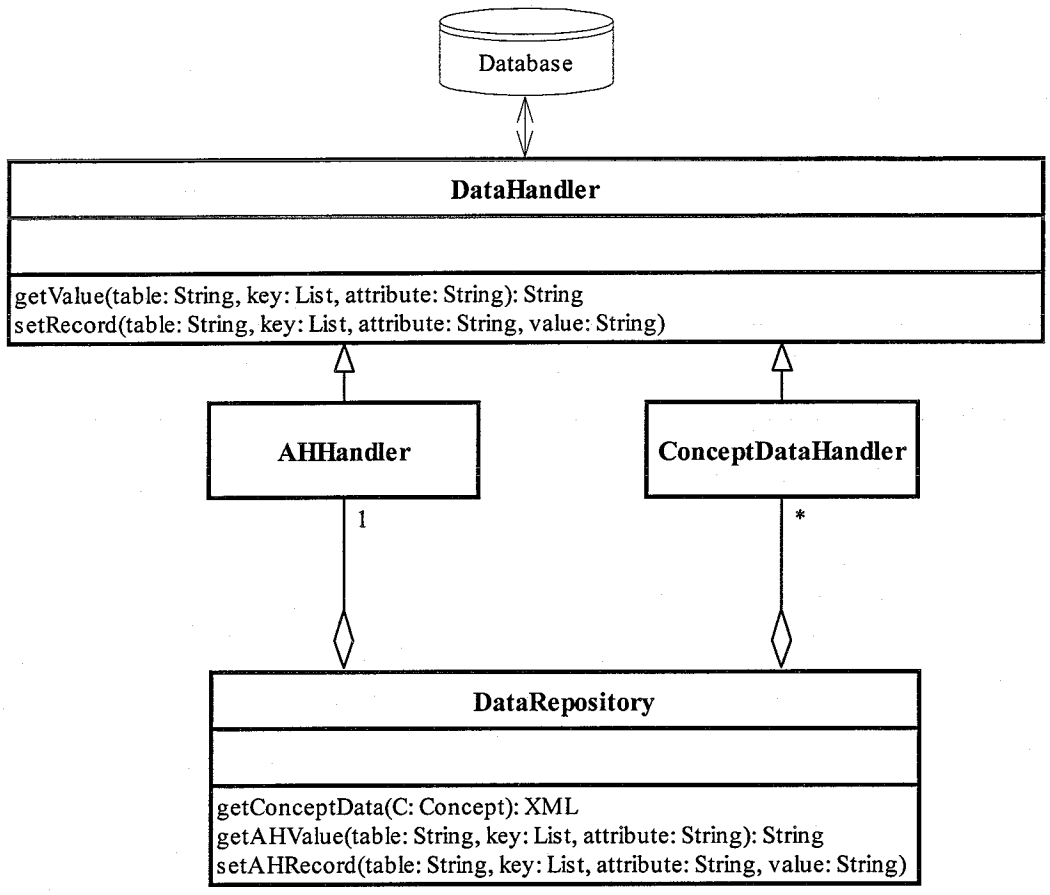
Alle data die worden opgevraagd, worden via de klasse DataRepository opgehaald. De klasse maakt gebruik van de klasse AHHandler en ConceptDataHandler. De klasse AHHandler geeft toegang tot de objecten die door het E-R diagram van paragraaf 5.2.5 zijn gemodelleerd. Een ConceptDataHandler geeft toegang tot een ContentDb. Zowel de AHHandler als de ConceptDataHandler erven over van de DataHandler. De DataHandler is de enige component die afhankelijk is van de implementatie.

Objectmodel
De klasse DataRepository kan worden aangeroepen door de volgende methoden.

1. getConceptData
   Deze methode retourneert de content van een concept in XML formaat. De methode doet een request aan

de ConceptDataHandler, die de content uit de betreffende database ophaalt.

2. getAHValue en setAHRecord
   Deze methoden maken het mogelijk om data respectievelijk op te vragen en op te slaan in de AHDatabase.



Figuur 17: De klasse DataRepository

5.4.2 RuleHandler

De RuleHandler vormt de centrale component van de Storage Layer. Hij identificeert getriggerde regels en voert deze regels uit.

Identificatie van regels
Bij aanroep van event E wordt een regel R uit het AM geïdentificeerd als getriggerde regel indien:

1. De naam van E overeenkomt met de naam van R.
2. De fase en subfase van triggering overeenkomen met R.
3. Het concept waarop de triggering van toepassing is in de lijst van concepten van R voorkomt.
4. De boolean expressie van R geldt.
5. Een regel S de regel R niet eerder heeft aangeroepen met hetzelfde event en concept. Dit ter voorkoming van het mogelijk niet termineren van de uitvoering, zie paragraaf 4.3.4.1.

De geïdentificeerde regels worden in een wachtrij geplaatst en na identificatie van alle getriggerde regels wordt met de uitvoering van de regels gestart.

Uitvoeren van regels
De acties van de regels in de wachtrij worden uitgevoerd, waarbij een assignment aan een attribuutwaarde in het UM of een assignment aan een Presentation Specification wordt gedaan. Als een regel een eventcall heeft gedefinieerd worden de regels geïdentificeerd behorende bij deze eventcall en achteraan in de wachtrij geplaatst. Op deze manier worden acties, behorende bij de regels die al in de wachtrij stonden, voor de acties

behorende bij de nieuwe geïdentificeerde regels uitgevoerd. Door het gebruik van de wachtrij wordt determinisme van het resultaat gegarandeerd zoals besproken in paragraaf 4.3.4.2.
Bij het identificeren en uitvoeren van de regels dienen de attribuutwaarden Boolean Expression, Action en Eventcall geëvalueerd te worden. Het evalueren bestaat uit twee onderdelen die aan de hand van het volgende voorbeeld besproken worden.

```
UC.Read = UC.Read + 1
```

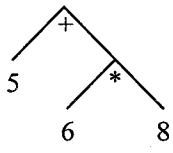
Opvragen en wijzigen van attribuutwaarden
De attribuutwaarde UC.Read dient zowel opgevraagd als gewijzigd te worden. De RuleHandler beschikt over een instantie UC van klasse UserConcept die de huidige entiteit User/Concept voorstelt. De methode getRead van UC vraagt de attribuutwaarde Read op en de methode setRead(x) wijzigt de attribuutwaarde Read in x. De RuleHandler beschikt naast de instantie UC ook over de instanties C, U en P die respectievelijk het huidige concept, de huidige gebruiker en de huidige Presentation Specification voorstellen. Onderstaande tabel geeft deze instanties weer, met daarbij vermeld of de attribuutwaarden gezet en gewijzigd mogen worden.

Instantie	Opvragen van attribuutwaarden	Wijzigen van attribuutwaarden
C	Ja	Nee
U	Ja	Nee
UC	Ja	Ja
P	Ja	Ja

Figuur 18: Instanties en attribuutwaarden

Figuur 18 geeft aan dat instanties van een User en een Concept niet gewijzigd mogen worden. Deze instanties kunnen alleen in de boolean expressie gebruikt worden.

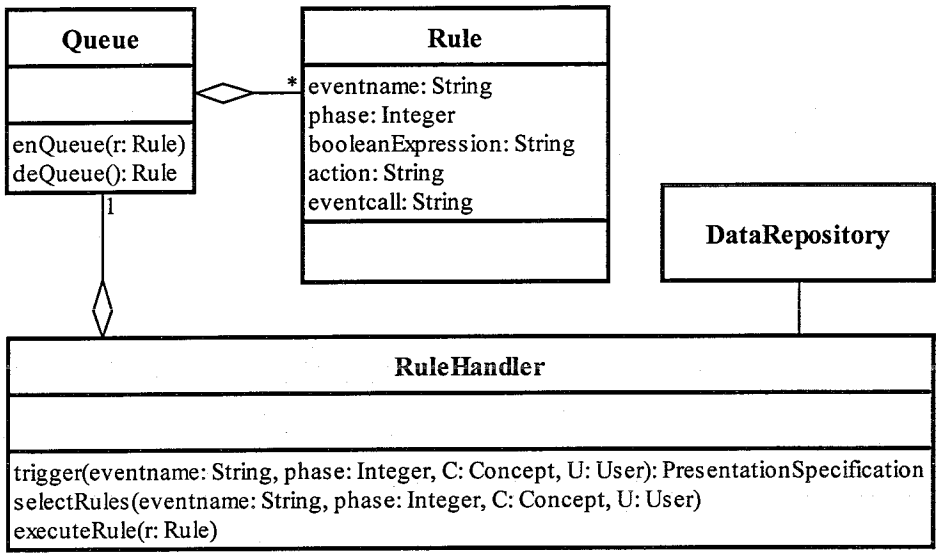
Uitrekenen van expressies
De expressie UC.Read + 1 dient uitgerekend te worden. Een expressie kan bestaan uit (attribuut)waarden en operaties. Een operatie heeft een prioriteit t.o.v. een andere operatie, een voorbeeld hiervan is dat vermenigvuldigen voor optellen uitgevoerd dient te worden. De prioriteiten van de operaties zijn verwerkt in de EBNF specificatie van de taal (zie bijlage A), waardoor de expressieboom rekening houdend met de prioriteiten wordt opgebouwd. Een voorbeeld hiervan is de boom voor de expressie 5 + 6 * 8.



Figuur 19: Expressieboom van 5 + 6 * 8

Als in de expressieboom om iedere knoop haakjes worden gezet en de boom startend in de top, linksom langs de takken, wordt afgelopen ontstaat de expressie (5 + (6 * 8)). De prioriteit van vermenigvuldigen t.o.v. optellen is nu duidelijk zichtbaar.

Objectmodel
De klasse RuleHandler wordt aangeroepen door de methode trigger. Deze methode roept de methode selectRules aan die de regels identificeert en in de Queue (wachtrij) plaatst. Vervolgens worden de regels, die in de Queue staan, uitgevoerd d.m.v. de methode executeRule. Als er een eventcall optreedt, wordt de methode selectRules opnieuw aangeroepen. Dit proces eindigt als de Queue leeg is.



Figuur 20: De klasse RuleHandler

5.4.3 Presentation Specification

Een Presentation Specification bestaat uit een verzameling conceptrelaties. De conceptrelaties worden aan de Presentation Specification toegevoegd door het specificeren van regels in het AM.

Toevoegen van conceptrelaties van type partof
Het toevoegen van de fragmenten van concept C aan Presentation Specification P wordt door de volgende regel gespecificeerd.

```
access(C) → P.addCR("partof", {C})
```

In het geval dat niet alle fragmenten getoond mogen worden, kunnen fragmenten verwijderd worden uit de Presentation Specification. Het verwijderen van de fragmenten title en author wordt door de volgende regel gespecificeerd.

```
access(C) → P.removeCR("partof", {"title", "author"})
```

Hoewel er eerst conceptrelaties toegevoegd dienen te worden om vervolgens conceptrelaties te kunnen verwijderen maakt de volgorde van uitvoering van bovenstaande regels niet uit. De Presentation Specification specificeert namelijk alleen de toevoegingen en verwijderingen van concepten. Dit resulteert in de gewenste situatie, omdat de RequestHandler eerst de toevoegingen en vervolgens de verwijderingen uitvoert.

Toevoegen van conceptrelaties van type link
Als een fragment F wordt toegevoegd aan de Presentation Specification P dan worden tevens alle links behorende bij F toegevoegd aan P. Dit is noodzakelijk omdat er links in de tekst kunnen staan. Als dit niet gebeurt, kunnen gaten in de tekst ontstaan.
Een link die is toegevoegd aan P krijgt een kleur, die de relevantie van de link representeert. De kleur van een link wordt gespecificeerd door regels in het AM. Er kunnen tevens links worden toegevoegd die niet bij een fragment horen. Het belangrijkste voorbeeld hiervan is het toevoegen van de meest relevante links.

MRL-algoritme
Het MRL-algoritme bepaalt de meest relevante links voor de gebruiker. Het volgende wordt uitgevoerd om het resultaat S op te leveren met N relevante links die verwijzen naar pagina's die deel uitmaken van de concepthiërarchie van abstract concept C:

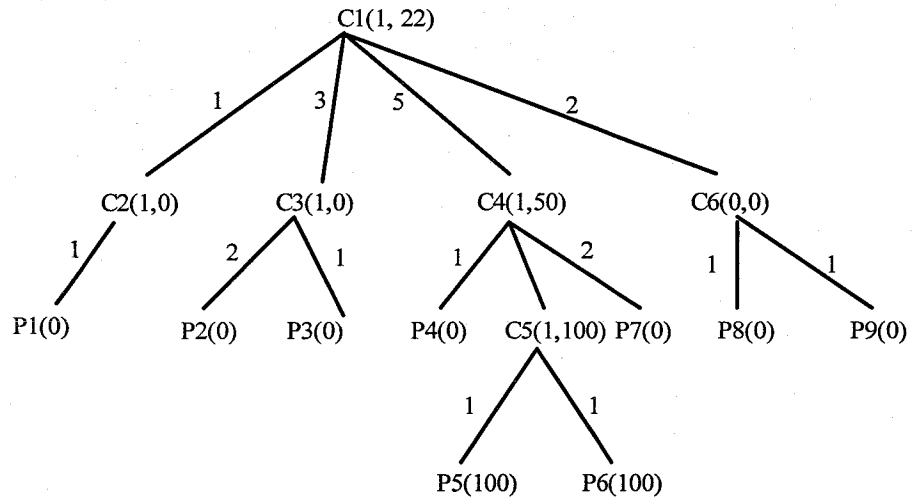
De pagina's waarnaar de relevante links moeten gaan verwijzen worden op een top-down manier opgezocht. Voor concept C wordt bekeken welk kind in aanmerking komt voor toevoeging aan S. Om een rangorde te maken van meest relevant naar minst relevant, worden de kinderen van C aan de hand van de volgende sleutels gesorteerd.

1. Aflopend op de initiële behoefte (BI-waarde)
   Er is hiervoor gekozen, omdat behoefte die is aangegeven door de gebruiker als meest betrouwbaar wordt geacht. Indien een kind een BVp-waarde van 100 heeft, wordt deze achteraan in de sortering geplaatst door de BI-waarde in de sortering te verlagen, zodat er geen pagina's worden aangeboden waar volledig in behoefte is voldaan.
2. Oplopend op de voldane behoefte (BVp-waarde)
   Hoe meer er in behoefte is voldaan hoe minder interessant het voor de gebruiker wordt geacht.
3. Aflopend op de behoefte bijdrage (BB-waarde) die het kind op C heeft
   Een concept dat meer bijdraagt in de behoefte wordt interessanter geacht voor de gebruiker.

Als de kinderen van concept C zijn gesorteerd, worden de kinderen op volgorde van de sortering toegevoegd aan het resultaat S. Indien een kind K een abstract concept is, worden de kinderen van K op dezelfde manier gerangschikt en toegevoegd als beschreven voor de kinderen van C. Dit proces herhaalt zich totdat S tenminste N pagina's bevat.

De gebruiker kan twee typen concepten opvragen:
1. Een abstract concept
   Het MRL-algoritme wordt uitgevoerd op dit concept.
2. Een pagina concept
   Er moet gezocht worden naar een geschikt abstract concept om het MRL-algoritme op uit te voeren. Er is voor gekozen om bottum-up te zoeken naar een ouder, waarvoor geldt dat de gebruiker heeft aangegeven dat hij er initieel behoefte aan heeft. Op deze manier wordt er gezocht naar een abstract concept, waarvan de gebruiker expliciet heeft aangegeven dat hij er behoefte aan heeft en zo dicht mogelijk bij de pagina ligt die wordt opgevraagd. Indien deze zoekmethode meerdere abstracte concepten oplevert, wordt het beste concept gepakt om het MRL-algoritme uit te voeren. Het beste concept wordt bepaald door deze concepten op dezelfde manier te rangschikken als in het MRL-algoritme.

Voorbeeld



De BB-waarden staan langs de conceptrelaties
Een abstract concept wordt aangeduid met Ci(BI, BVp)
Een pagina concept wordt aangeduid met Pi(BVp)

Figuur 21: Voorbeeld van een behoeftetoestand

Stel dat N = 5, C = C1.
Resultaat S wordt bepaald door de volgende stappen uit te voeren:
1. De kinderen van C1 worden gesorteerd met als resultaat {C3, C2, C4, C6}.
2. De kinderen van C3 worden gesorteerd en pagina's P2 en P3 worden toegevoegd aan S.
3. P1 van C2 wordt toegevoegd aan S (C2 is het volgende element in de verzameling van stap 1).
4. De kinderen van C4 worden gesorteerd met als resultaat {P7, P4, C5}. Merk op dat C5 achteraan komt, omdat geldt dat er volledig in behoefte is voldaan (BVp = 100).
5. Pagina's P7 en P4 worden toegevoegd aan S.
6. Er zijn 5 pagina's gevonden, S = {P2, P3, P1, P7, P4}.

5.5 Runtime Layer

De belangrijkste component uit de Runtime Layer is de RequestHandler. De RequestHandler handelt een request van de gebruiker af door triggering van regels bij de RuleHandler. De triggering is verdeeld in drie fasen, zie paragraaf 4.4.1. Bij het afhandelen van een request dient rekening gehouden te houden met de volgende problemen.

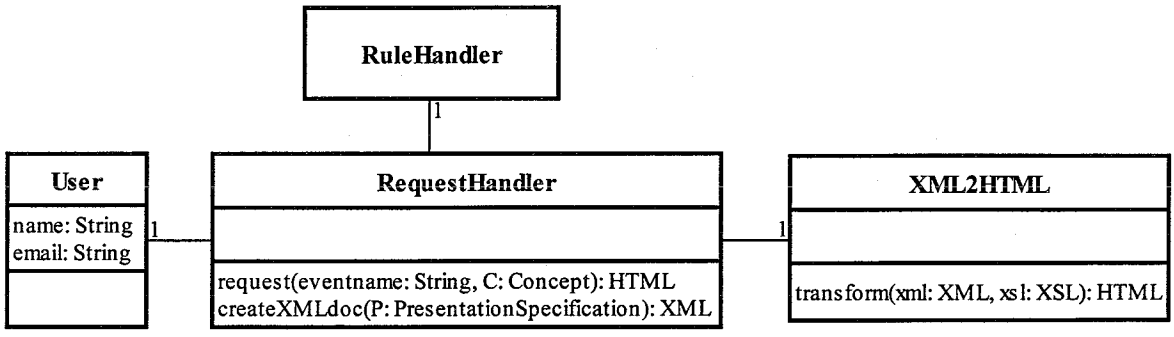
Synchronisatie
Als een gebruiker een request doet voordat een vorig request is afgewerkt kan er interferentie optreden. Een request van een gebruiker dient in zijn geheel te worden afgehandeld voordat een nieuwe request wordt afgehandeld. Dit kan worden bewerkstelligd door het proces van triggeren te synchroniseren. Bij synchronisatie van dit proces vormt de uitvoering van de drie fasen een atomaire actie. Het gevolg van deze synchronisatie is dat alle requests sequentieel afgehandeld worden. Om ervoor te zorgen dat de requests van verschillende gebruikers parallel afgehandeld kunnen worden, is gekozen om voor iedere gebruiker een RequestHandler te instantiëren.

Uitvoering regels in de post-fase
De regels in de post-fase dienen uitgevoerd te worden indien een concept gelezen is. Regels in de post-fase werken namelijk de behoeftetoestand van een gebruiker bij en kan alleen veranderden als een concept gelezen is. Voor het bepalen of een concept daadwerkelijk is gelezen, dient rekening gehouden te worden met de volgende problemen.

1. De maat die gehanteerd dient te worden of een concept wel of niet gelezen is
   Het meten van de tijd dat een concept aan de gebruiker gepresenteerd wordt is een voorbeeld van zo'n maat. Het probleem is echter dat een tijdsmaat niet aangeeft of het concept daadwerkelijk gelezen is. De enige mogelijkheid is het aan de gebruiker vragen of hij het concept gelezen heeft, maar hij zal over het algemeen niet bereid zijn deze vraag voor elk concept te beantwoorden.
2. Het afhandelen van een ongewenst concept
   Het systeem kan een concept aanbieden dat als ongewenst wordt beschouwd. De gebruiker zal in dit geval vrij snel een ander concept opvragen. Het onjuiste concept wordt als ongelezen beschouwd en het MRL-algoritme kan dit concept aanbieden als meest relevante link. Deze situatie wordt als onwenselijk beschouwd.

De oplossing die de RequestHandler hanteert om bovenstaande problemen op te lossen is het altijd uitvoeren van de post-fase nadat een concept is gepresenteerd. De RequestHandler houdt dus geen rekening met het daadwerkelijk gelezen zijn van een concept, omdat er geen goede maat is te geven en het MRL-algoritme een ongewenst concept opnieuw zou kunnen aanbieden.

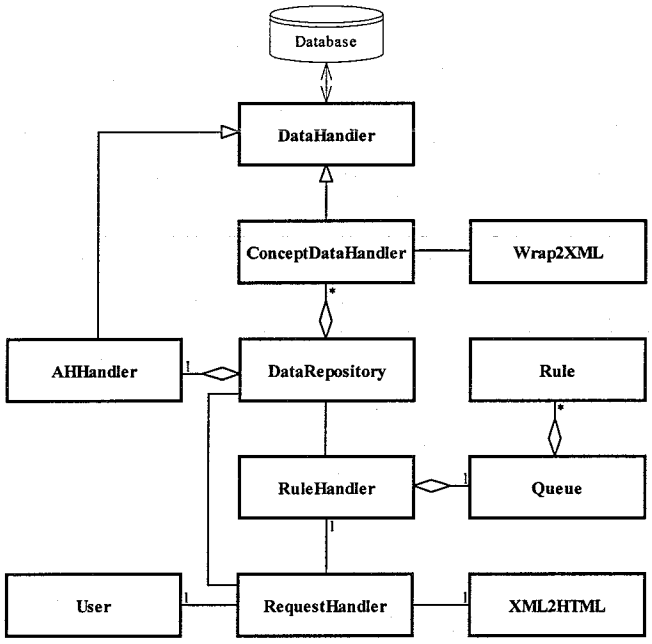
Objectmodel
De klasse RequestHandler wordt aangeroepen door de methode request. Deze methode roept de methode trigger bij de RuleHandler aan. De RuleHandler levert een Presentation Specification op, waarmee de RequestHandler een XML document samenstelt. De klasse XML2HTML transformeert dit document naar een HTML document.



Figuur 22: De klasse RequestHandler

5.6 Klassendiagram

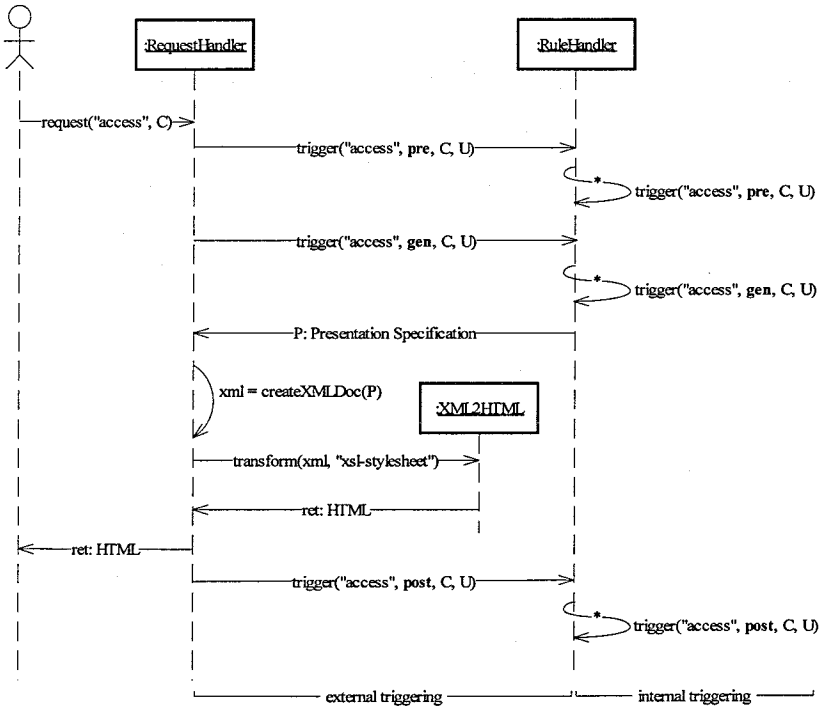
Figuur 23 geeft het complete klassendiagram met weglating van methoden en eigenschappen weer, dat tot stand komt door samenvoeging van de diagrammen die gepresenteerd zijn in de voorgaande paragrafen.



Figuur 23: Klassenmodel

5.6.1 Sequentie diagram: Triggering

Figuur 24 geeft het sequentiediagram van het afhandelen van request weer.

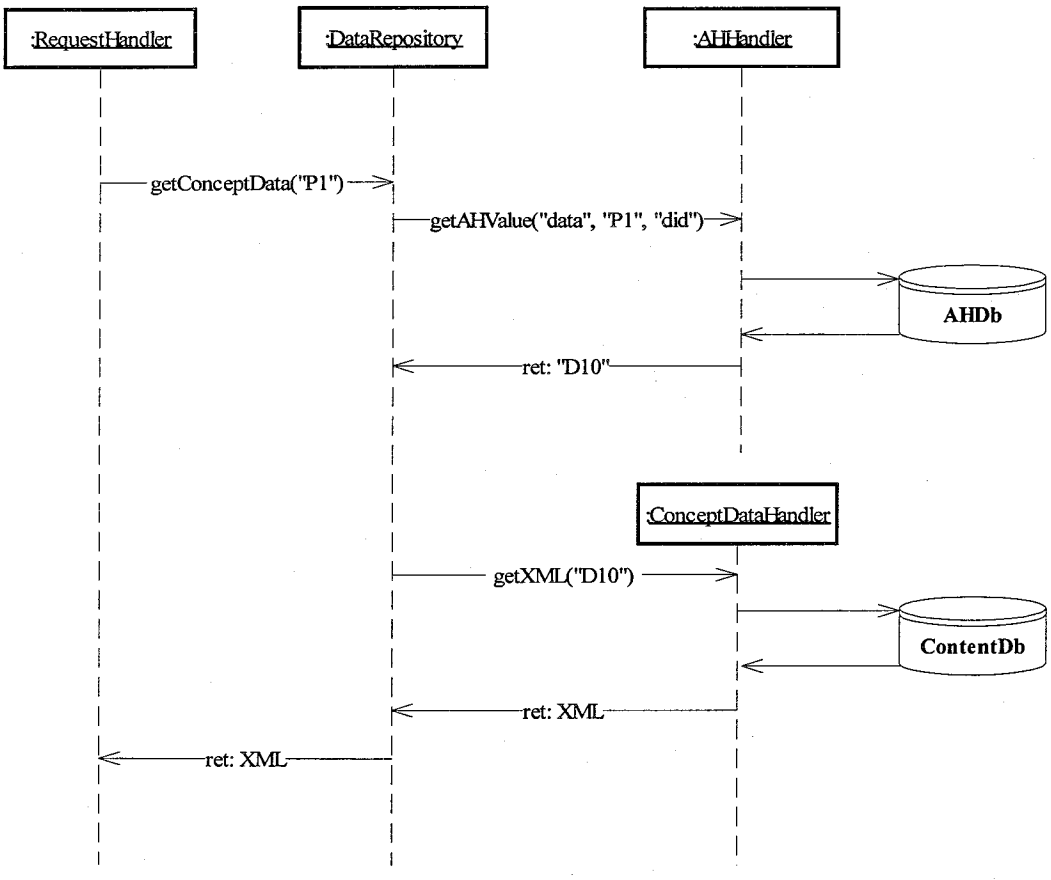


Figuur 24: Triggering

Bij het afhandelen van een request worden de volgende stappen uitgevoerd:
1. Concept C wordt opgevraagd door de methode request met parameters "access" en C aan te roepen.
2. De RequestHandler identificeert de gebruiker U en roept de methode trigger in de pre-fase bij de RuleHandler aan (external triggering).
3. De RuleHandler identificeert de getriggerde regels en voert deze uit. Tijdens de uitvoering kunnen andere regels getriggerd worden (internal triggering).
4. Na het uitvoeren van de regels in de pre-fase roept de RequestHandler de methode trigger in de gen-fase bij de RuleHandler aan.
5. Een XML document wordt gecreëerd aan de hand van de geretourneerde Presentation Specification P door de methode createXMLDoc(P) bij de RequestHandler aan te roepen.
6. Het XML document wordt aan de hand van een XSL stylesheet getransformeerd naar een HTML document door de methode transform aan te roepen bij de component XML2HTML.
7. Het HTML document wordt aan de gebruiker geretourneerd.

5.6.2 Sequentie diagram: Ophalen van data

Figuur 25 geeft het sequentiediagram voor het ophalen van data weer.



Figuur 25: Ophalen van data

Bij het ophalen van data worden de volgende stappen uitgevoerd:
1. De RequestHandler vraagt aan de DataRepository de data, die door concept P1 worden gerepresenteerd, door aanroep van de methode getConceptData("P1").
2. De DataRepository vraagt d.m.v. de methode getAHValue het bijbehorende dataID (did) op uit de AHDb. Deze waarde wordt opgevraagd aan de tabel "data" met sleutel "P1".
3. De AHHandler retourneert de waarde "D10" voor did.
4. De DataRepository roept de bijbehorende ConceptDataHandler aan d.m.v. de methode getXML met parameter "D10".
5. De ConceptDataHandler retourneert de content in XML formaat aan de DataRepository.
6. De DataRepository retourneert de content aan de RequestHandler.

6 Implementatie

Dit hoofdstuk behandelt de implementatie van het in hoofdstuk 5 gepresenteerde ontwerp. Alleen de belangrijkste aspecten van de implementatie, zoals gemaakte keuzes en gebruikte software van derden, worden besproken. De implementatie van ADIRA valt uiteen in een DBMS voor gegevensopslag en een applicatie die de beschreven functionaliteit implementeert.

6.1 DBMS

Het DBMS beheert de volgende twee typen databases.

1. Een AHDb
   In deze database worden gegevens over DM, UM en AM opgeslagen.
2. Een ContentDb
   In deze database wordt de content, die door het DM abstract gerepresenteerd wordt, opgeslagen.

Er is bij de opdrachtgever een KnowledgeDb beschikbaar die als ContentDb gebruikt kan worden. De AHDb is in Lotus Notes geïmplementeerd, omdat de KnowlegdeDb in Lotus Notes is geïmplementeerd.

Lotus Notes
Lotus Notes wordt aangeduid als een groupware omgeving. Groupware stelt gebruikers in staat in teamverband te werken, onafhankelijk van tijd en fysieke locatie. Een Lotus Notes omgeving bestaat onder andere uit één of meerdere fysieke servers (Domino Server) waarop de applicaties (databases) zijn geïnstalleerd. De server is toegankelijk via een client die via het netwerk in verbinding staat met de server. Er zijn vier soorten clients: de Lotus Notes Client, de Domino Designer Client, de Domino Administrator en de browser.

6.1.1 ContentDb

De informatie die de opdrachtgever wil publiceren is opgeslagen in een KnowledgeDb. De KnowledgeDb is geschikt als ContentDb, omdat de gegevens in de KnowlegdeDb hiërarchisch te structureren zijn zoals besproken is in paragraaf 2.1. De structuur van de informatie wordt gerepresenteerd in het DM door een hoofd- en subonderwerp af te beelden op een abstract concept, een samengesteld pagina concept op een artikel en een atomair concept op een fragment van een artikel. De structuur van de KnowledgeDb heeft de volgende additionele eigenschappen [GV00]:

1. Ieder artikel is deel van één (sub) onderwerp.
2. Ieder (sub) onderwerp is deel van één ander (sub) onderwerp.

De bovenstaande eigenschappen impliceren dat ieder concept in het DM precies één ouder heeft. De nadelen m.b.t. determinisme en terminatie van de regels, zoals respectievelijk besproken in paragrafen 4.3.4.1 en 4.3.4.2, zijn niet aan de orde vanwege deze eigenschappen.

6.1.2 AHDb

De AHDb is een Lotus Notes database en daarom een niet-relationele database, waardoor men in andere termen spreekt over het ontwerp. In onderstaande tabel wordt weergegeven hoe deze termen overeenkomen met het ontwerp.

Klassiek db ontwerp	Lotus Notes db ontwerp	E-R Model
Veld	Veld	Attribuut
Recorddefinitie	Formulier	Entiteit
Record	Document	
Query	View (zonder join operatie)	

Voorbeeld
Het AM is gemodelleerd in de entiteiten Rule, Condition, SetOfConcepts en Concept (zie paragraaf 5.2.3). Dit is in Lotus Notes geïmplementeerd met een formulier genaamd Rule. Vanwege het feit dat Condition met Rule een 1 op 1 relatie heeft, is de entiteit Condition samen met de entiteit Rule op één formulier geïmplementeerd. De entiteit SetOfConcepts is tevens geïmplementeerd in het formulier, omdat het mogelijk is een veld meerdere waarden te geven.

Beperking
Bij implementatie van een Lotus Notes database dient er rekening mee gehouden te worden dat er geen relationele join uitgevoerd kan worden door het DBMS. Het ontbreken van een relationele join heeft consequenties voor de implementatie van het MRL-algoritme. Dit algoritme maakt gebruik van een rangorde die tot stand komt door attributen van records uit verschillende tabellen te joinen en te sorteren. De join operaties, die nodig zijn voor de totstandkoming van de rangorde, worden niet door het DBMS uitgevoerd maar door de DataHandler.

6.2 Applicatie

De implementatie van de Within-component Layer, de Storage Layer en de Runtime Layer kan op een aantal manieren gerealiseerd worden. De belangrijkste zijn de implementatie d.m.v. Lotus Notes en de implementatie d.m.v. een servlet. Een servlet is een applicatie die gebaseerd is op Java.

Java
Java is een programmeertaal die op C++ lijkt. In Java kan de programmeur zelfstandige applicaties, applets en servlets maken. Een Java compiler vertaalt de programmacode naar bytecode, die machine onafhankelijk is en verstuurd kan worden naar een ander computersysteem, om daar te worden geïnterpreteerd of te worden gecompileerd naar machine-instructies voor dat systeem. Een applet is een Java programma dat uitgevoerd wordt door een browser op het moment dat de browser een HTML document inlaadt dat een applet tag bevat.

Servlet
De Java Servlet technologie [SM01] biedt een consistent mechanisme voor het uitbreiden van de functionaliteit van een webserver. Bij een servlet kan gedacht worden aan een applet zonder user interface. Servlets bieden een component gebaseerde, platformonafhankelijke methode voor web gebaseerde applicaties. Een servlet is te vergelijken met CGI-script. Het grote verschil met CGI-script is dat een servlet niet per gebruiker een proces uitvoert. Een servlet blijft in het geheugen, zodat een interne status bijgehouden kan worden.

Voor de implementatie van de applicatie wordt gebruik gemaakt van een servlet i.p.v. Lotus Notes om de volgende redenen:
1. In Lotus Notes wordt per request een proces uitgevoerd.
2. Er kan geen interne status in het geheugen bijgehouden worden. Per request dienen alle gegevens van een gebruiker opnieuw geïnitialiseerd te worden.
3. Het onderscheid tussen de applicatie en de data is vaag, omdat de applicatie en de data in elkaar verweven zijn in één databasebestand.

Een webserver is noodzakelijk om een servlet uit te voeren. De Domino Server kan dienst doen als webserver en is dus in staat om servlets uit te voeren.

6.2.1 DataHandler

De DataHandler verzorgt de koppeling van de applicatie met het DBMS. De implementatie van deze component is dus afhankelijk van het gebruikte DBMS. De koppeling met Lotus Notes kan worden gerealiseerd m.b.v. het Notes pakket of het NSCO pakket.

Notes pakket
Dit pakket bevat klassen die de koppelingen met een lokale Domino Server realiseert en is met name bedoeld voor de ontwikkeling van servlets. De toegang tot een database is in Lotus Notes afgeschermd door een Access Control List (ACL). Door de servlet op te nemen in de ACL van een database verkrijgt de servlet rechten tot lezen en/of schrijven van de data uit die database.

NSCO pakket
Het NSCO pakket bevat klassen die het mogelijk maken om een lokale Java applicatie een CORBA gebaseerde remote-call tot een andere Domino Server te laten uitvoeren. Het is tevens mogelijk om d.m.v. dit pakket een local-call te maken, maar is echter veel trager dan het Notes pakket.

Er is voor gekozen om gebruik te maken van het Notes pakket, omdat de servlet alleen local-calls behoeft te doen aan de Domino Server. Bij de implementatie van de DataHandler zijn een aantal performanceproblemen opgelost, zie bijlage C.

6.2.2 RuleHandler

De implementatie van de RuleHandler maakt gebruik van het Java Compiler Compiler (JavaCC) pakket [M01] om de onderdelen boolean expression, action en eventcall van een adaptatieregel te parsen.

JavaCC pakket
JavaCC is zowel een lexical scanner als een grammatica parser. Terminals zoals keywords en speciale karakters moeten worden gedefinieerd. Er kan tevens gedefinieerd worden hoe spaties, tabs, einde regel en commentaar gerepresenteerd moeten worden door karakters. Aan de hand van een taal die gespecificeerd is in EBNF notatie genereert JavaCC een klasse die de gespecificeerde taal kan parsen.

6.2.3 XML2HTML

Voor de transformatie van een XML document naar een HTML document, d.m.v. een XSL stylesheet, wordt gebruik gemaakt van het XSLT Compiler (XSLTC) pakket [SMDC01].

XSLTC pakket
XSLTC is een tool om snelle en compacte klassen te creëren voor het transformeren van XML documenten. De transformatie vindt plaats aan de hand van een XSL stylesheet. XSLTC parst de input stylesheet en creëert een klasse met transformatie instructies gespecificeerd door de stylesheet.

7 Oplevering

Dit hoofdstuk bespreekt de oplevering van het systeem en de resultaten van de tests die daaraan vooraf zijn gegaan. Ter voorbereiding van de tests zijn een aantal regels in het Adaptation Model gespecificeerd, om ADIRA de gewenste adaptatie te kunnen laten uitvoeren.

Voorbeeld
Als een concept opgevraagd is, wordt het aantal keer dat het concept gelezen is verhoogd met 1. Dit geldt ook voor alle ouders van het opgevraagde concept.

Event	Phs	Subphs	Soc	Boolean Expression	Action	Eventcall
access	post	external	all	true	UC.Read = UC.Read + 1	forall p in C.Parents do updateRead(p) od
updateRead	post	internal	all	true	UC.Read = UC.Read + 1	forall p in C.Parents do updateRead(p) od

Voor de overige regels wordt er verwezen naar bijlage C.

Gebruikerstest
Er is een gebruikerstest uitgevoerd door enkele medewerkers van KGP. Deze test heeft zich voornamelijk gericht op het outputproces van ADIRA. Daarbij is getoetst of ADIRA gebruiksvriendelijk is en de gebruiker de juiste informatie aanbiedt.

Door middel van de gebruikerstest zijn de volgende problemen geïdentificeerd:

1. Omschrijving bij een MRL-link
   Het is gebleken dat plaatsing van een korte samenvatting van het bijbehorende concept wenselijk is.
2. Rubriceren van MRL-links
   Het is gebleken dat het wenselijk is om de links te rubriceren.
3. Historie
   Het is gebleken dat het terugvinden van gelezen artikelen moeilijk is. Dit komt omdat het MRL-algoritme steeds de meest relevante links oplevert waarbij een gelezen artikel als minder relevant wordt beschouwd.
4. Zoeken
   Naast de aangeboden links is gebleken dat de gebruiker wil zoeken door de gehele verzameling van informatie.

Performancetest
Er is een performancetest uitgevoerd met 10.000 artikelen, 15 hoofdonderwerpen en 30 subonderwerpen. Het is gebleken dat de gemiddelde responsetijd korter is dan 1,5 seconde. Dit is acceptabel, gezien het feit dat deze test is uitgevoerd op een pentium-II 300 MHz.

Invoering
De functionaliteit beschreven in het ontwerp is volledig geïmplementeerd en ingevoerd op het intranet van KGP. Tevens zijn de problemen die naar voren zijn gekomen bij de gebruikerstest opgelost.

8 Applicatiedomein

In dit hoofdstuk worden enkele belangrijke aspecten van ADIRA besproken om vervolgens te bepalen tot welke categorie van adaptieve hypermedia systemen ADIRA behoort.

De generiekheid van ADIRA komt tot uitdrukking in de volgende aspecten:

1. Adaptatie
   De wijze waarop ADIRA adaptatie toepast is op eenvoudige wijze te beïnvloeden door de regels te wijzigen die gedefinieerd zijn in het AM. Voor een auteur zonder enige informatica-ervaring zijn de regels echter niet eenvoudig te bewerken. Dit probleem kan opgelost worden door een tool te ontwikkelen die het bewerken van deze regels ondersteunt.
2. Informatie
   ADIRA kan met grote hoeveelheden informatie werken die te rubriceren is. De informatie moet zodanig geschreven zijn dat er geen voorkennis nodig is. Er kunnen meerdere ContentDb's gebruikt worden als input voor de informatie.
3. Presentatie
   De presentatie is eenvoudig aan te passen door de XSL-stylesheet te bewerken. Een auteur zonder enige informatica-ervaring kan de stylesheet echter niet eenvoudig bewerken.
4. Platform
   ADIRA is platformonafhankelijk omdat de programmeertaal Java en het DBMS Lotus Notes vrijwel door alle platformen ondersteund worden.
5. DBMS
   De AHDb is geïmplementeerd in Lotus Notes en is als zodanig afhankelijk van dit DBMS. Er geldt voor ADIRA, dat alleen de AHDb en de DataHandler aangepast moeten worden indien er een ander DBMS wordt gebruikt.

In de literatuur worden door Brusilovsky [B96] de volgende zes categorieën adaptieve hypermedia systemen onderscheiden:

1. Educatieve hypermedia systemen
   Deze hypermedia systemen ondersteunen student gerichte verwerving van de leerstof. De grootte van de hyperspace is constant, omdat er geen informatie wordt toegevoegd.
2. On-line informatie systemen
   Deze systemen verschaffen referentiele toegang tot informatie voor gebruikers met verschillende kennisniveaus. De hyperspace is groot en redelijk constant.
3. On-line help systemen
   Deze systemen dienen als on-line informatie systemen voor computer applicaties, zoals spreadsheets, die nodig zijn om gebruikers van die applicaties te ondersteunen. De hyperspace varieert van groot tot klein en is constant.
4. Informatie retrieval systemen
   Deze systemen combineren traditionele informatie retrieval technieken met een hypertext-achtige toegang tot de indextermen van de documenten. Tevens bieden deze systemen hulp bij de navigatie door het voorstellen van de meest relevante links. De hyperspace is groot en heeft een dynamisch karakter.
5. Institutionele hypermedia
   Deze systemen bieden alle informatie die nodig is om het werk van een instituut te ondersteunen, bijvoorbeeld een ziekenhuis. De hyperspace is redelijk groot en redelijk constant.
6. Gepersonaliseerde views
   Deze systemen bevatten grote hoeveelheden ongestructureerde informatie die een ongelimiteerde hyperspace vormen. Om gebruikers te beschermen tegen de complexiteit van deze hyperspace worden er gepersonaliseerde views gemaakt, die van toepassing zijn op één van de doelen of interesses die gerelateerd zijn aan het werk van de gebruiker.

Er dient opgemerkt te worden dat de genoemde categorieën elkaar niet uitsluiten. De categorie informatie systemen waarmee ADIRA de meeste overeenkomsten heeft, is de categorie information retrieval systemen. De overeenkomsten zijn:

1. De navigatievrijheid wordt ingeperkt door alleen de meest relevante links aan te bieden (zie paragraaf 5.4.3).
2. De hyperspace is groot (zie paragraaf 2.2.1), maar niet ongelimiteerd zoals bij gepersonaliseerde views.
3. De hyperspace heeft een dynamisch karakter (zie paragraaf 2.2.1).

Het is mogelijk om ADIRA toe te passen als educatief of on-line helpsysteem. Bij deze systemen kan het nodig zijn dat de gebruiker voorkennis heeft van bepaalde zaken om een artikel te kunnen begrijpen. De gebruiker zal eerst genavigeerd dienen te worden naar artikelen, die ervoor zorgen dat hij de benodigde voorkennis vergaart. Dit kan bewerkstelligd worden door ADIRA uit te breiden met de conceptrelatie prerequisite. Een link naar een bepaald concept wordt gemarkeerd als "not-ready-to-read" totdat de benodigde voorkennis is vergaard. Op het moment dat de benodigde voorkennis vergaard is, verandert de status van de link naar "ready-to-read". Door op de links een juiste linkadaptatie techniek toe te passen (bijvoorbeeld link-hiding voor links gemarkeerd met "not-ready-to-read") wordt de gebruiker genavigeerd naar de links die gemarkeerd zijn als "ready-to-read".

9 Conclusie

Dit hoofdstuk bespreekt de belangrijkste aspecten van de realisatie van de opdracht. Daarnaast worden onderdelen die voor verbetering in aanmerking komen besproken.

ADIRA is ontworpen om aan de opdracht, het aanbieden van grote hoeveelheden informatie op behoefte van de gebruiker, te voldoen. Het inputproces van ADIRA zorgt ervoor dat informatie uit verschillende bronnen verzameld wordt. ADIRA focust zich op het outputproces dat ervoor zorgt dat de verzamelde informatie op behoefte aangeboden kan worden. Dit proces is gerealiseerd d.m.v. een adaptief hypermedia systeem.
Voor de architectuur van ADIRA is het Adaptive Hypermedia Application Model (AHAM) als referentie model gekozen. De belangrijkste onderdelen van AHAM zijn het Domain Model, het User Model en het Adaptation Model. Het Adaptation Model bestaat voornamelijk uit regels die het aanbieden van informatie op behoefte ondersteunen. De belangrijkste regels zijn de regels die de behoeftetoestand van een gebruiker bijwerken en regels die aan de hand van de behoeftetoestand relevante informatie selecteren. Het selecteren van de meest relevante pagina's wordt gerealiseerd door het MRL-algoritme. ADIRA zorgt ervoor dat de juiste regels geïdentificeerd en uitgevoerd worden om de adaptatie mogelijk te maken. ADIRA garandeert dat de uitvoering van de regels termineert en het resultaat deterministisch bepaald wordt.
Door de regels in het Adaptation Model te wijzigen is het adaptatiegedrag van ADIRA volledig beïnvloedbaar. Het gebruik van stylesheets maakt het mogelijk de presentatie van de informatie volledig aan te passen. De implementatie van ADIRA is volledig gerealiseerd en daarmee is aan de opdracht voldaan.

Toekomstig werk
De volgende onderdelen komen voor verbetering in aanmerking:

1. Auteurstool
   De tool dient ter ondersteuning van de auteur (zonder informatica-ervaring) bij het specificeren van regels in het Adaptation Model en het specificeren van de uiteindelijke presentatie van de informatie.
2. Vergoten applicatiedomein
   De categorie hypermedia systemen, waarvan ADIRA deel uit maakt, is geïdentificeerd als informatie retrieval systemen. Het applicatiedomein kan vergroot worden door voorbeeld de conceptrelatie prerequisite toe te voegen. Door deze toevoeging kan ADIRA dienst doen als educatief of on-line helpsysteem.

Bibliografie

[B96] Brusilovsky, P., Methods and techniques of adaptive hypermedia, http://www.contrib.andre.cmu.edu/~plb/UMUAI.ps, 1996.

[D98] De Bra, P., 2L690: Hypermedia Structures and Systems, http://wwwis.win.tue.nl/2L690, 1998.

[DBH99] De Bra, P., Brusilovsky, P., Houben, G.J., Adaptive Hypermedia: From Systems to Framework, http://wwwis.win.tue.nl/~debra/survey/debra.ps, 1999.

[DHW99] De Bra, P., Houben, G.J., Wu, H., AHAM: A Dexter-based Reference Model for Adaptive Hypermedia, http://wwwis.win.tue.nl/ht99/ht99.ps, 1999.

[EP98] Eriksson, H., Penker, M., UML Toolkit, Wiley Computer Publishing, 1998.

[GV00] Geraats, J., Van Poppel, M., Het delen van kennis, een praktische benadering, Afstudeerverslag, 2000.

[HS90] Halasz, F., Schwartz, M., The Dexter Hypertext Reference Model, NIST Hypertext Standardization Workshop, February 1990, pp 95-133.

[II96] ISO/IEC 14977, Extended BNF, http://www.iso.ch/cate/d26153.html, 1996.

[K00] Kamps, E.P.G., Exceldus, Intern document, 2000.

[M01] Metamata, JavaCC - The Java Parser Generator, http://www.webgain.com/products/metamata/java_doc.html, 2001.

[O94] O'Neill, P., Database Principles Programming Performance, Morgan Kaufmann Publicers, Inc., 1994.

[SK89] Shneiderman, B., Kearsley G., Hypertext Hands-On!: An Introduction to a New Way of Organizing and Accessing Information, Addison Wesley, 1989.

[SM01] Sun Microsystems, Java(™) Servlet Technology, http://java.sun.com/products/servlet, 2001.

[SMDC01] Sun Microsystems Developer Connection, XSLT Compiler, http://www.sun.com/xml/developers/xsltc/, 2001.

[WDAH00] Wu, H., De Bra, P., Aerts, A., Houben, G.J., Adaption Control in Adaptive Hypermedia Systems, http://wwwis.win.tue.nl/~debra/ah2000/wu.ps, 2000.

[WHD98] Wu, H., Houben, G.J., De Bra, P., AHAM: A Reference Model to Support Adaptive Hypermedia Authoring, http://wwwis.win.tue.nl/~debra/infwet98/paper.pdf, 1998.

[WHD99] Wu, H., Houben, G.J., De Bra, P., Authoring support for Adaptive Hypermedia Applications, http://wwwis.win.tue.nl/edmedia99/ED-final.ps, 1999.

Bijlage A: EBNF specificatie adaptatieregels

Deze bijlage geeft de specificatie van de adaptatieregels in EBNF notatie weer. Achtereenvolgens worden de verschillende onderdelen van de specificatie behandeld. De volledige specificatie van de adaptatieregels is weergegeven in figuur 27.

Whitespace
Als een non-terminal <A> direct gevolgd wordt door non-terminal <B>, is het niet duidelijk waar <A> eindigt en <B> start. Dit geldt alleen als <A> en <B> gebruik maken van dezelfde terminals. De non-terminal <W> is geïntroduceerd om twee non-terminals van elkaar te scheiden. De notatie om non-terminals <A> en <B> te scheiden is <A> <W> <B>. De terminals die de non-terminal <W> specificeren mogen niet gebruikt worden in andere non-terminals. In de specificatie van de adaptatieregels zoals weergegeven in figuur 27 is de non-terminal <W> niet in de rechterzijde van de produktieregels opgenomen om de overzichtelijkheid te bewaren.

Gereserveerde woorden
Een woord kan bestaan uit een reeks van letters, getallen of een underscore. De reeks die het woord definieert dient te starten met een letter. Een gereserveerd woord is een woord dat de applicatie voor interne verwerking van de adaptatieregels gebruikt. Een gereserveerd woord mag niet gebruikt worden om een variabele of een functie te specificeren. De gereserveerde woorden kunnen bestaan uit:

1. Basiswaarden
   Non-terminals <TRUE> en <FALSE> definiëren de boolean waarden "true" en "false".
2. Expressie operatoren
   Non-terminal <SQRT> gedefinieerd door het woord "sqrt" wordt gebruikt om de wortel operatie aan te duiden.
3. Commando's
   Een voorbeeld van een commando non-terminal is <FORALL>, die aangeeft dat er een repetitie uitgevoerd dient te worden.

Speciale karakters
Een speciaal karakter wordt net zoals een gereserveerd woord gebruikt voor de interne verwerking van de adaptatie regels. Voorbeelden van speciale karakters zijn "{" en "}" die een verzameling weergegeven. Door het gebruik van deze accolades worden de non-terminals tussen de accolades als elementen van een verzameling gezien.

Typen
Voor de interne verwerking van een expressie dient bepaald te worden wat het type is van de waarden die in de expressie gebruikt worden. Het type wordt afgeleid aan de notatie van de waarde. Een aantal voorbeelden waaruit de typen Integer, Float, String en Boolean volgen:

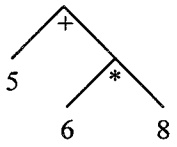
1. Het type van 0 is Integer.
2. Het type van 0.0 is Float.
3. Het type van "nul" is String.
4. Het type van true is Boolean.

Operatoren
Een operator bestaat uit één of twee speciale karakters (zoals + of >=) of uit een gereserveerd woord (zoals "true"). De volgende operatoren zijn gedefinieerd:

1. Rekenkundige operator
   Een rekenkundige operator is gedefinieerd tussen waarden van het type Integer of Float.
2. Logische operator
   Een logische operator is gedefinieerd tussen waarden van het type Boolean.
3. String operator
   Een string operator is gedefinieerd tussen waarden van het type String. De + operator voegt twee waarden van type String samen. Merk op dat de + operator ook gedefinieerd is als rekenkundige operator.
4. Vergelijking operator
   Een vergelijking operator wordt gebruikt om twee waarden van hetzelfde type ten opzichte van elkaar te vergelijken.
5. Assignment operator
   Een toekenning van een waarde aan een variabele wordt door de assignment operator gedefinieerd.

Expressies

Een expressie bestaat uit een aantal waarden, variabelen, functies en operatoren. Een voorbeeld van een expressie is 5 + 6 * 8.



Figuur 26: Expressieboom van 5 + 6 * 8

Figuur 26 maakt duidelijk dat er rekening gehouden dient te worden met de prioriteiten van de verschillende operatoren. De volgende tabel geeft deze prioriteiten weer, waarbij de prioriteit toeneemt in de richting van de pijlen.

Logisch	Vergelijking	Rekenkundig
| (or)	<, <=, ==, >=, >, !=	+, -
& (and)		/
! (not)		*
		power
		sqrt

In de bovenstaande tabel is de assignment operator niet opgenomen, omdat het geen expressie operator is. De vergelijking operatoren kennen geen onderlinge prioriteit, net zoals de rekenkundige operatoren + en -. Een prioriteit kan altijd overschreven worden door gebruik te maken van haakjes. De expressie (5 + 6) * 8 is dus een andere expressie dan 5 + 6 * 8, die als 5 + (6 * 8) geïnterpreteerd wordt.

Een expressie kan gebruik maken van variabelen en functies die gedefinieerd zijn in de applicatie via een program object. Dit object stelt de interne naamgeving van een variabele of een functie voor.

Adaptatieregels

Een adaptatieregel bestaat uit de non-terminals <CONDITION>, <ACTION> en eventueel <EVENTCALL>.

```
<RULE> ::= <CONDITION> <ACTION> (<EVENTCALL>)?
```

Non-terminal <EVENTCALL> geeft met het gereserveerde woord <FORALL> het commando van een repetitie weer en is als volgt gedefinieerd.

```
<EVENTCALL> ::= <FORALL> <IDENTIFIER> <IN> <EXPRESSION> <DO> <EVENT> <OD>
```

Volledige specificatie adaptatieregels

Figuur 27 geeft de volledige specificatie van de adaptatieregels weer.

Whitespace:

```
<W>  ::=  (" " | "\t")+
```

Gereserveerde woorden:

```
<TRUE>      ::=  "true"
<FALSE>     ::=  "false"
<PRE>       ::=  "pre"
<GEN>       ::=  "gen"
<POST>      ::=  "post"
<INTERNAL>  ::=  "internal"
<EXTERNAL>  ::=  "external"
<ALL>       ::=  "all"
<FORALL>    ::=  "forall"
<IN>        ::=  "in"
<DO>        ::=  "do"
<OD>        ::=  "od"
<POWER>     ::=  "power"
<SQRT>      ::=  "sqrt"
```

Speciale karakters:

```
<SEMICOLON>  ::=  ";"
<DOT>        ::=  "."
<COMMA>      ::=  ","
<LPAREN>     ::=  "("
<RPAREN>     ::=  ")"
<LBRACE>     ::=  "{"
<RBRACE>     ::=  "}"
<UNDERSCORE> ::=  "_"
<QUOTE>      ::=  "\""
```

Typen:

```
<CHAR>       ::=  ["A" - "z"]
<DIGIT>      ::=  ["0" - "9"]
<IDENTIFIER> ::=  <CHAR> (<CHAR> | <DIGIT> | <UNDERSCORE>)*
<TEXT>       ::=  <QUOTE> (~["\""])* <QUOTE>
<INTEGER>    ::=  <DIGIT>+
<FLOAT>      ::=  <INTEGER> | <INTEGER> <DOT> <INTEGER> | <DOT> <INTEGER>
<NUMBER>     ::=  <FLOAT>
<BOOLEAN>    ::=  <TRUE> | <FALSE>
```

Rekenkundige operatoren:

```
<POWER>      ::=  Reeds gedefinieerd als gereserveerd woord.
<MULTIPLY>   ::=  "*"
<DIVIDE>     ::=  "/"
<SQRT>       ::=  Reeds gedefinieerd als gereserveerd woord.
<PLUS>       ::=  "+"
<MINUS>      ::=  "-"
```

Logische operatoren:

```
<AND>        ::=  "&"
<OR>         ::=  "|"
<NOT>        ::=  "!"
```

Vergelijking operatoren:

```
<LESS_THAN>                ::=  "<"
<LESS_THAN_OR_EQUAL_TO>    ::=  "<="
<EQUAL_TO>                 ::=  "=="
<GREATER_THAN_OR_EQUAL_TO> ::=  ">="
<GREATER_THAN>             ::=  ">"
<NOT_EQUAL_TO>             ::=  "!="
```

Assignment operator:

```
<BECOMES>    ::=  "="
```

Expressie (aangepast op prioriteiten en haakjes structuur):

```
<EXPRESSION>      ::=  <PRIORITY_2_LOGIC>
<PRIORITY_2_LOGIC>::=  <PRIORITY_1_LOGIC> <OR> <PRIORITY_1_LOGIC>
<PRIORITY_1_LOGIC>::=  <COMPARISON> <AND> <COMPARISON>
<COMPARISON>      ::=  <PRIORITY_4_EXPR>
                       (
                        <LESS_THAN>
                       |
                        <LESS_THAN_OR_EQUAL_TO>
                       |
                        <EQUAL_TO>
                       |
                        <GREATER_THAN_OR_EQUAL_TO>
                       |
                        <GREATER_THAN>
                       |
                        <NOT_EQUAL_TO>
                       )
                       <PRIORITY_4_EXPR>
<PRIORITY_4_EXPR> ::=  <PRIORITY_3_EXPR>
                       (
                        <PLUS>
                       |
                        <MINUS>
                       )
                       <PRIORITY_3_EXPR>
<PRIORITY_3_EXPR> ::=  <PRIORITY_2_EXPR> <DIVIDE> <PRIORITY_2_EXPR>
<PRIORITY_2_EXPR> ::=  <PRIORITY_1_EXPR> <MULTIPLY> <PRIORITY_1_EXPR>
<PRIORITY_1_EXPR> ::=  <UNARY_TERM> <POWER> <UNARY_TERM>
<UNARY_TERM>      ::=   <MINUS> <ELEMENT>
                       |
```

```
                                  <SQRT> <ELEMENT>
                                 |
                                  <NOT> <ELEMENT>
                                 |
                                  <ELEMENT>
<ELEMENT>          ::=           <NUMBER>
                                 |
                                  <TEXT>
                                 |
                                  <LPAREN> <EXPRESSION> <RPAREN>
                                 |
                                  <BOOLEAN>
                                 |
                                  <FUNTIONCALL>
                                 |
                                  <PROGRAMOBJECT>
                                 |
                                  <IDENTIFIER>
<PROGRAMOBJECT>    ::=           <IDENTIFIER> <DOT> <IDENTIFIER>
<FUNTIONCALL>      ::=           <PROGRAMOBJECT>
                                 <LPAREN>
                                 (
                                  (<EXPRESSION> | <SET>)
                                  (<COMMA> (<EXPRESSION> | <SET>) )*
                                 )?
                                 <RPAREN>
<SET>              ::=           <LBRACE> <EXPRESSION> (<COMMA> <EXPRESSION>)* <RBRACE>
```

Adaptatieregels:

```
<RULE>             ::=           <CONDITION><ACTION>(<EVENTCALL>)?
<CONDITION>        ::=           <EVENT><PHASE><SUBPHASE><SOC><BOOLEAN EXPRESSION>
<BOOLEAN EXPRESSION> ::=         <EXPRESSION>
<PHASE>            ::=           <PRE> | <GEN> | <POST>
<SUBPHASE>         ::=           <INTERNAL> | <EXTERNAL>
<SOC>              ::=           <ALL> | <LBRACE> <IDENTIFIER> (<COMMA> <IDENTIFIER>)* <RBRACE>
<ACTION>           ::=           (<ASSIGNMENT> | <FUNCTIONCALL>)
                                 (<SEMICOLON> (<ASSIGNMENT> | <FUNCTIONCALL>))*
<ASSIGNMENT>       ::=           <PROGRAMOBJECT> <BECOMES> <EXPRESSION>
<EVENT>            ::=           <IDENTIFIER>
                                 <LPAREN> <EXPRESSION> (<COMMA> EXPRESSION>)? <RPAREN>
<EVENTCALL>        ::=           <FORALL> <IDENTIFIER> <IN> <EXPRESSION> <DO> <EVENT> <OD>
```

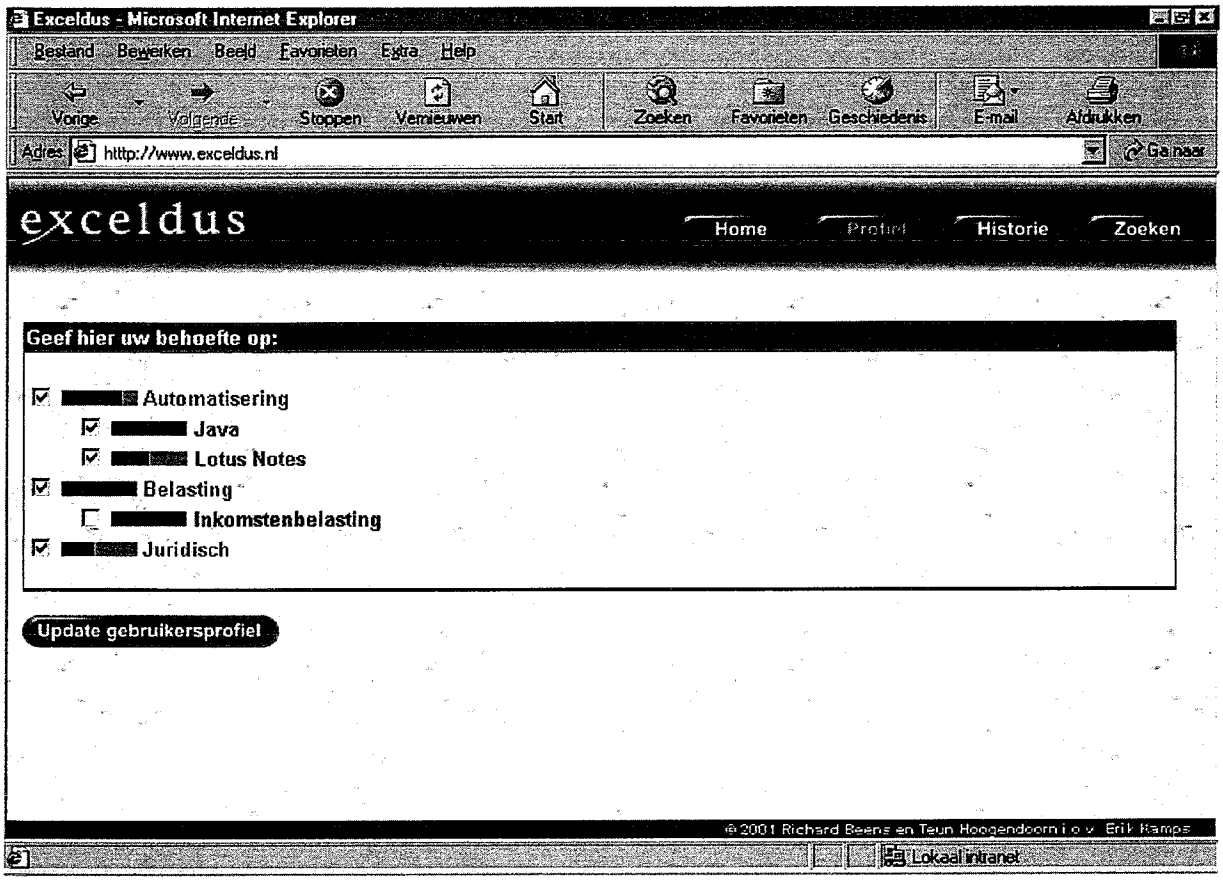
Figuur 27: EBNF specificatie adaptatieregels

Bijlage B: Website Exceldus

Deze bijlage bespreekt de verschillende onderdelen van de website van Exceldus. In paragraaf B.1 wordt het gebruikersprofiel besproken. Paragraaf B.2 bespreekt de homepage. De historiefunctionaliteit wordt besproken in paragraaf B.3 en de zoekfunctionaliteit in paragraaf B.4.

B.1 Gebruikersprofiel

Een gebruiker dient zijn behoefte op te geven door één of meerdere concepten aan te vinken in zijn gebruikersprofiel. Figuur 28 geeft een weergave van het gebruikersprofiel.

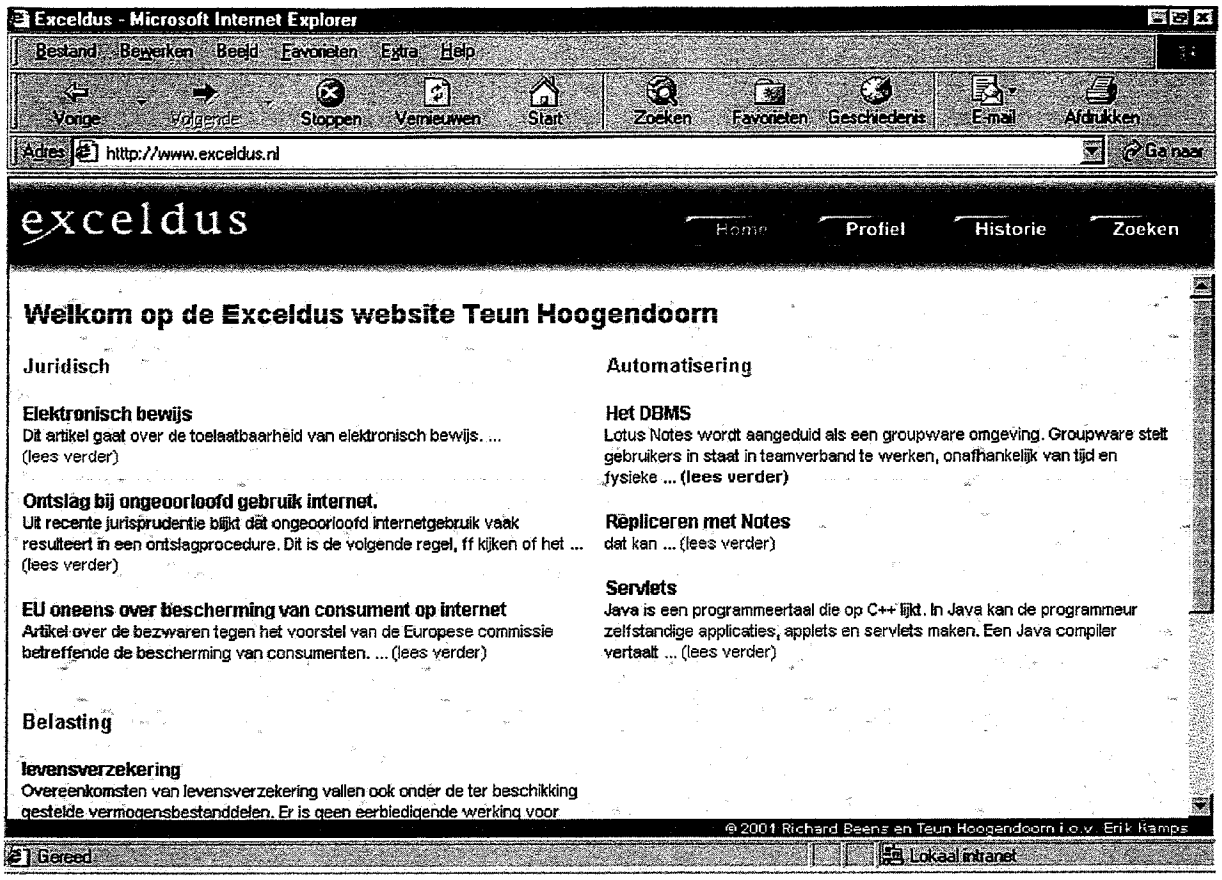


Figuur 28: Het gebruikersprofiel

Het gebruikersprofiel is hiërarchisch ingedeeld volgens de concepthiërarchie in het DM. Deze hiërarchische indeling wordt gerealiseerd door regels in het AM, zie bijlage C. Door op een concept te klikken verschijnen de kinderen van het concept en kan hieraan behoefte opgegeven worden. Het concept "Automatisering" wordt getoond met zijn kinderen "Java" en "Lotus Notes". De balken die aan de linkerzijde van het concept getoond worden geven de BVp-waarde in een blauwe kleur weer, zodat de gebruiker kan zien in welke mate er in zijn behoefte voldaan is volgens ADIRA.

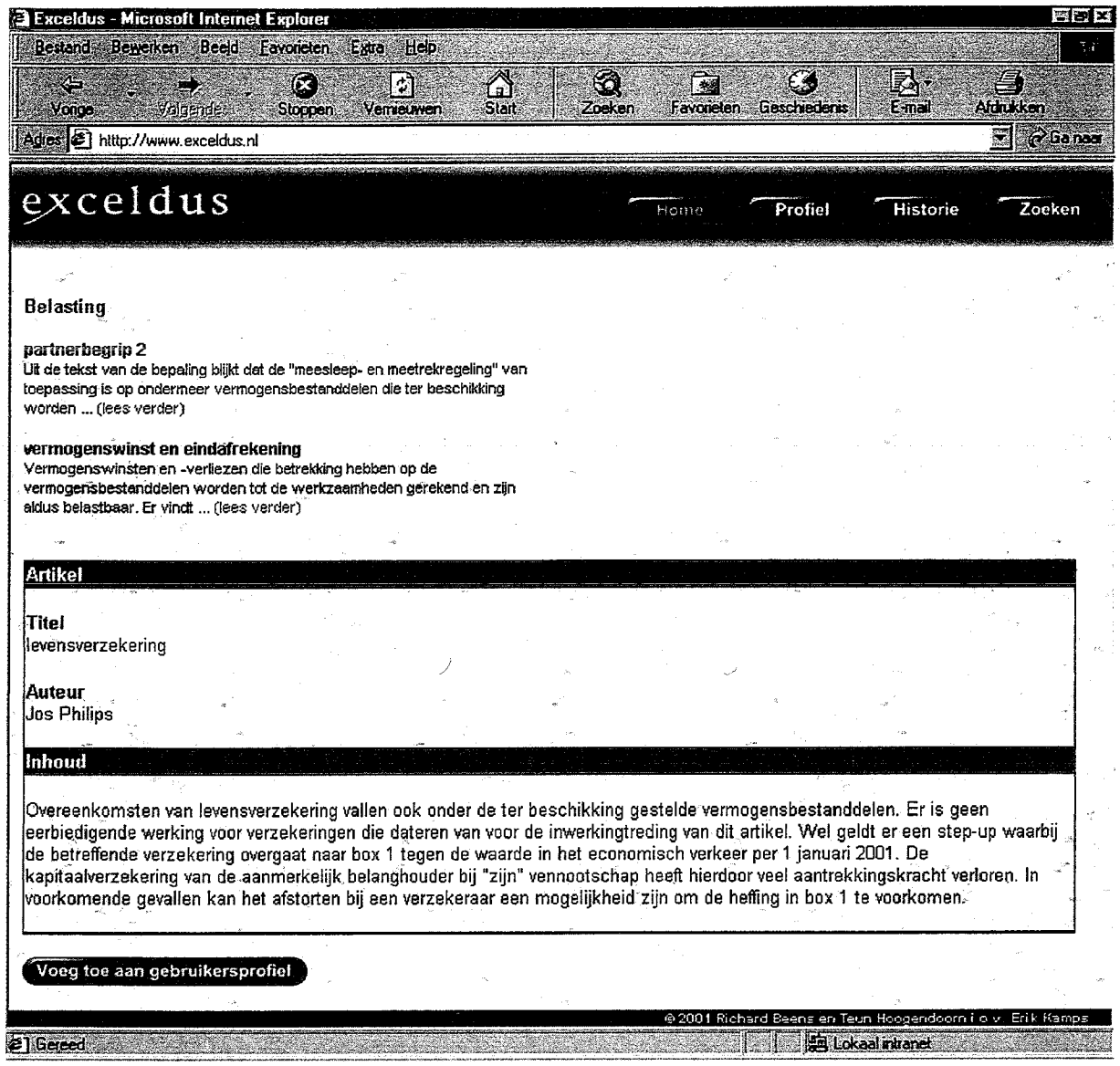
B.2 Homepage

Als er behoefte is opgegeven in het gebruikersprofiel toont de homepage links naar concepten waaraan behoefte is. Deze links worden bepaald aan de hand van het MRL-algoritme, zie paragraaf 5.4.3. Figuur 29 geeft de homepage weer als het gebruikersprofiel is ingevuld zoals in figuur 28.



Figuur 29: De homepage

Op de homepage worden de drie concepten, "Juridisch", "Automatisering" en "Belasting", getoond waaraan initieel behoefte is opgegeven. Het concept waar nog de meeste behoeftevervulling plaats kan vinden wordt als eerste getoond. Dit is het concept met de kleinste BVp-waarde, oftewel het concept "Juridisch", zie figuur 29. Onder ieder concept zijn de drie meest relevante links opgenomen naar concepten die deel uitmaken van het getoonde concept. Het aantal te vertonen links wordt bepaald door een regel in het AM, zie bijlage C. De links die getoond zijn in figuur 29 hebben drie verschillende kleuren. Deze kleuren representeren goede (vet gedrukt blauw), neutrale (blauw) en slechte links (donker blauw) om te volgen.
Via de homepage kunnen, door op de links te klikken, concepten opgevraagd worden. Een voorbeeld van een opgevraagd concept is het artikel weergegeven in figuur 30. Een artikel vertoont aan de bovenzijde maximaal drie relevante links naar concepten die deel uit maken van hetzelfde concept als het getoonde artikel. De fragmenten "Titel", "Auteur" en "Inhoud" zijn getoond. De knop "Voeg toe aan gebruikersprofiel" kent initiële behoefte aan de ouders van het getoonde artikel toe. Op deze manier wordt de initiële behoefte bottum-up gerealiseerd i.t.t. het gebruikersprofiel waar de initiële behoefte top-down wordt gerealiseerd.



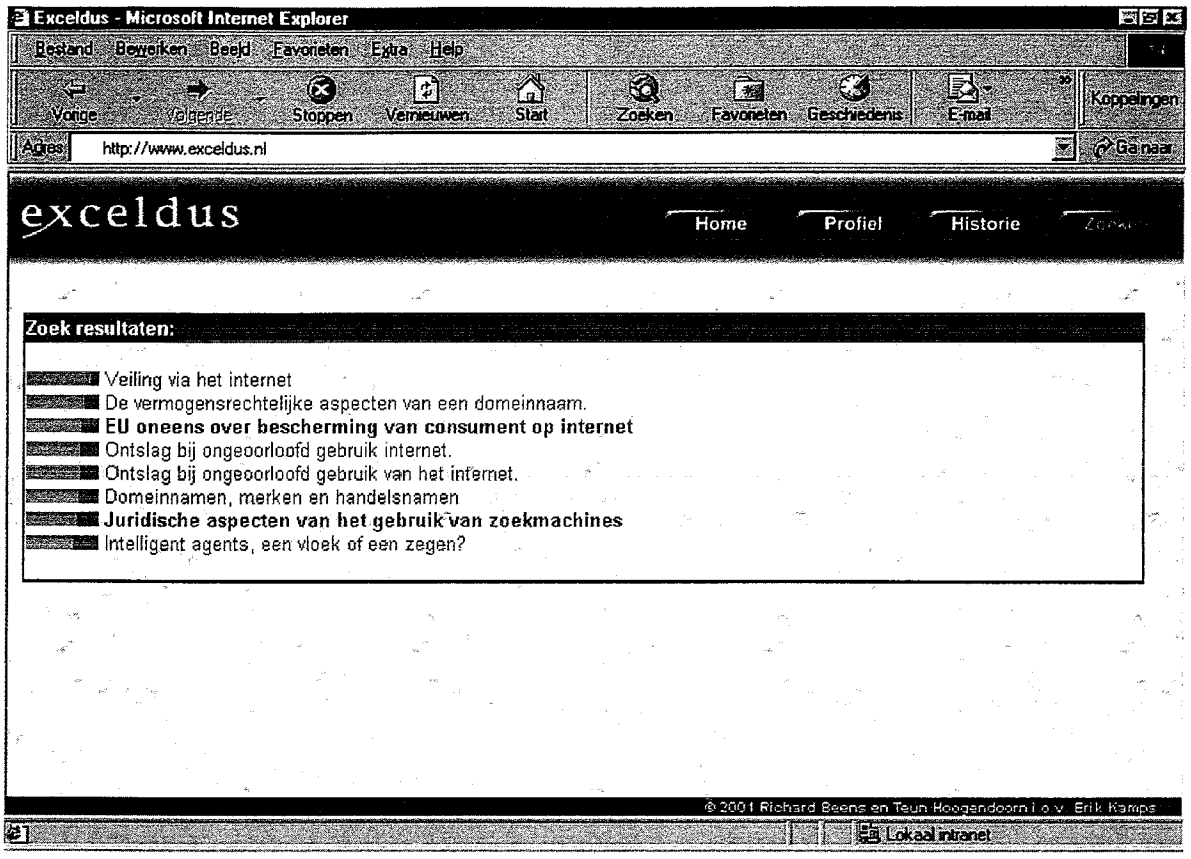
Figuur 30: Een artikel

B.3 Historie

Het MRL-algoritme levert steeds de meest relevante links op waarbij een gelezen artikel als minder relevant wordt beschouwd. Om gelezen artikelen opnieuw te kunnen raadplagen is de historiefunctionaliteit opgenomen. De historiefunctionaliteit geeft een overzicht van alle door de gebruiker gelezen artikelen.

B.4 zoeken

Het zoeken van een artikel d.m.v. een sleutel in de gehele inhoud van de website wordt door de zoekfunctionaliteit gerealiseerd. Er wordt dus ook gezocht in categorieën concepten waaraan geen behoefte is opgegeven. Een voorbeeld van een zoekresultaat is weergegeven in figuur 31. De balken aan de linkerzijde van het resultaat geven met een groene kleur de relevantie van de gezochte sleutel weer. Het zoekresultaat is op deze relevantie gesorteerd en niet op de behoefte, omdat er gezocht is op een sleutel en niet op behoefte. De links die in het zoekresultaat zijn opgenomen worden wel als goed, neutraal of slecht gekenmerkt waardoor goede links duidelijker opvallen in het zoekresultaat.



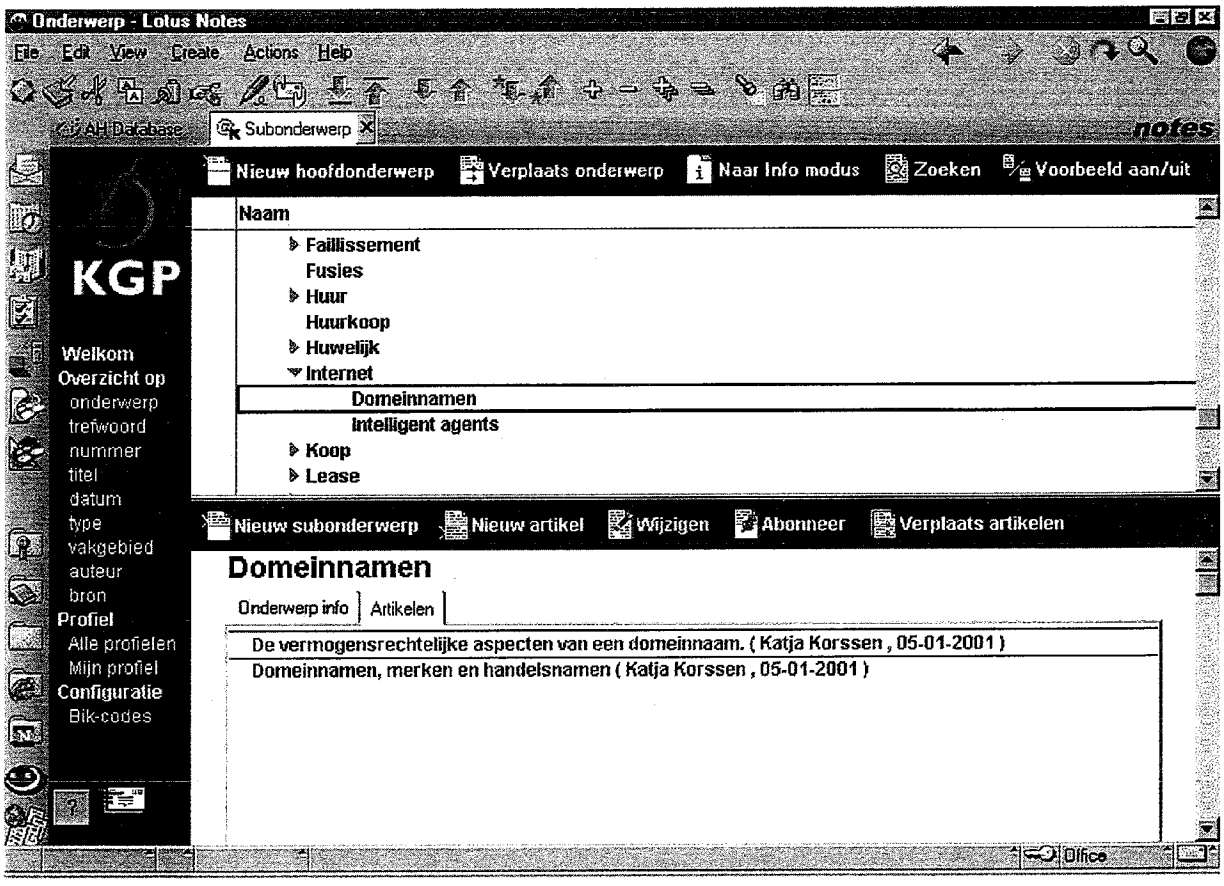
Figuur 31: Zoekresultaat met sleutel internet

Bijlage C: Configuratie ADIRA

In deze bijlage wordt de configuratie van ADIRA besproken. De configuratie van de AHDb wordt in termen van het Domain Model, User Model en Adaptation Model besproken in paragrafen C.1, C.2 en C.3. Algemene configuratie instellingen worden besproken in paragraaf C.4.

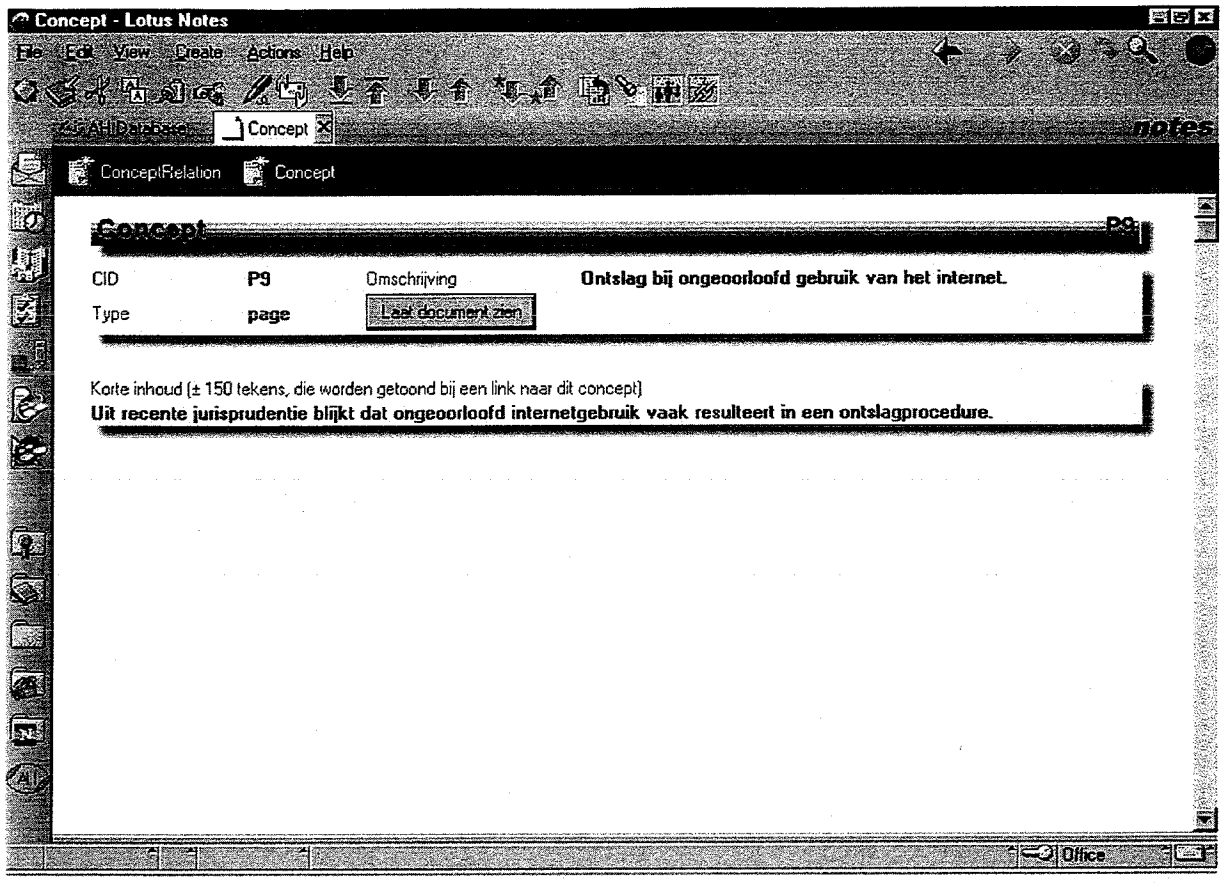
C.1 Domain Model

Het DM bevat een concepthiërarchie van de data uit de KnowledgeDb. Een voorbeeld van de datastructurering in de KnowledgeDb is weergegeven in figuur 32.



Figuur 32: Datastructuur KnowledgeDb

Het artikel "De vermogensrechtelijke..." is ingedeeld onder hoofdonderwerp "Juridisch" (niet in de figuur te zien) en vervolgens onder de subonderwerpen "Internet" en "Domeinnamen".
Het DM dat ADIRA intern gebruikt betreft een conversie van de datastructuur van de KnowledgeDb. Het DM wordt automatisch gecreëerd in de AHDb, waarbij conceptrelaties van type partof gegenereerd worden aan de hand van de datastructuur uit de KnowledgeDb.
De concepten en conceptrelaties zijn handmatig wijzigbaar in de AHDb. Door wijzigingen is het mogelijk een totaal andere concepthiërarchie te creëren dan de datastructuur in de KnowledgeDb.
In figuur 33 wordt een voorbeeld gegeven van een concept in de AHDb gegeven.



Figuur 33: Voorbeeld van een concept in de AHDb

C.2 User Model

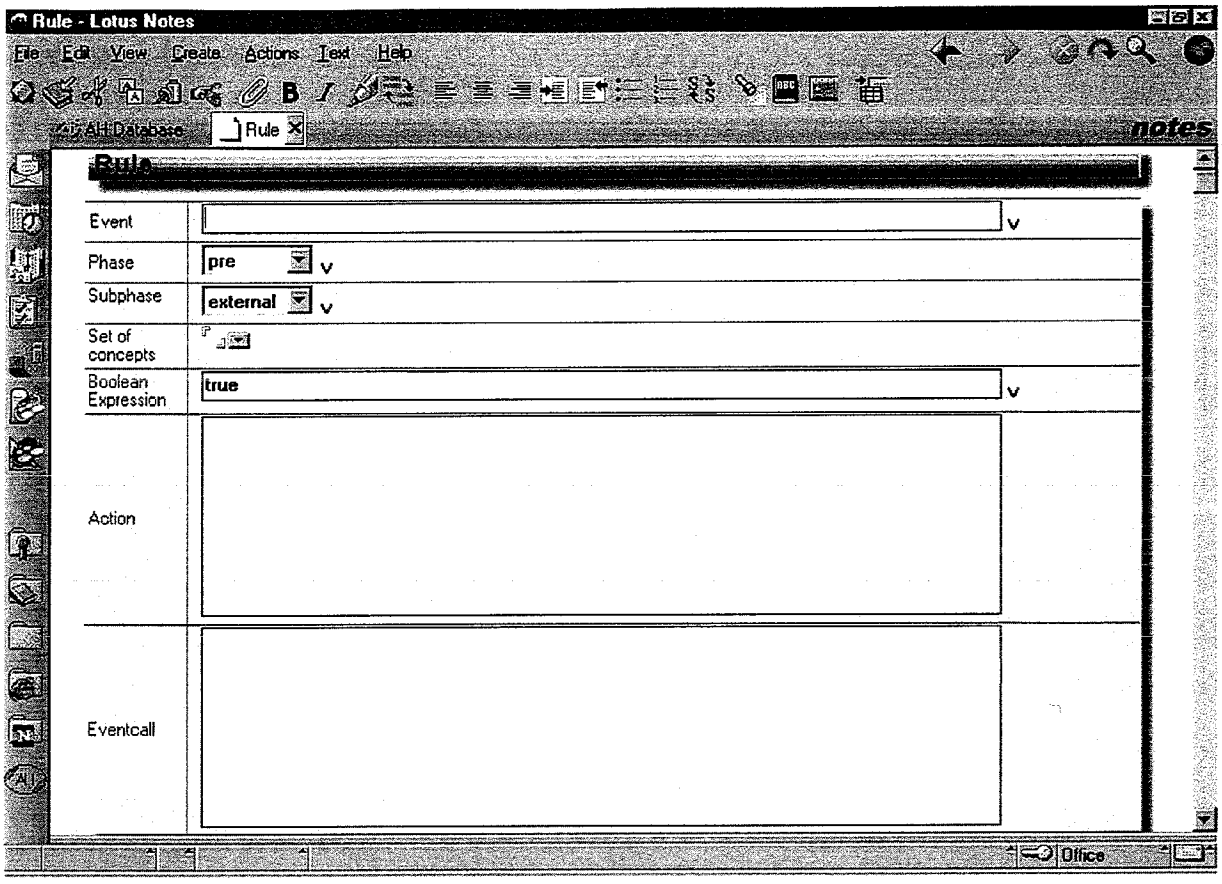
Een gebruiker krijgt toegang tot ADIRA als er een User document in de AHDb is aangemaakt. De sleutel voor de gebruiker betreft de Lotus Notes gebruikersnaam. Deze naam wordt verkregen door de gebruiker te registreren in het "Address Book" van de Domino Server. De Lotus Notes gebruikersnaam heeft de volgende structuur.

```
Voornaam Achternaam/Organistatie
```

Voorbeelden van gebruikersnamen zijn Richard Beens/Exceldus en Teun Hoogendoorn/Exceldus.

C.3 Adaptation Model

De adaptiviteit van ADIRA wordt door het definiëren van regels geconfigureerd. De regels worden ingevoerd aan de hand van het formulier "Rule" dat weergegeven wordt door figuur 34. De velden "Event", "Phase", "Subphase" en "Boolean Expression" dienen verplicht ingevuld te worden voor iedere regel.



Figuur 34: Het formulier Rule

C.3.1 Standaard regels

Voor het configureren van de adaptatie zijn standaard een aantal regels opgenomen in het AM. Deze regels zijn naar eigen inzicht wijzigbaar, maar er dient rekening mee gehouden te worden dat dit grote veranderingen in de functionaliteit teweeg kan brengen.

Het toevoegen van de fragmenten van concept C.
Het opvragen van een concept C resulteert in het toevoegen van alle fragmenten behorende bij C.

Event	Phs	Subphs	Soc	Boolean Expression	Action	Eventcall
access	pre	external	all	true	P.addCR("partof", {C})	

Het toevoegen van de meest relevante links.
Aan een abstract en een pagina concept worden drie relevante links toegevoegd, hetgeen wordt uitgedrukt in de eerste twee regels. De overige regels, met het event findCandidate, zoeken vanaf het opgevraagde concept naar kandidaat concepten voor het MRL-algoritme.
Indien een abstract concept C wordt opgevraagd dan zijn alle kinderen van C, waaraan initieel behoefte is, kandidaat.
Indien een pagina concept C wordt opgevraagd dan zijn alle ouders van C, waarvoor geldt dat er initieel behoefte aan is en de behoefte nog niet voor meer dan 90% is vervuld, kandidaat. Als voor alle ouders geldt dat de behoeftevoldoening groter is dan 90% dan wordt de root (C1) als kandidaat concept genomen. Merk op dat ieder concept één ouder heeft waardoor altijd één kandidaat concept wordt gevonden.

Event	Phs	Subphs	Soc	Boolean Expression	Action	Eventcall
access	pre	external	all	C.Type == "page"	P.addCR("mrlink",3,U,C)	forall p in C.Parents do findCandidate(p, true) od
access	pre	external	all	C.Type == "abstract"	P.addCR("mrlink",3,U,C)	forall c in C.Children do findCandidate(c, false) od
findCandidate	pre	internal	all	val&!UC.BI		forall p in C.Parents do findCandidate(p, val) od

findCandidate	pre	internal	all	val&UC.BI&UC.BVp<90	P.addMRCandidate(C)	
findCandidate	pre	internal	all	val&UC.BI&UC.BVp>=90		forall p in C.Parents do findCandidate(p, val) od
findCandidate	pre	internal	all	!val&UC.BI	P.addMRCandidate(C)	
findCandidate	pre	internal	C1	val&UC.BI&UC.BVp>=90	P.addMRCandidate(C)	

Het toekennen van een status aan een link

In de pre-fase worden alle conceptrelaties (partof en mrlink) aan de Presentation Specification toegevoegd. De conceptrelaties van type link krijgen een status toegekend. Deze status wordt gebruikt om de link een andere kleur te geven als de link als goed, neutraal of slecht geïdentificeerd is.
Een link wordt als een slechte link geïdentificeerd als het concept, waarnaar de link verwijst, al gelezen is.
Een link wordt als een neutrale link geïdentificeerd als het bijbehorende concept C nog niet gelezen is en aan de ouders van C geen initiële behoefte bestaat. Een link wordt als een goede link geïdentificeerd als het bijbehorende concept C nog niet gelezen is en aan de ouders van C initiële behoefte bestaat.

Event	Phs	Subphs	Soc	Boolean Expression	Action	Eventcall
access	gen	external	all	true		forall p in P.Links do setLinkStatus(l, l) od
setLinkStatus	gen	internal	all	C.Type=="abstract"&!UC.BI	P.linkStatus(val,"neutral")	
setLinkStatus	gen	internal	all	C.Type=="abstract"&UC.BI	P.linkStatus(val,"good")	
setLinkStatus	gen	internal	all	C.Type=="page"&UC.Read==0		forall c in C.Parents do setLinkStatus(p, val) od
setLinkStatus	gen	internal	all	C.Type=="page"&UC.Read>=0	P.linkStatus(val,"bad")	

Updaten van het aantal gelezen concepten

Als een concept opgevraagd is, wordt het aantal keer dat het concept gelezen is verhoogd met 1. Dit geldt ook voor alle ouders van het opgevraagde concept.

Event	Phs	Subphs	Soc	Boolean Expression	Action	Eventcall
access	post	external	all	true	UC.Read = UC.Read + 1	forall p in C.Parents do updateRead(p) od
updateRead	post	internal	all	true	UC.Read = UC.Read + 1	forall p in C.Parents do updateRead(p) od

Updaten van de behoeftetoestand: Afgeleide behoefte

Als een niet gelezen pagina opgevraagd is, wordt de behoeftetoestand van die pagina en zijn ouders bijgewerkt. De bepaling van de behoefteverandering zoals weergegeven in de eventcall van de twee regels staat beschreven in paragraaf 4.3.1.

Event	Phs	Subphs	Soc	Boolean Expression	Action	Eventcall
access	post	external	all	UC.Read==0&C.Type="page"	UC.BVa=1	forall p in C.Parents do updateBVa(p, C.BB(p)) od
updateBVa	post	internal	all	true	UC.Read = UC.Read + 1	forall p in C.Parents do updateBVa(p, (val/C.MB) * C.BB(p)) od

Opvragen van het gebruikersprofiel

Het gebruikersprofiel waarin de initiële behoefte kan worden vastgelegd wordt aan de hand van de volgende regels gegenereerd. Door deze regels kan in het gebruikersprofiel initiële behoefte opgegeven worden aan de kinderen van C en de kinderen van de kinderen van C.

Event	Phs	Subphs	Soc	Boolean Expression	Action	Eventcall
accessProfile	pre	external	all	C.Type != "page"		forall c in C.Children do addTopics(c, C) od
addTopics	pre	internal	all	C.Type != "page"	P.addTopic(C, val,UC.BI)	forall c in C.Children do addSubTopics(c, C) od
addSubTopics	pre	internal	all	C.Type != "page"	P.addTopic(C, val,UC.BI)	

Updaten van de behoeftetoestand: Initiële behoefte

Het updaten van de initiële behoefte kan plaatsvinden doordat op de knop "Voeg toe aan gebruikersprofiel" is geklikt, dat resulteert in de triggering van de eerste regel. Als de initiële behoefte is gewijzigd in het gebruikersprofiel (bevat alleen abstracte concepten), resulteert dit in de triggering van de tweede en de derde regel. Als aan een concept C initiële behoefte wordt toegekend, betekent dit dat aan alle ouders van C ook initiële behoefte is (vierde regel). Als de initiële behoefte aan een concept C wordt opgezegd, betekent dit dat aan alle kinderen van C geen behoefte meer is (vijfde regel).

Event	Phs	Subphs	Soc	Boolean Expression	Action	Eventcall
updateBI	pre	external	all	C.Type == "page"		forall p in C.Parents do processBIp(p, val) od
updateBI	pre	external	all	val&!UC.BI&C.Type!="page"	UC.BI=true	forall p in C.Parents do processBIp(p, val) od
updateBI	pre	external	all	!val&UC.BI&C.Type!="page"	UC.BI=false	forall c in C.Children do processBIc(c, val) od
processBIp	pre	internal	all	val&!UC.BI&C.Type!="page"	UC.BI=true	forall p in C.Parents do processBIp(p, val) od
processBIc	pre	internal	all	!val&UC.BI&C.Type!="page"	UC.BI=false	forall c in C.Children do processBIc(c, val) od

C.3.2 Door de auteur gedefinieerde regels

Naast de standaard gedefinieerde regels in het AM kan de auteur naar wens regels toevoegen.

Toevoegen van een reclame banner
Het toevoegen van een reclame banner is niets anders dan een conceptrelatie met de banner toevoegen aan de Presentation Specification.
Het toevoegen van reclame banner "b1", als de omschrijving van een ouder van het opgevraagde concept "Juridisch" heeft, is gedefinieerd door de volgende twee regels.

Event	Phs	Subphs	Soc	Boolean Expression	Action	Eventcall
access	pre	external	all	true		forall p in C.Parents do addBanner(p) od
addBanner	pre	internal	all	C.Description=="Juridisch"	P.addCR("partof",{"b1"})	

Het toevoegen van reclame banner "b1" aan de concepten "p1", "p12" en "p23" wordt gerealiseerd door de volgende regel.

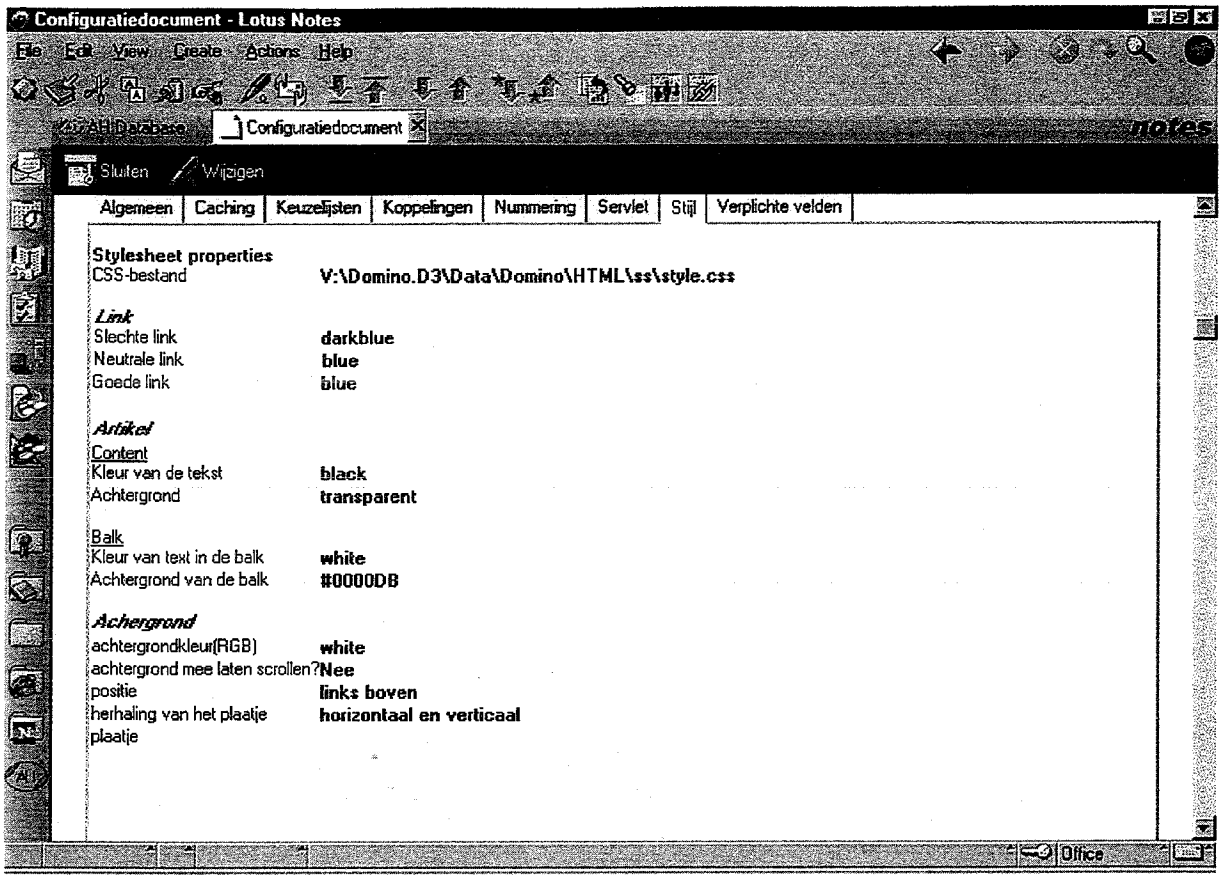
Event	Phs	Subphs	Soc	Boolean Expression	Action	Eventcall
access	pre	internal	p1, p12, p23	true	P.addCR("partof",{"b1"})	

Verwijderen van fragmenten
Sommige fragmenten mogen niet zichtbaar zijn voor bepaalde gebruikers. Het niet zichtbaar maken van een fragment komt overeen met het verwijderen van de conceptrelatie met het fragment. Een voorbeeld is het verwijderen van het fragment "detailedContent" als de gebruiker geen recht heeft om de gedetailleerde inhoud van een concept te bekijken.

Event	Phs	Subphs	Soc	Boolean Expression	Action	Eventcall
access	pre	external	all	U.Role == "simpleReader"	P.removeCR("partof", {"detailedContent"})	

C.4 Algemene instellingen

Algemene instellingen kunnen geconfigureerd worden in het configuratiedocument. De instellingen betreffen koppelingen met databases, servletinstellingen, stijlinstellingen en nog een aantal andere instellingen, zoals welke velden per formulier verplicht zijn. Het configuratiedocument is ingedeeld in categorieën. Figuur 35 geeft het configuratie document met de stijlinstellingen geselecteerd weer.



Figuur 35: Configuratiedocument

Stijlinstellingen

De HTML pagina die gecreëerd wordt (d.m.v. de conversie van XML naar HTML) maakt gebruik van Cascading Style Sheets (CSS) om aan de verschillende HTML attributen een stijl toe te kennen. De stijlinstellingen in het configuratiedocument geven de auteur de mogelijkheid de stijl in zijn algemeenheid te wijzigen. Een voorbeeld van een stijlinstelling is het kleuren van goede, neutrale en slechte links. De links krijgen door uitvoering van regels in het AM een status en worden door gebruik te maken van CSS gekleurd. Er dient opgemerkt te worden dat een goede link vetgedrukt wordt weergegeven (niet te zien in figuur 35).

Bijlage D: Performance

Deze bijlage bespreekt het performance probleem dat is opgetreden tijdens de implementatie van ADIRA. Het probleem betreft de implementatie van de DataHandler die de koppeling tussen de applicatie en het DBMS Lotus Notes bewerkstelligt (zie paragraaf 6.2.1). Paragraaf D.1 bespreekt hoe de DataHandler geïmplementeerd zou moeten worden volgens Lotus Notes. Paragraaf D.2 behandelt de alternatieven. De keuze voor een alternatief brengt een wijziging van het ontwerp met zich mee, die besproken wordt in paragraaf D.3.

D.1 Implementatie volgens Lotus Notes documentatie

Volgens de Lotus Notes documentatie zou een request uit een Lotus Notes database als volgt geïmplementeerd moeten worden (in pseudo-code).

```
public class DataHandler
|[
 getValue(): String
 |[
  NotesThread.sinitThread;
  session := new NotesSession;
  getValue := "value";
  NotesThread.stermThread;
 ]|
]|
```

Dit houdt in dat per request een sessie met de server geopend moet worden. Dit resulteert in een responsetijd van 250 ms per request. De gemiddelde zoektijd van een harde schijf is onder de 10 ms. Onder aanname dat dit de bottleneck is, zou het mogelijk moeten zijn om een request te doen aan de database in ongeveer 10 ms. Dit betekent dat deze implementatie van een request 25 keer trager is dan mogelijk is, hetgeen onacceptabel is.

D.2 Alternatieven

Er zijn twee andere mogelijkheden om de DataHandler zodanig te implementeren dat een sessie niet elke keer opnieuw geopend behoeft te worden.

1. Gebruikmaking van het NSCO pakket, waardoor een lokale Java applicatie een CORBA gebaseerde remote-call tot een andere Domino Server kan uitvoeren.
2. Implementatie van de DataHandler als continue proces.

Beide mogelijkheden zijn getest en worden besproken in de volgende paragrafen.

D.2.1 Alternatief 1: NSCO pakket

Een request met behulp van het NSCO pakket wordt als volgt geïmplementeerd. Bij het initialiseren van de klasse DataHandler wordt er eenmalig een sessie geopend met de Domino Server. Bij een request d.m.v. de methode getValue() kan een waarde worden opgehaald uit een Lotus Notes database.

```
public class DataHandler
|[
 public DataHandler
 |[
    session = NotesFactory.createSession("host", "user", "pwd");
 ]|

 getValue(): String
 |[
   getValue := "value";
 ]|
]|
```

Bij het testen van deze implementatie is gebleken dat de responsetijd ongeveer 300 ms per request is.

D.2.2 Alternatief 2: Datahandler als continue proces

De DataHandler kan als continue proces worden geïmplementeerd door deze te implementeren in een Thread, waarbij de sessie continu geopend blijft. De volgende stappen worden uitgevoerd bij een request:

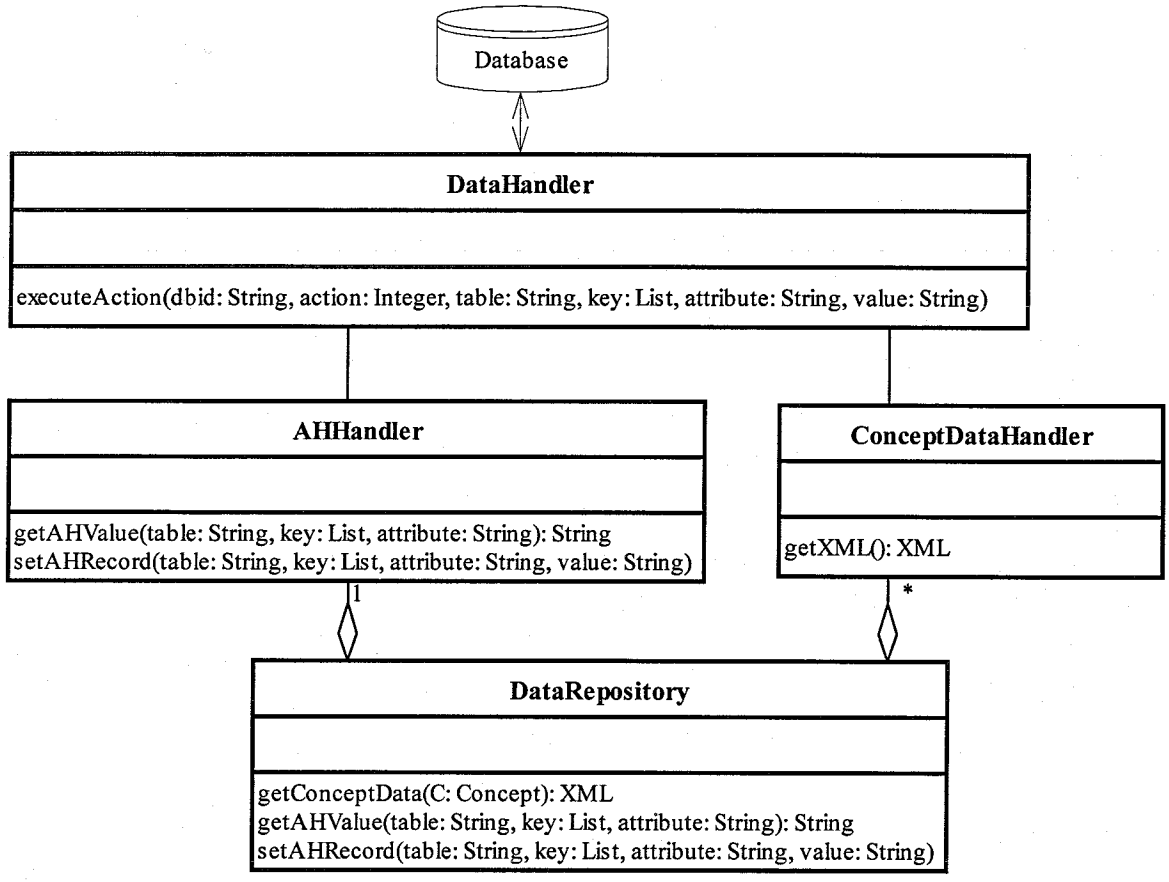
1. Methode fileAction wordt aangeroepen.
2. Methode fileAction voert een V-operatie uit op seinpaal s1.
3. Methode run voert een P-operatie uit op seinpaal s1.
4. Methode run voert de request uit.
5. Methode run voert een V-operatie uit op seinpaal s2.
6. Methode fileAction eindigt met een P-operatie op seinpaal s2.

```
public class DataHandler extends Thread
|[
 private s1: BinarySemaphore;
 private s2: BinarySemaphore;
 private dbKey : DBKey;
 private db: NotesDatabase;
 private view: NotesView;
 private session: NotesSession;
 private action: Integer;
 private value: String;
 const GV_BYKEY = 1;

 run()
 |[
    session := new NotesSession; {eenmalig een sessie openen)
    do (true) ->
    P(s1);
    if action = GV_BYKEY -> GetValue() fi;
    V(s2);
    od;
 ]|

 synchronized fileAction(dbKey: DBKey, table: String, key: List, attribute: String) : String
 |[ {deze procedure moet gesynchroniseerd worden, m.a.w. alle aanroepen van deze procedure
    worden sequentieel uitgevoerd. Anders treedt er interferentie op bij de variabelen
    dbKey, table, key en attribute}

    this.dbKey := dbKey;
    this.table := table;
    this.key := key;
    this.attribute :=attribute;
    V(s1);
    P(s2);
    fileAction := value;
 ]|

 getValue(table: String, key: List, attribute: String) : String
 |[
    Var doc: NotesDocument;
    db := session.getDatabase(dbKey);
    view := db.getView(table);
    doc := view.getDocumentByKey(key);
    getValue :=  doc.getItemValue(attribute);
 ]|
]|
```

Bij het testen van deze implementatie is gebleken dat de responsetijd ongeveer 10 ms per request is. Er is voor deze implementatie gekozen vanwege de snelle responsetijd.

D.3 Aanpassing aan het ontwerp

Het is nodig om het klassenmodel aan te passen, omdat ervoor gekozen is om de DataHandler als continue proces te implementeren. De klasse DataHandler moet worden aangeroepen d.m.v. één procedure genaamd fileAction, omdat er dan gesynchroniseerd kan worden zodat er geen interferentie optreedt. De ConceptDataHandlers en de AHHandler maken gebruik van de DataHandler. De generalisatie relatie tussen de ConceptDataHandler, AHHandler en de DataHandler wordt voor de implementatie vervangen door een associatierelatie.



Figuur 36: Aangepast ontwerp voor implementatie