TECHNISCHE UNIVERSITEIT EINDHOVEN
Faculteit Wiskunde en Informatica

AFSTUDEERVERSLAG

Dynamische object insluiting in AHA!

door

B.H. de Lange

Afstudeerdocenten: Dr. A.T.M. Aerts
Prof. dr. P.M.E. De Bra
Dr.ir. G.J. Houben

juni 2003

Distributieblad:

Ontvangers	Locatie	aantal
TU/e Eindhoven: Dr. A. Aerts Prof Dr. P. De Bra Dr. ir. G.J. Houben	a.t.m.aerts@tue.nl debra@win.tue.nl g.j.houben@tue.nl Department of Computer Science PO Box 513 NL 5600 MB Eindhoven	3
Student Ing. B.H. de Lange	barenddelange@hetnet.nl Grote Berg 49a 5611 KJ Eindhoven	1
Datum distributie:	**mei 2003**	

Versie 0.9c 27 mei 2003

Voorwoord

Voor u ligt de scriptie welke mijn afstudeerperiode op de TU/e afsluit. Tevens sluit dit afstudeeronderzoek mijn drie jaar durende verkorte opleiding aan de TU/e af. Na het lezen van deze scriptie zou u een redelijk beeld van mijn afstudeeropdracht gekregen moeten hebben.

De 9 maanden durende afstudeeropdracht is uitgevoerd in de vakgroep Informatie Systemen (IS) van de faculteit Wiskunde en Informatica aan de Technische Universiteit Informatica.

Daar is onder de begeleiding van prof.dr. Paul De Bra en dr. Ad Aerts onderzoek gedaan in het vakgebied Adaptieve Hypermedia Systemen en specifiek binnen het AHA! project.

Een uitgebreid dankwoord vindt u aan het eind van deze scriptie en ik wens u veel plezier met het lezen van deze scriptie.

Eindhoven, mei 2003

Inhoudsopgave

Voorwoord 3

Samenvatting 3

Summary 3

1. Inleiding 3

1.1. Hypertext 3
1.1.1. Hypermedia 3

1.2. Adaptief Hypermedia systeem 3
1.2.1. Toepassing adaptief systeem 3
1.2.2. Adaptatie doelgebied 3
1.2.3. Adaptieve elementen 3

1.3. Opbouw van AHS 3
1.3.1. User Model 3
1.3.2. Domein Model 3
1.3.3. Adaptatie Engine 3
1.3.4. Pagina's en fragmenten 3
1.3.5. Overzicht AHA! systeem 3

1.4. Onderzoeksvragen 3

2. AHA! 3

2.1. Inleiding 3

2.2. Overzicht opbouw in AHA! 1.96 3
2.2.1. Functie overzicht AHA! 1.96 3
2.2.2. Architectuur AHA! 1.96 3

2.3. AHA! 3.0 3
2.3.1. Inleiding 3
2.3.2. Wijzigingen in systeemcomponenten 3

3. Oplosmethode 3

3.1. Aanleiding 3

3.2. Fragment 3

3.3. Fragment identificatie en verwerking 3
3.3.1. IFRAME 3
3.3.2. SSI 3
3.3.3. AHA! definitie 3
3.3.4. Object Tag constructie 3

3.4. Keuze van oplosstrategie 3
3.4.1. Conclusie keuze oplosstrategie 3

3.5. Fragment opslag door DM/content splitsing 3

4. Theoretische consequenties 3

4.1. Inleiding 3

4.2. Meerdimensionale object structuur 3

4.3. Terminatie bij recursieve objectstructuren 3
4.3.1. Restrictie boomdiepte 3
4.3.2. Monotone UM mutaties 3
4.3.3. Patroon herkenning 3
4.3.4. conclusie terminatie 3

4.4. Verminderd overzicht AHA! applicatie 3

4.5. AHA! MEETS SMIL 3
4.5.1. Conclusie AHA MEETS SMIL 3
4.6. Hergebruik fragmenten in AHS 3
4.7. Autonoom gedrag van een fragment 3
4.8. Modulaire opbouw AHA! systeem 3
5. Technische consequenties *3*
5.1. Inleiding 3
5.2. Performance problemen 3
5.3. Verandering in de concept definitie (domein model) 3
5.4. Verandering auteurstools 3
5.5. Verandering in de AE 3
5.5.1. Inleiding 3
6. Implementatie techniek *3*
6.1. Inleiding 3
6.2. Object structuur herkenning en verwerking 3
6.3. Terminatie garanderen 3
6.4. Correct parallellisme garanderen 3
6.4.1. Inconsistentie 3
6.4.2. Livelock 3
6.4.3. Oplossing voor correct parallellisme in AHA! 3
7. Conclusie en aanbevelingen *3*
7.1. Conclusies 3
7.2. Aanbevelingen 3
7.2.1. Aanpassen editors 3
7.2.2. Versiebeheer systeem 3
7.2.3. Recursieve structuren afbreken en visualiseren 3
7.2.4. Fragment identificatie extern opslaan 3
7.2.5. Andere XML toepassingen exploreren 3
8. Verklarende woordenlijst *3*
9. Onderzoeksbronnen *3*
9.1. Referenties: 3
9.2. Afstudeerverslagen 3
9.3. Stageverslagen 3
9.4. Literatuur 3
9.5. Lijst van figuren 3
10. Dankwoord *3*
bijlage I software stroomschema's *3*
Bijlage II papers *3*

Samenvatting

Iedere twee jaar verdubbelt de hoeveelheid informatie in de wereld. Mensen zijn in staat deze informatie te raadplegen bijvoorbeeld via het Internet. Om een gebruiker te ondersteunen bij het opnemen en raadplegen van deze informatie zou de informatie op een gebruikersvriendelijke manier gepresenteerd moeten worden. Gebruiksvriendelijk betekent in deze context, dat sommige stukken informatie bewust niet worden getoond. Andere stukken informatie bewust beter onder de aandacht gebracht moeten worden. De presentatie van informatie aanpassen is dat wat het AHA! systeem doet. Dit wordt een Adaptief Hypermedia Systeem genoemd.

Dit verslag beschrijft het onderzoek en de aanpassingen aan een klein deel van dit Adaptief Hypermedia Systeem (AHA!). Er is primair gekeken naar fragment verwerking binnen het AHA! systeem. Een fragment is een uniek identificeerbaar stuk informatie welke op meerdere plaatsen in een informatiesysteem voor kan komen. In AHA! 1.96 werden fragmenten op een vrij primitieve manier gedefinieerd en verwerkt. De doelstelling was om deze primitieve manier van fragment verwerking te vervangen door een geavanceerde fragmentverwerkingsmethode. Deze methode zorgt voor veel nieuwe mogelijkheden binnen adaptieve toepassingen, zoals meerdimensionale fragment structuren, autonoom fragment gedrag en versterkte structurering van informatie, in een informatiesysteem. Deze effecten zijn beschreven in deze thesis. De nieuwe structuur heeft ook bijwerkingen die het systeem instabiel kunnen maken. Deze bijwerkingen zijn te detecteren en op te vangen door tijdig in te grijpen.

De scriptie wordt afgesloten met conclusies en aanbevelingen waar toekomstig werk uit te distilleren valt.

Summary

Every year the total amount of information in the world doubles. Humans are able to access this information, for example using the Internet. To support a user with retrieving and understanding this amount of information, the information must be pre-processed. Some part of the information must be skipped (because a user already knows it) other information must be highlighted to get it more under the attention. Adapting the presentation to a user is the field of Adaptive Hypermedia. AHA! Is a computer system that changes the presentation for a specific user.

This thesis describes the research development and changes of a small part in a Adaptive Hypermedia system called AHA!. Within this Adaptive Hypermedia System we mainly focussed on processing fragments. A fragment is in our definition a small identifiable piece of information, used on one ore more page(s). The current version of AHA! (1.96) uses a very primitive way of handling these fragments. Our aim was to use a more flexible and advanced way of handling these fragments. This new way of handling fragments comes with great advantages like: more dimensional fragment structure, autonomic behaviour of a fragment and a richer domain model because of the new strong structure. All these new possibilities are written down in this thesis. The new structure with all its new possible constructions, add also some dangerous situations which can damage the system (i.e. server down), these risks are also documented and solutions are proposed.

The thesis ends with a conclusion and future work chapter which can be used to create new assignments.

1. Inleiding

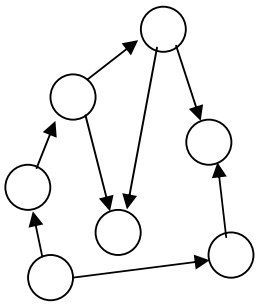
In de huidige informatiemaatschappij wordt er meer dan ooit tevoren informatie aan de mens aangeboden. Tussen 1800 en 1900 verdubbelde de hoeveelheid informatie, dit gebeurde weer tussen 1900 en 1950, vervolgens tussen 1950 en 1960 en tussen 1960 en 1966. Nu verdubbelt de hoeveelheid informatie iedere 2 jaar. Op het Internet zelfs iedere 8 maanden. [1]. De hoeveelheid kennis zal alleen maar toenemen. Het wordt voor een mens een probleem om al deze kennis te benaderen, raadplegen en op te nemen. Een algemene en triviale oplossing voor dit probleem is, per gebruiker informatie op maat te maken. Informatie aanpassen per gebruiker kan niet alleen een beperking aan informatie opleveren, het kan er ook voor zorgen dat de informatie geschikter is voor de gebruiker. Systemen die de informatie aanpassen aan de gebruiker, zijn adaptieve systemen.

De vakgroep Informatiesystemen van de faculteit Wiskunde & Informatie verricht onderzoek naar Adaptieve Hypermedia Systemen. Het AHA! (Adaptive Hypermedia Architecture) systeem wordt door de vakgroep Informatiesystemen gebouwd en onderhouden. Binnen deze vakgroep heeft mijn afstudeeropdracht zich afgespeeld. Onder de begeleiding van Prof. Dr. Paul De Bra en Dr. Ad Aerts heb ik onderzoek verricht naar een fragment objectstructuur welke deels oude functionaliteit verving en deels nieuwe functionaliteit toevoegde aan het huidige AHA! 1.96 systeem. Bij de start van het afstuderen is als uitgangspositie de in augustus uitgekomen AHA! versie 1.96. Parallel werd door N. Stach aan versie 2.0 gewerkt. Onze functionaliteit wordt pas in versie 3.0. toegevoegd aan het AHA! systeem. De probleemstelling wordt verderop in dit hoofdstuk uitvoerig uiteengezet.

Dit verslag is opgedeeld in een zevental hoofdstukken. Dit hoofdstuk beschrijft de componenten binnen een Adaptief Hypermedia Systeem (AHS) en de onderzoeksvragen. Het volgende hoofdstuk, hoofdstuk twee, “AHA!”, beschrijft hoe het AHA! systeem functioneel werkt en wat de verschillen tussen AHA! 1.96 en het door ons gemaakte AHA! 3.0 versie zijn. Hoofdstuk drie beschrijft de verschillende oplosstrategieën en de uiteindelijk gekozen oplossing voor de in hoofdstuk één beschreven onderzoeksvragen. De theoretische gevolgen van de nieuwe functionaliteit worden in hoofdstuk vier beschreven. De technische consequenties worden in hoofdstuk vijf toegelicht. Hoofdstuk zes handelt over de implementatietechnieken die de nieuwe structuur mogelijk gemaakt hebben. De conclusies en aanbevelingen van het onderzoek staan in het laatste hoofdstuk, hoofstuk zeven. De rest van dit hoofdstuk beschrijft de werking en de verschillende componenten binnen een adaptief systeem en de onderzoeksvraag.

1.1. Hypertext

Gedrukte informatie (zoals een boek) is standaard lineair opgezet. Je leest een boek, pagina voor pagina (van voor naar achter). Het is wel mogelijk om te springen naar andere pagina’s maar dit gebeurt aan de hand van een index of inhoudsopgave. Deze manier van informatie opslag is niet erg flexibel, een gebruiker zou soms direct van een naar een andere locatie willen “springen”. Bij een leesboek wordt dit normaal gesproken niet gedaan. Bij een woordenboek is dit wel een gebruikelijke techniek. Er wordt een woord opgezocht, in de uitleg van dit woord staat een nieuw onbekend woord. Dit woord wordt vervolgens opgezocht.
Hypertext is een niet lineaire manier van gegevens ordening. Hypertext bestaat uit verschillende stukken informatie, nodes genaamd die verbonden zijn met elkaar middels links [SHNEIDERMAN1989]. Links worden bijvoorbeeld op het Internet veelvuldig gebruikt. In het voorbeeld van het woordenboek; in de uitleg van een woord staan link naar de woorden welke in de uitleg staan.



Figuur 1 Hypertext

1.1.1. Hypermedia

Hypermedia is een combinatie van hypertext en multimedia. Er kunnen in de nodes ook multimedia elementen voorkomen. Multimedia elementen zijn bijvoorbeeld video fragmenten.

1.2. Adaptief Hypermedia systeem

Een adaptief systeem is een systeem welke de presentatie aanpast aan de gebruiker. Een definitie is gegeven door Brusilovsky [BRUSILOVSKY1996] (vrij vertaald):
"Adaptieve Hypermedia systemen zijn alle hypertext en hypermedia systemen welke eigenschappen van gebruikers beschouwen en opslaan, teneinde deze eigenschappen te gebruiken bij de diverse presentatie aspecten van het adaptief hypermedia systeem naar de gebruiker. Het systeem moet voldoen aan drie criteria: het moet een hypertext of hypermedia systeem zijn, het heeft een user model, het moet in staat zijn gebruikmakend van het user model de hypermedia inhoud aan te passen."

Nu een definitie gegeven is, reizen er enkele vragen op:

- Waar is een adaptief systeem in te zetten?
- Voor welk doelgebied adaptief presenteren?
- Wat kan er adaptief gepresenteerd worden?
- Uit welke componenten bestaat een AHS (Adaptief Hypermedia Systeem)?

1.2.1. Toepassing adaptief systeem

Indien een informatie systeem verschillende soorten bezoekers ontvangt en voor elke bezoeker een andere presentatie gewenst is, zijn adaptieve systemen uiterst nuttig. Voorbeelden zijn bijvoorbeeld te vinden in educatieve toepassingen. Studenten die zeer snel stof tot zich kunnen nemen zouden eerder complexe informatie moeten ontvangen dan studenten die meer moeite met de stof hebben. Een adaptief systeem kan middels het user model van de gebruiker bijhouden welke informatie een gebruiker al gezien heeft. Een volgende keer dat de gebruiker dezelfde informatie aanvraagt, kan het systeem zijn presentatie aanpassen door bijvoorbeeld een samenvatting te presenteren. Een adaptief systeem kan ook inspelen op de fysieke eigenschappen van een gebruiker. Indien een gebruiker blind is, zou een stuk informatie voorgelezen kunnen worden in plaats van een stuk tekst te tonen. De invoer voor een blind persoon kan een braille invoer zijn. Een voorbeeld adaptieve applicatie die in de praktijk gebruikt wordt, is de adaptieve cursus 2L670 [DEBRA1996].

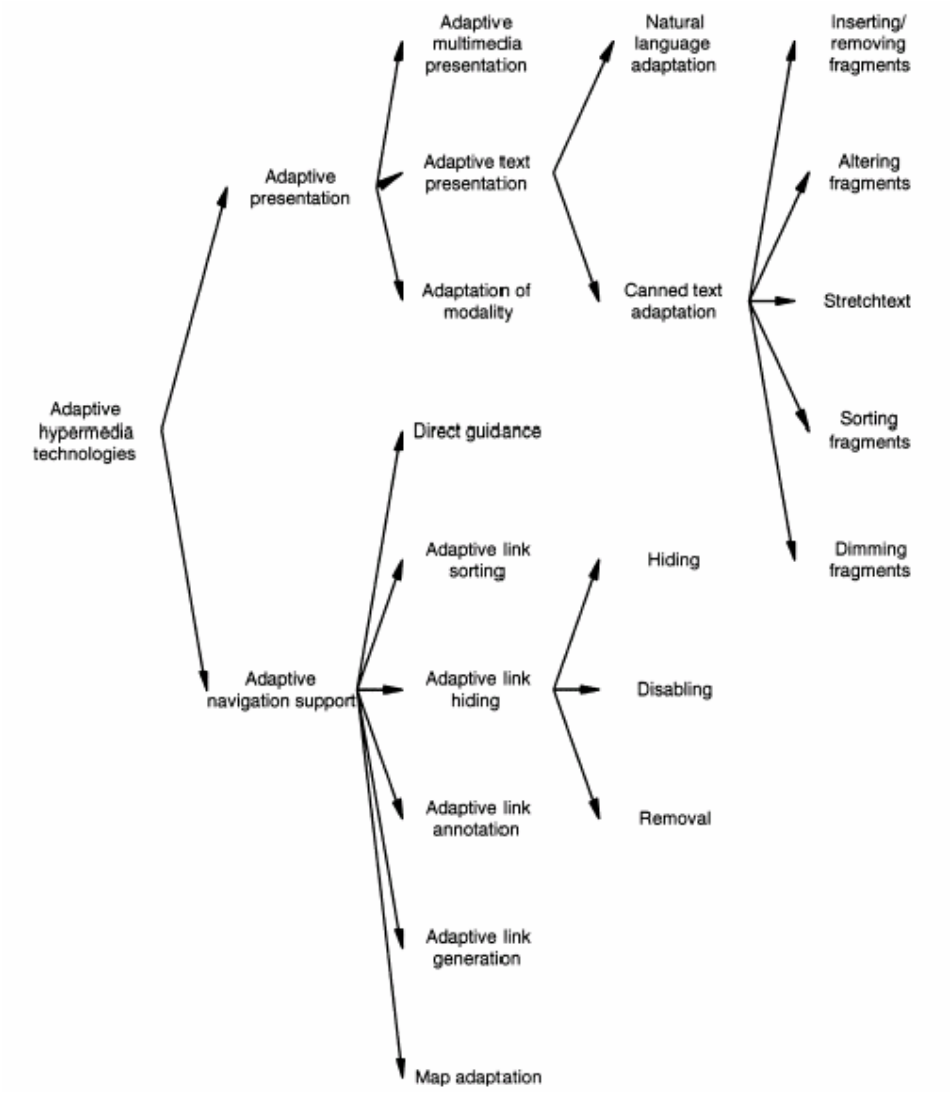
1.2.2. Adaptatie doelgebied

Een adaptief systeem kan aanpassingen bieden op twee doelgebieden [BRUSILOVSKY2001]. Een de gebruiker; twee: zijn omgeving. De eigenschappen van een gebruiker hebben betrekking op zijn kennis, visuele eigenschappen etc. Zijn omgeving heeft te maken met de manier waarop

hij een adaptief systeem benadert. Indien iemand vanuit een GSM een adaptief systeem benadert dan moet de presentatie anders zijn dan vanuit een stand alone computer met een groot scherm. Ook netwerkeigenschappen (snel/langzaam) kunnen gebruikt worden bij het aanpassen van de presentatie.

1.2.3. Adaptieve elementen

Brusilovsky onderscheidt hier twee groepen: content adaptatie en link adaptatie. Content adaptatie is de manier waarop informatie gepresenteerd wordt (b.v. andere kleuren en lettertype) en inhoud van de informatie (welke informatie wordt aan een gebruiker getoond). Link adaptatie is de manier waarop een link naar een andere node gepresenteerd wordt. Brusilovsky heeft hier een taxonomy van gemaakt:



Figuur 2 Taxonomy technologieën van een Adaptieve Hypermedia Systemen

Het afstuderen heeft zich vooral gericht op “Inserting/removing fragments”. Er is ook echter gekeken naar “Adaptive multimedia presentation”.

Een adaptief systeem is opgebouwd uit een aantal basiscomponenten. Deze componenten worden alle beschreven in § 1.3.

1.3. Opbouw van AHS

De volgende hoofdcomponenten zorgen samen voor een adaptief systeem:

- User Model (UM)
- Domein Model (DM)
- Adaptatie Engine (AE)
- Fragmenten en pagina's

Een AHS levert verder nog output, deze output wordt normaliter naar de gebruiker gestuurd. De gebruiker heeft verder in het AHS de taak om informatie aan te vragen, en/of gepresenteerd te krijgen.

1.3.1. User Model

Het User Model bevat allerlei gebruikersinformatie, zoals kennis, (leer)doelen, ervaring, voorkeuren, interesses en individuele gewoonten van een gebruiker. Het Adaptief Hypermedia Systeem kan gebruik maken van dit (persoonlijk) model in het bepalen van de presentatie.

1.3.2. Domein Model

Het domein model beschrijft de relaties tussen verschillende concepten in een applicatie. Een concept beschrijft het gedrag van een pagina fragment of abstract kennis item. Dit gedrag bestaat uit bijvoorbeeld het verhogen van kennis over een onderwerp indien hier een pagina van bezocht is. Relaties kunnen heel abstract gedefinieerd worden, bijvoorbeeld: *als pagina X gezien is dan pagina Y tonen anders pagina Z.* Een domein model wordt ook wel de afspiegeling van de werkelijkheid genoemd. In het domein model worden alle beslissingen die genomen moeten worden door een AE gedefinieerd. Binnen het AHA! systeem is het domein model gerealiseerd door een verzamelingen concepten in de vorm van bestanden. Deze concepten zijn gekoppeld aan pagina's of abstracte concepten. Een abstract concept is een in het UM identificeerbaar object waar mutaties op mogelijk zijn.

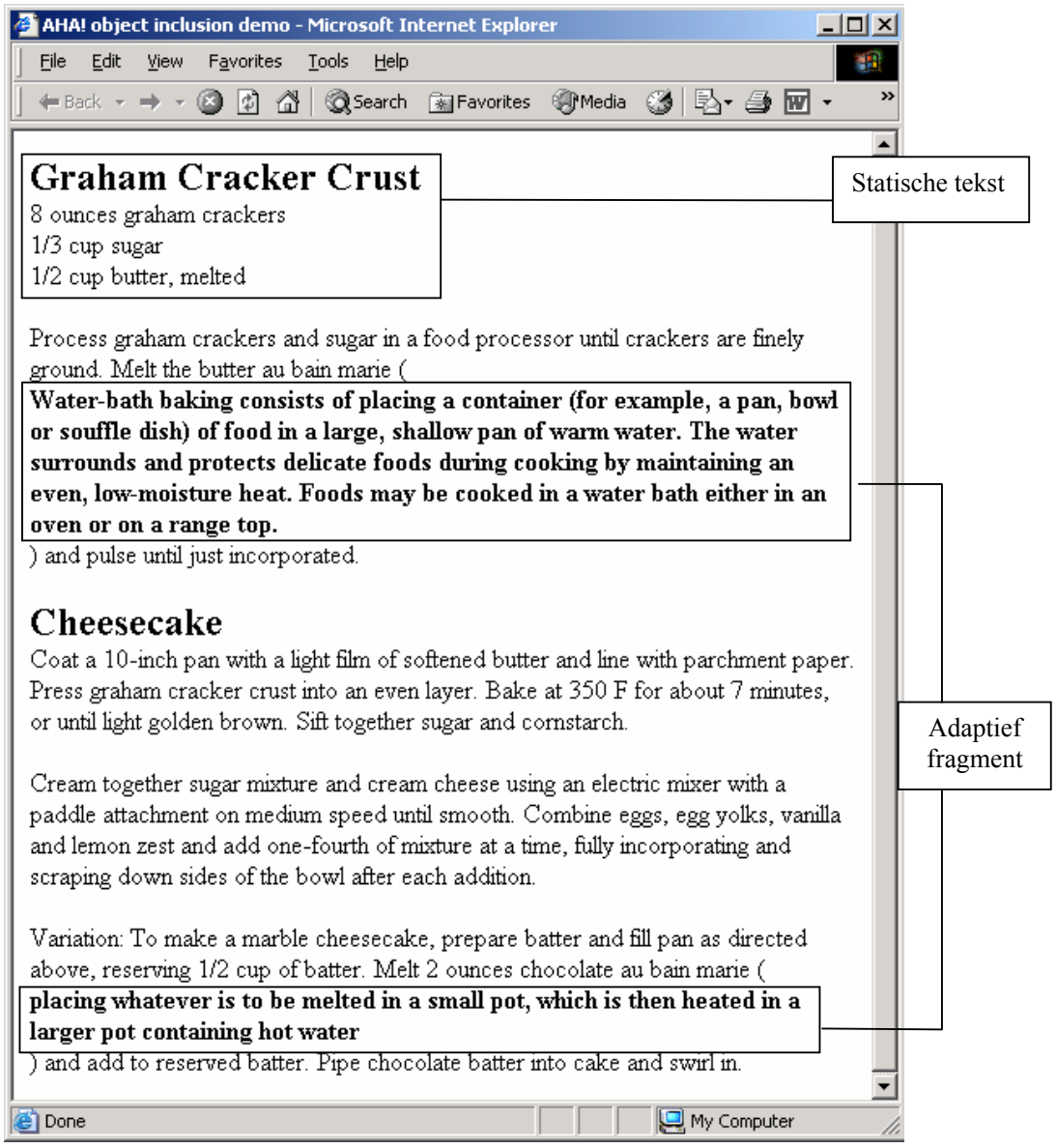
1.3.3. Adaptatie Engine

De adaptatie Engine is het hart van een Adaptief systeem. Hierin wordt op basis van het UM en DM de presentatie van informatie voorbereid en gepresenteerd. De Adaptatie engine krijgt een aanvraag van een gebruiker om een pagina te presenteren. De AE gebruikt het Domein en User model om alle fragmenten bij elkaar te rapen, op een juiste manier te presenteren en de links presentatie aan te passen. In dit proces past de AE het user model indien nodig aan. Vervolgens stuurt de AE deze pagina terug naar de webserver die het vervolgens aan de gebruiker presenteert.

1.3.4. Pagina's en fragmenten

Het vierde onderdeel van een AHS zijn pagina's en fragmenten. Een pagina kan is opgebouwd uit statische fragmenten, dynamische fragmenten en links welke naar andere pagina's wijzen. Een fragment is een uniek identificeerbaar stuk informatie die op een of meerdere pagina's staat. Een statisch fragment is een fragment welke eenmalig door een auteur wordt aangemaakt en vervolgens op een pagina wordt geplaatst (bijvoorbeeld bedrijfslogo of productbeschrijving) een dynamisch fragment is een in zijn presentatie wisselend fragment. Een weerbericht is een dynamisch fragment. Het wordt iedere dag gewijzigd. De prijs van een product is ook een dynamisch fragment. Een fragment kan in een adaptief systeem conditioneel getoond worden. Verder bestaat de mogelijkheid in een adaptief systeem om te kiezen welk fragment er getoond moet worden.

Een pagina heeft bijvoorbeeld de volgende opbouw:



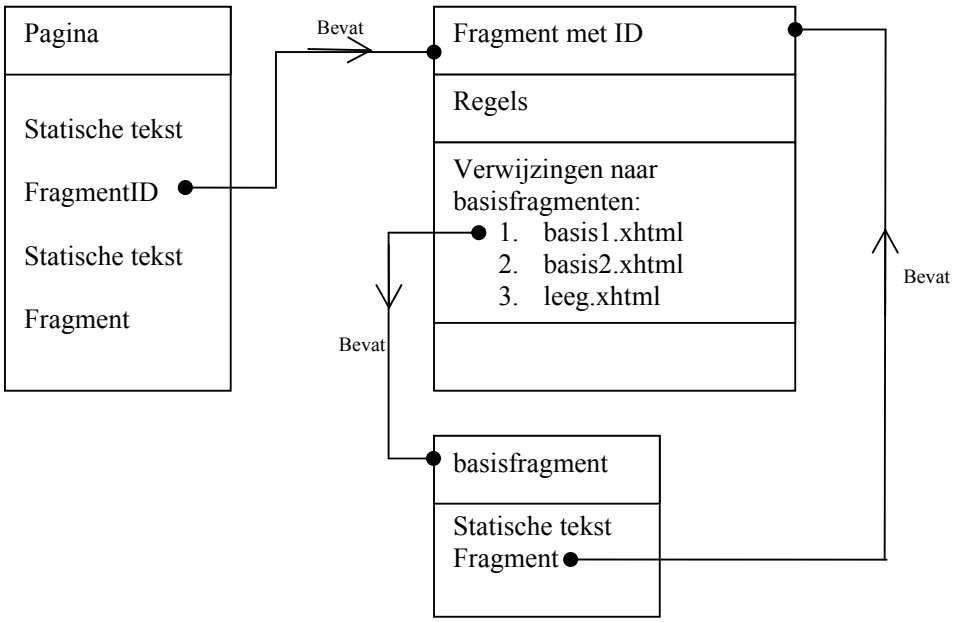
Figuur 3 fragment en pagina voorbeeld

In dit recept wordt een kookgerecht uitgelegd. Het recept is onder te verdelen in stukken statische tekst en stukken dynamische tekst. Een niet veranderend fragment is bijvoorbeeld de titel van het recept. Een dynamisch fragment is de kookterm au bain marie. Op enkele plaatsten wordt deze kookterm gebruikt [DEBRA2003a]. Achter deze kookterm wordt adaptief uitleg over de kookterm gegeven. De eerste keer wordt de kookterm uitgebreid uitgelegd, de 2de keer beknopt. De twee in zwart gedrukte stukken tekst zijn de basisfragmenten die bij het fragment au bain marie horen. Deze basis fragmenten staan los op het systeem opgeslagen (als XHTML bestanden) en worden in het document geplakt door de Adaptatie Engine. De plaats waar ze worden ingeplakt wordt in een document gemarkeerd door een fragment identifier. Dit is de plaatshouder van het fragment.

Dit levert het volgende stappenplan voor een AE op:

1. Bekijk de pagina en zoek naar fragment identifiers
2. bepaal welk basisfragment op de plaats van een fragment identifier geplaatst moet worden (afhankelijk van het UM en DM)
3. Plak het basisfragment op de plaats van de fragment identifier.

In het voorgaande voorbeeld zijn alle fragment identifiers al vervangen door basisfragmenten. Schematisch ziet het er alsvolgt uit:



Figuur 4 conceptueel overzicht pagina-fragment-basisfragment

Dit structuurschema moet als volgt gelezen worden:
Een pagina bevat statische tekst en fragmenten. Een fragment bevat regels welke bepalen welk basisfragment in de pagina gesubstitueerd moet worden. Een basisfragment welke in de pagina gesubstitueerd wordt kan op zijn beurt weer andere fragmenten bevatten.
De hierboven geschetste situatie is een ideaalbeeld. In AHA 1.96 ziet het er conceptueel anders uit. Het onderstaande schema geeft de schematische opbouw van AHA 1.96.

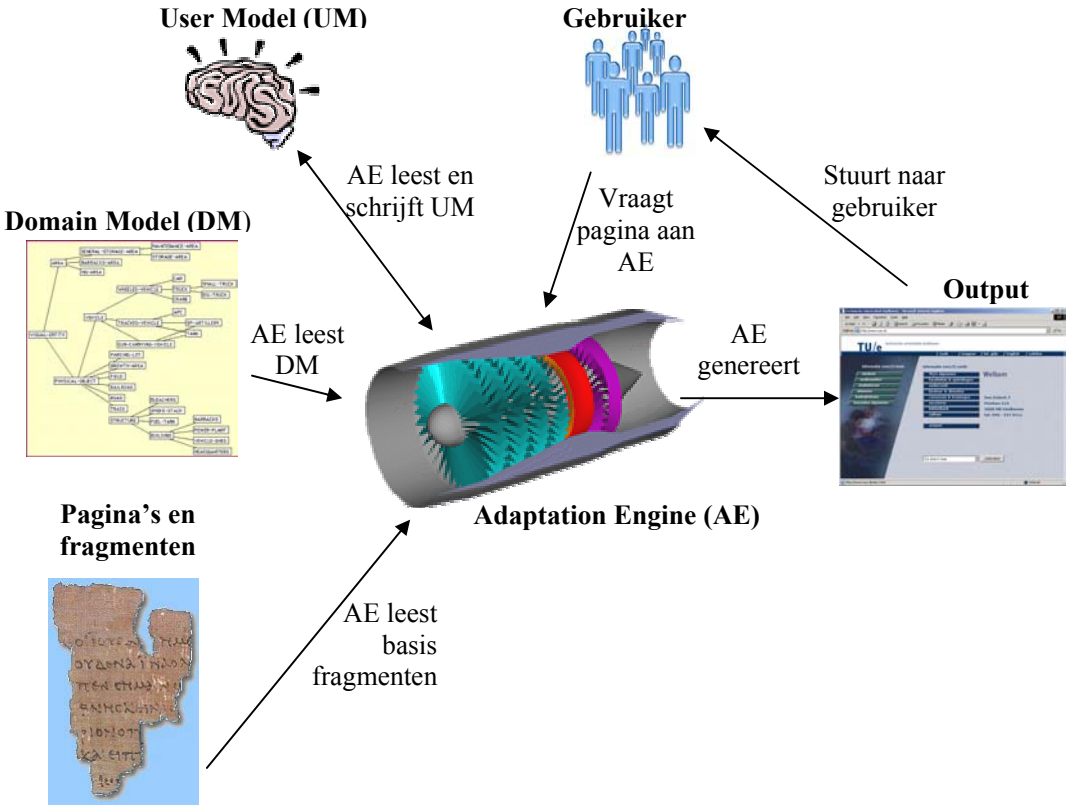
```
Pagina
Statische tekst

Fragment
  Begin fragment
  Conditie
        Conditie goed
          Begin
          Statische tekst
          Einde
        Conditie fout
          Begin
          Statische tekst
          Einde
  Einde Fragment

Statische tekst
```

Dit schema moet als volgt geïnterpreteerd worden: een pagina bevat statische tekst en fragmenten. Een fragment begint met een conditie. Indien deze conditie, die een uitspraak doet over het UM, correct is, dan wordt de tekst tussen "*begin*" en "*einde*" van "*Conditie goed*", op die plek gesubstitueerd. Indien de conditie onwaar is, wordt de tekst die tussen "*begin"* en "*einde*" van "*Conditie fout*"staat, op de plek gesubstitueerd.
Dit schema geeft duidelijk het probleem van AHA 1.96 weer. Fragmenten worden niet losgekoppeld van pagina's. Deze worden allemaal op een pagina geplaatst. Op de pagina staan dus de regels en de inhoud van de fragmenten. Ongeacht of deze getoond worden of niet.

Figuur 5 conceptueel overzicht pagina AHA! 1.96

1.3.5. Overzicht AHA! systeem

AHA! is een AHS (Adaptief Hypermedia Systeem) met alle in 1.3 beschreven componenten.
Een gebruiker vraagt een pagina aan de AE, deze bepaalt middels het DM en UM welke pagina's en fragmenten hij als output moet leveren. Deze output wordt vervolgens in de regel naar de gebruiker gestuurd. Tijdens dit generatieproces wordt het UM zonodig aangepast.
Dit levert het volgende schema op:



Figuur 6 Conceptueel overzicht AHS systeem

In AHA! 1.96 is het domein model echter verweven met de pagina's. Verder wordt er in AHA! 1.96 geen onderscheid gemaakt tussen pagina's en fragmenten. De fragmenten zijn verweven in de pagina's. Een fragment wordt in AHA! 1.96 door een auteur op een pagina geplaatst en eventueel getoond aan de gebruiker door de AE. De AE gebruikt hiervoor enkel en alleen het UM en niet het DM. De fragmenten worden middels een eigen gekozen XML [BRAY1998] syntax in de XHTML pagina's geplaatst. Het AHS herkent deze fragmenten en bepaalt of ze getoond moeten worden. Normaliter zou de keuze of een fragment getoond moet worden beschreven staan in het DM. In AHA! 1.96 is dit niet het geval. In de XHTML pagina's staan deze DM eigenschappen. Dit functioneert bij een beperkt aantal fragmenten en pagina's. Bij een grote hoeveelheid fragmenten en pagina's wordt dit een niet te onderhouden systeem en zijn enkele constructies niet maken. Dit omdat normaliter bij een modulair opgebouwd systeem, het DM en de fragmenten/pagina's strikt gescheiden zijn. Verder zijn grote recursieve structuren niet met de hand te construeren daar men het overzicht snel kwijt raakt. § 4.4 en § 4.6 beschrijven dit probleem

1.4. Onderzoeksvragen

Nu globaal duidelijk is in welke context het afstuderen zich afspeelt, kan de onderzoeksvraag nader uiteengezet worden.
In een AHS worden nodes over het algemeen als pagina's gezien. Tussen deze pagina's kan genavigeerd worden middels links. Op een pagina kunnen meerdere fragmenten met informatie staan. Een fragment is een uniek identificeerbaar stuk informatie binnen een informatie systeem. Een adaptief systeem bepaalt of een fragment en eventueel, welk fragment, het beste aan een gebruiker getoond kan worden. In dit deelgebied van AHS valt mijn afstudeeronderzoek. Een eis van het afstuderen was:

"Onderzoek en implementeer een systeem waarbij geen AHA! specifieke code meer in pagina's voorkomt. Ofwel koppel het DM en pagina's van elkaar los.
Als deze koppeling niet meer bestaat is het ook mogelijk om andere XML georiënteerde talen te gebruiken."
Dit levert een aantal onderzoeksvragen op:

- Op welke manier kunnen fragmenten geïdentificeerd worden (hoofdstuk 3)
- Hoe kan de Domeinmodel kennis uit de pagina's verplaatst worden naar het Domein Model (hoofdstuk 3)
- Waar moeten de basisfragmenten opgeslagen worden (hoofdstuk 3)

Gedurende het afstudeerproces bleek de nieuwe structuur een grote mate van flexibiliteit te introduceren. Deze flexibiliteit is zo groot dat het het functioneren van het AHS in gevaar kan brengen. Vandaar de volgende nieuwe onderzoeksvragen:

- Onderzoek de consequenties van deze nieuwe structuur (hoofdstuk 4)
- Onderzoek oplosmethoden om deze gevaren te herkennen en het systeem ertegen te wapenen (hoofdstuk 4,5)

Hoofdstuk 2 beschrijft hoe het AHA! 1.96 systeem is opgebouwd en functioneert en welke functionele veranderingen er gemaakt gaan worden.

2. AHA!

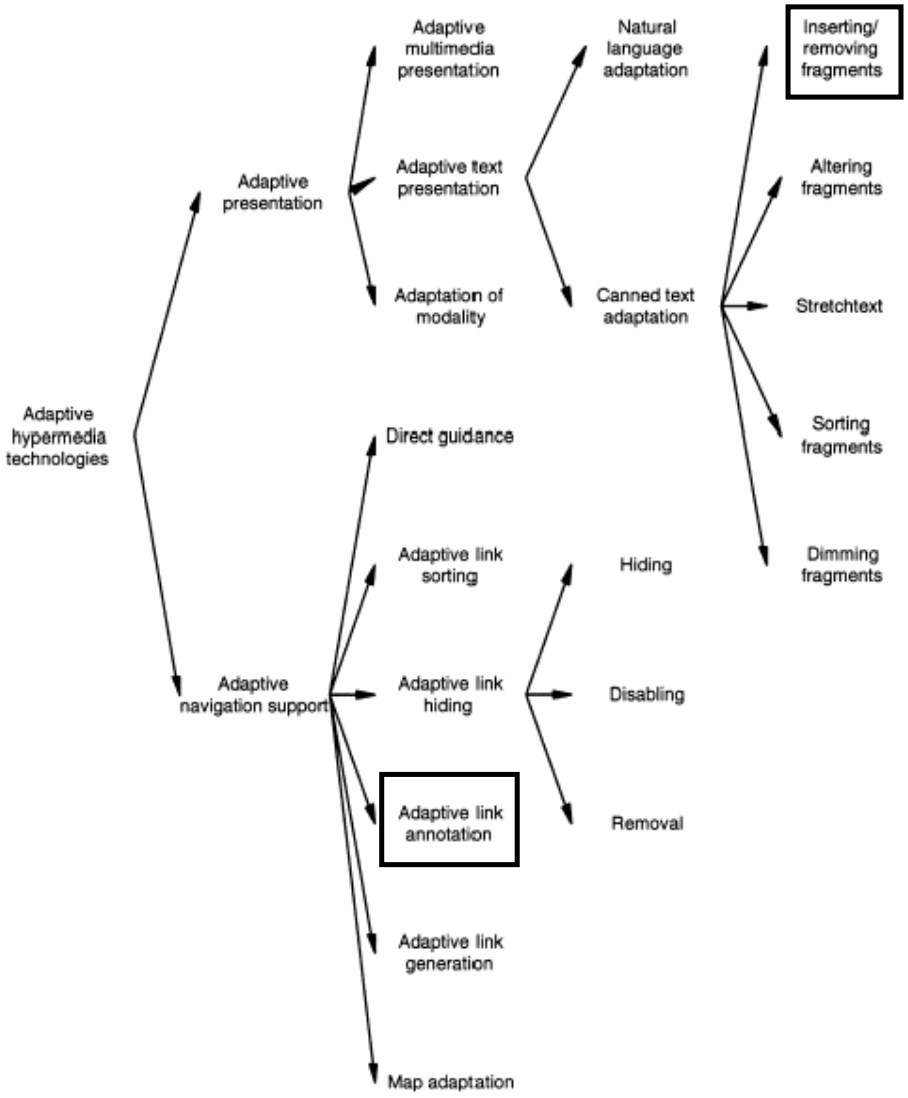
2.1. Inleiding

AHA! is een door de TU/e ontwikkelde AHS. Het is gebouwd in de vorm van JAVA Servlets wat kleine programma's zijn die op een server draaien. Dit hoofdstuk beschrijft kort welke functionele eigenschappen het AHA! systeem heeft, welke functionele uitbreidingen er toegevoegd zijn in versie 3.0. Eerst wordt het AHA! 1.96 systeem beschreven. Vervolgens wordt het uiteindelijke AHA! 3.0 systeem beschreven en alle wijzigingen hierin ten opzichte van het AHA 1.96 systeem.

2.2. Overzicht opbouw in AHA! 1.96

2.2.1. Functie overzicht AHA! 1.96

Zoals in de inleiding beschreven is, AHA! is een AHS. Uit de taxonomy van Bruselovsky (figuur 7) zijn de volgende adaptieve eigenschappen in AHA! 1.96 zwart omkaderd:



Figuur 7 Taxonomy van de in AHA! 1.96 geboden technologieën

Dat dit beperkt lijkt, is numeriek correct. Echter de gemarkeerde eigenschappen zijn de belangrijkste eigenschappen welke een adaptief systeem kan bezitten.
Deze twee eigenschappen worden nader bekeken.
AHA! 1.96 kan fragmenten op een beperkte manier tonen/verbergen. Dit gebeurt door in een XHTML pagina een speciale constructie te gebruiken die de AE herkent en verwerkt. Zo een constructie ziet er als volgt uit:

```
<if expr="ahacook_aubainmarie_knowledge==0">
      <block>
            Au bain marie - koken op of in heet water
            Vul een pan voor ongeveer 3/4 met water, breng het water
            aan de kook. Temper de warmtebron zodra het water kookt.
            Plaats in de pan met het kokende water een hittebestendige
            kom of pannetje met de ingredienten, die gekookt moeten
            worden, bijvoorbeeld een sabayon.
            De hitte die dit hete water overbrengt, zal nooit boven de
            100 C. komen, is veel gelijkmatiger en werkt minder direct
            - dus beter te controleren - waardoor schiften en
            aanbranden minder kans hebben.
      </block>
</if>
```

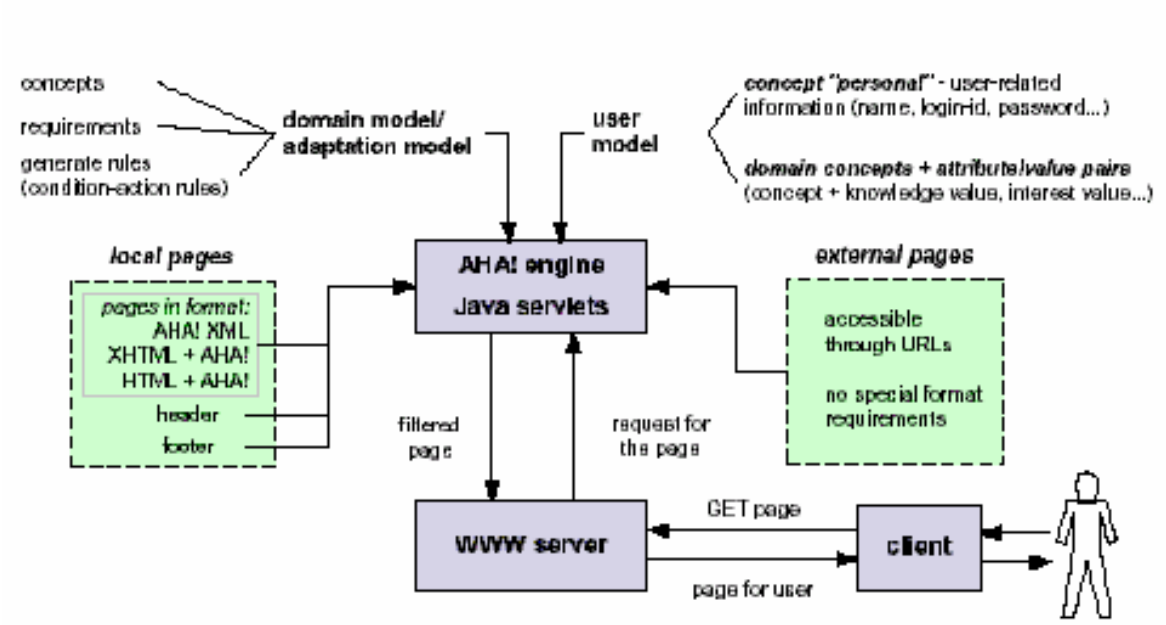
Indien de kennis van aubainmarie nul bedraagt, zal de tekst tussen <block> en </block> getoond worden. De AE herkent de <if> tag en bepaalt of de tekst tussen <block> en </block> naar de gebruiker zijn browser gestuurd moet worden. Deze constructie staat op een XHTML pagina tussen de andere HTML code. Het voordeel van dit soort constructies is, dat er snel een adaptief fragment in een pagina geplaatst kan worden. Een pagina voordat deze door de AE is verwerkte pagina staat hieronder. Als de kennis van aubainmarie 10 is, wordt het fragment tussen <block> en </bock> getoond.

```
<!DOCTYPE html SYSTEM "xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
<h1>AHA v3.0</h1>
<a href="pagina-cookindex.xhtml" > AHA v3.0 The adaptive cook test
case <img src="duck.jpg" height="80" width="80"> </img> </a>
<hr/>
<if expr="ahacook_aubainmarie_knowledge==10">
      <block>
          Kennis van au bain marie is 10.
      </block>
</if>
</body>
</html>
```

Het 2de adaptief onderdeel welke AHA! 1.96 bezit is link adaptatie. Iedere pagina bevat een beschrijvend concept. In dit concept staat het gedrag en de eigenschappen van een pagina. Gedrag in de vorm van wat er moet gebeuren indien een pagina benaderd wordt. En eigenschappen indien een link naar deze pagina wijst. In het concept staat een expressie welke (indien uitgerekend) bepaald of een pagina gewenst is. Dit al dan niet gewenst zijn van een pagina wordt omgezet in een “adaptive link annotation”. Indien een pagina gewenst is, kan de link een blauwe kleur (dit is een gewenste link) krijgen, indien niet gewenst kan de link als gewone tekst getoond worden.

2.2.2. Architectuur AHA! 1.96

De architectuur van het AHA! 1.96 systeem is in figuur 8 afgebeeld.



Figuur 8 Architectuur AHS AHA! 1.96

Figuur 8 is te koppelen met het in hoofdstuk 1 getoond overzicht van een AHS. Wat hier meteen duidelijk wordt is dat de local pages XHTML + AHA! bevatten. Dit is de in § 1.2.1. genoemde eigen syntax definitie.

2.3. AHA! 3.0

2.3.1. Inleiding

Deze paragraaf beschrijft welke functionele wijzigingen tussen AHA! 1.96 en AHA! 3.0 er zijn. Door drie personen zijn er wijzigingen in het systeem aangebracht om het naar versie 3.0 te tillen. Dit is gebeurd op drie verschillende gebieden te weten:

1. Dynamische object insluiting in AHA! door B.H. de Lange [DEBRA2003a], [DEBRA2003b]
2. Zichtbaarheid van Fragmenten door M.L.A. Ansems [stageverslag ANSEMS2003]
3. Stabiliteit binnen AHA! door B.A. Berden [DEBRA2003b] en [afstudeerverslag BERDEN2003]

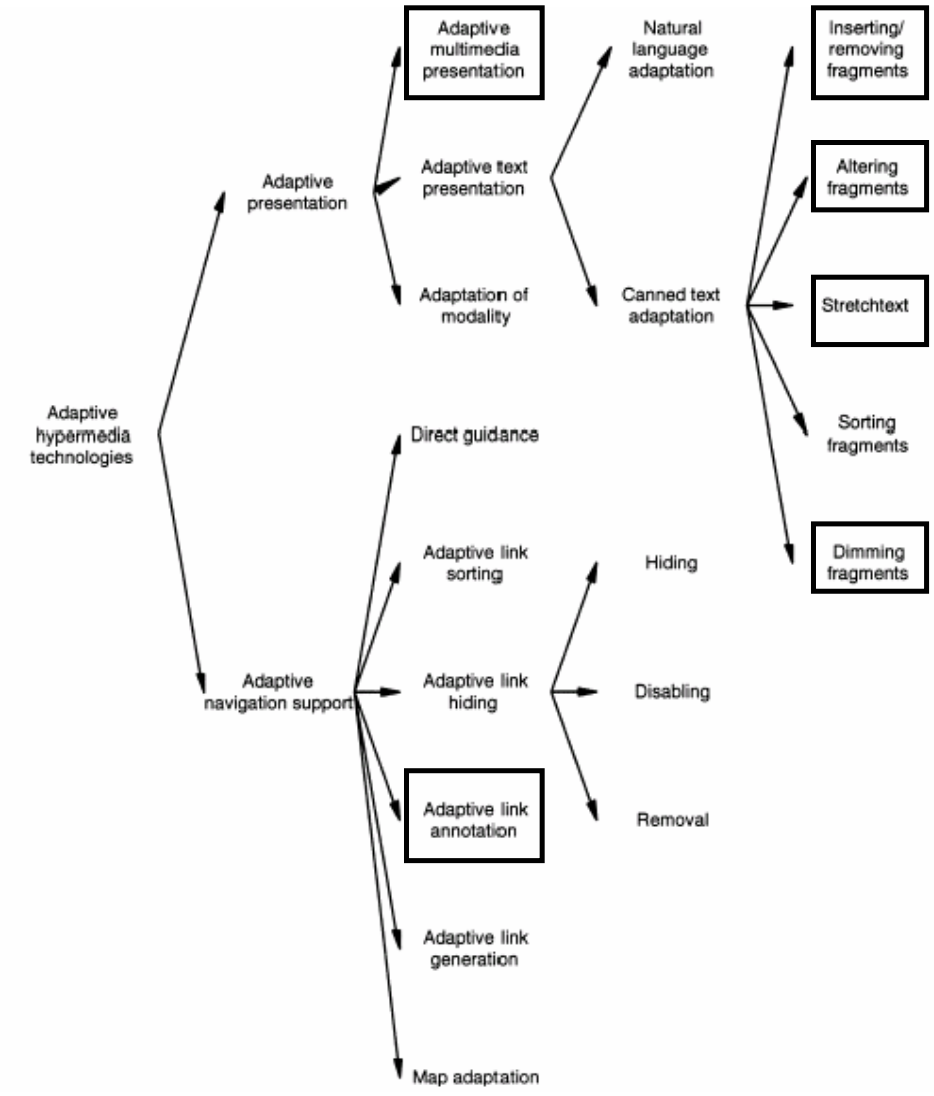
Onderdeel twee en drie worden in dit verslag niet uitgebreid uitgelegd. Ze worden kort aangestipt als wijzigingen tussen AHA! 1.96 en 3.0. In de literatuurlijst zijn verwijzingen te vinden naar documenten welke dieper op deze stof ingaan.

Om object georiënteerde fragment insluiting in AHA! toe te passen waren enkele wijzigingen nodig aan de AE, DM en UM. Allereerst zal worden uitgelegd wat de verschillen zijn in het werken met fragment-objecten. Een van de onderzoeksvragen was een oplossing te bedenken waarmee het gebruik van <if> tags uit de HTML code verdween. Dat dit later enkele andere vragen opriep is te lezen in het vorige hoofdstuk. De concrete technische oplossing voor dit probleem is in het volgende hoofdstuk (hoofdstuk 3) te lezen.
In AHA 3.0 is het mogelijk om op een pagina een fragment te plaatsen die door de engine wordt herkend en verwerkt. Dit verwerken houdt in, dat de engine het fragment controleert op de aanwezigheid van andere fragmenten en deze zonodig ook invoegt op de pagina. Dit invoegen

van fragmenten herhaalt zich tot het laatste fragment geen fragmenten meer bevat. Dan wordt de pagina gepresenteerd aan de gebruiker.
Hoe dit fragment identificatie proces en het verwerking proces is ontworpen en opgelost is in hoofdstuk drie uiteengezet.
Er is onderzoek naar de zichtbaarheid van fragmenten door M. Ansems verricht in december 2002. Het hoofddoel was om adaptief de presentatie van een fragment aan te passen. Met presentatie wordt hier bedoeld, de kleur, helderheid en lettertype van een tekst. Tekst kan op basis van de staat van een user model op verschillende manieren getoond worden. In de taxonomy van Bruselovsky valt dit onder de twee gebieden "*altering fragments*" en "*dimming fragments*". In de praktijk kan dit groot nut hebben. Indien een gebruiker aangeeft dat hij visueel gehandicapt is, kan de AE de tekst bijvoorbeeld groter maken. Verder is er onderzocht hoe er gemakkelijk gebruik gemaakt kon worden van een tooltip. Een tooltip is een presentatiemethode van een fragment waar en "i" teken op de plaats van het fragment zichtbaar zou worden. Als de gebruiker boven deze "i" met de muis hangt wordt het fragment zichtbaar. Dit lijkt op stretchtext [BOYLE1994].

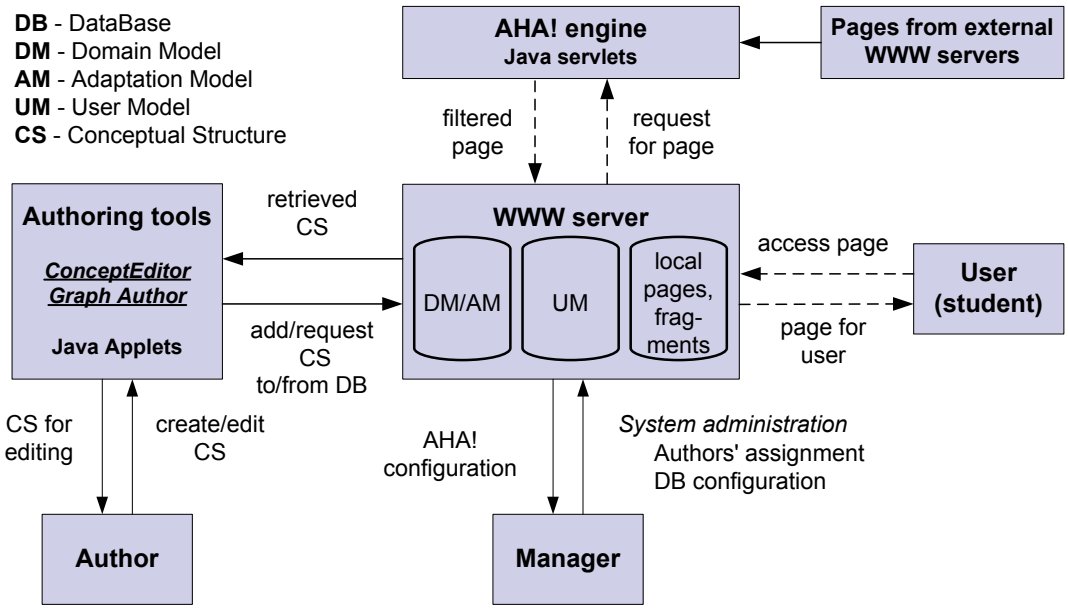
Het laatste veranderingspunt in AHA! 3.0 is gedaan door B. Berden en behelst stabiliteit in adaptieve applicaties. Deze schijnbaar tegenstrijdige titel is niet zo tegenstrijdig als het lijkt. Een gebruiker ervaart in een adaptieve applicatie vaak het gevoel dat pagina's als los zand door de vingers glijden. Eerder geziene definities, veranderen (adaptief) omdat de gebruiker kennis vergaart en op sommige posities geen definities meer uitgelegd hoeft te krijgen. Indien de gebruiker dan terugkeert naar een pagina waar hij een definitie verwacht, kan deze definitie verdwenen zijn. Deze eigenschappen die verbonden is met adaptieve websites vinden veel gebruikers vervelend. Om hier iets aan te doen heeft B. Berden onderzocht welke mogelijkheden er bestaan om gedeeltelijk stabiele pagina's te genereren [DEBRA2003b]. Een van de stabiliteitmethoden was het eenmalig adaptief zijn van een pagina. Als een gebruiker een pagina bezoekt wordt deze eenmalig adaptief opgebouwd. Er wordt dus rekening gehouden met de gebruiker zijn User model (en dus al zijn kenmerken). Daarna wordt de pagina nooit meer op een andere manier getoond dan deze eerste keer. Dit eenmalig adaptief zijn van pagina is een grote toevoeging voor een adaptief systeem en is niet in de taxonomy van Bruselovsky terug te vinden. Er zou een "*Adapt the presentation once*" tak moeten worden toegevoegd met alle standaard "*adaptive presentation*" bladeren en takken eronder.

Structureel verandert er in de pagina's het volgende; de <if> statements verdwijnen en fragmenten worden geïdentificeerd door het systeem. Vervolgens wordt de in 1.3.4. beschreven fragment structuur, welke in een AHS bestaat, doorgevoerd in de paginaopbouw. De paginastructuur is hiermee volledig X(HT)ML compatibel. Het gebruik van louter HTML pagina's is hiermee overboord gezet. Pagina's kunnen elke structuur hebben zolang ze maar aan de eis dat het een wellformed XML document is voldoen. Dit biedt mogelijkheden om AHA! 3.0 op meerdere gebieden in te zetten. Er is geëxperimenteerd met de multimediataal SMIL (zie 4.5). Al deze wijzigingen leveren een aanzienlijke hoeveelheid extra functionaliteit op binnen het AHS AHA!. De in Figuur 9 staande taxonomy geeft alle componenten welke AHA! 3.0 ondersteund zwart omrandt weer.



Figuur 9 Taxonomy van de in AHA! 3.0 geboden technologieën

Dit levert de volgende nieuwe architectuur op:



Figuur 10 Architectuur AHA! 3.0.

Local pages en fragments zijn nu onderdeel van de www server en hangen niet meer los buiten het systeem. Dit door de gewenste splitsing van DM en content.

2.3.2. Wijzigingen in systeemcomponenten

Om deze functionaliteit toe te voegen zijn enkele systeem componenten in het AHA! 1.96 gewijzigd. Er zijn op drie gebieden wijzigingen doorgevoerd, in het User Model om stabiliteit informatie op te slaan, het domeinmodel is aangepast om informatie over stabiliteit op te slaan de nieuwe fragment structuur en de zichtbaarheid van een fragment te kunnen beschrijven en als laatste is de Adaptatie Engine veranderd om de nieuwe gegevens welke in het UM en DM staan te kunnen interpreteren en hiermee de presentatie aan te passen. Om de object georiënteerde fragment structuur te gebruiken zijn enkele delen van de AE veranderd. Dit wordt in hoofdstuk zes beschreven.

3. Oplosmethode

3.1. Aanleiding

Een probleem dat opgelost moet worden, is de <if> constructie's uit XHTML basispagina's te verwijderen dit is de laatste plaats waar DM kennis en content door elkaar geweven zijn. Hiervoor zijn een aantal oplosmethoden bekeken. Het probleem is op te delen in twee deelproblemen;

1. Hoe kan een fragment geïdentificeerd worden.
2. Waar moet de basisfragment-informatie worden opgeslagen en hoe moeten de DM eigenschappen uit de pagina's verplaatst worden naar het DM.

De basis eis dat er geen XHTML vreemde code gebruikt mag worden heeft betrekking op vraag 1. Dit zijn alle tags welke niet door het W3C zijn gespecificeerd [specificatie xhtml]. Dit hoofdstuk beschrijft welke oplossingen er onderzocht zijn voor de hier bovenstaande vragen. Aan het eind van dit hoofdstuk worden alle oplossingen met elkaar vergeleken en wordt gemotiveerd welke er gekozen is in AHA3.0.

3.2. Fragment

Een fragment is een stuk informatie dat in AHA! 1.96 conditioneel ingesloten kan worden. Dit wordt gedaan op basis van de uitkomst van een expressie. Deze expressie heeft een Boolean (waar/onwaar) uitkomst en doet een uitspraak over het User Model. De eerste uitdaging is het specificeren van een fragment en de relatie tot het systeem. In AHA! 1.96 is een fragment een autonoom stukje code welke enkel invloed kan uitoefenen op de presentatie van een pagina. Hieronder staat een voorbeeld van een fragment die conditioneel wordt ingevoegd. Indien de kennis van bainmarie 100 is, wordt de tekst "*U heeft voldoende kennis over Au bain marie, geen extra uitleg nodig."* getoond.

```
<if "expr=ahacook_bainmarie_knowledge=100">
  <block>
      U heeft voldoende kennis over Au bain marie, geen extra uitleg
      nodig.
  </block>
</if>
```

De expressie hoort duidelijk niet thuis in de XHTML pagina maar in het DM. Dit om de modulaire structuur van een AHS te handhaven. Verder hoort de informatie tussen <block> en </block> niet thuis in de XHTML pagina maar in een externe buiten de XHTML pagina bestaand basisfragment. Verder kan het fragment geen directe invloed uitoefenen op het user model. Indien een gebruiker een fragment gezien heeft, is het in AHA! 1.96 moeizaam (maar niet onmogelijk) om hier een UM mutatie aan te koppelen. Theoretisch is dit mogelijk, door in een pagina concept, te bepalen of een fragment getoond gaat worden en hieraan user model wijzigingen te koppelen. Bij meerdere fragmenten op een pagina wordt dit snel een onduidelijk en chaotisch te specificeren proces. In de conclusie zal verder worden toegelicht in welk soort situaties het zo goed als onmogelijk is. De doelstelling van een fragment in AHA! 3.0, is ervoor te zorgen dat het een een autonoom onderdeel in de AHA! applicatie wordt. Een fragment bepaalt zelf middels zijn beschrijvend concept (welke een onderdeel is van het DM), hoe hij zichzelf wil presenteren aan de gebruiker. Tevens kan een fragment in AHA! 3.0 directe invloed uitoefenen op het user model. Een fragment wordt een "first class citizen", het heeft net zoveel mogelijkheden als een pagina.

3.3. Fragment identificatie en verwerking

Om fragmenten deze vorm van functionaliteit toe te kennen, moet het AHA! systeem in staat zijn een fragment te herkennen en te identificeren. Het systeem moet een fragment declaratie herkennen en een concept hieraan koppelen. Verder moet het systeem, indien hij een fragment herkent, acties ondernemen om het fragment te verwerken. De volgende vier methoden van fragment herkenning/identificatie en verwerking zijn onderzocht:

1. Iframe
2. SSI
3. AHA! definitie
4. Object definitie

Aan iedere oplossing zijn een aantal criteria gesteld waaraan ze moeten voldoen. Deze zijn:

- Gebruiksvriendelijkheid voor auteur
- Performance
- Universeel gebruik XHTML code
- Flexibiliteit voor complexe structuren
- Geen onoverbrugbare problemen

3.3.1. IFRAME

Als eerste is onderzocht of er gebruik gemaakt kan worden van IFrame's. Een iframe ofwel een Inline Frame is een HTML constructie welke het toestaat binnen een Frame een ander document te plaatsen [RAGGETT1999]. Hieronder is een voorbeeld van een IFRAME constructie.

```
<IFRAME SRC="voorbeeld.html" WIDTH="275" HEIGHT="105"
FRAMEBORDER="0" SCROLLING="no" > </IFRAME>
```

In dit voorbeeld wordt op de plaats waar deze constructie staat het document voorbeeld.html gesubstitueerd. In een AHA! systeem substitueert de adaptatie engine de string "*voorbeeld.html"* naar een door de engine bepaald fragmentnaam. De fragmentidentificatie wordt bepaald middels het veld SRC. Een fragment wordt herkend door het IFRAME tag en de naam die in SRC staat. Hier zal een speciale naam voor gebruikt moeten worden, om standaard IFRAME constructies te onderscheiden van AHA IFRAME constructies. Het SRC veld zou de volgende syntax kunnen hebben: "aha.fragmentnaam". Het is onwaarschijnlijk dat er op een niet AHA! manier een veldnaam geconstrueerd wordt welke begint met "aha.". De adaptieve engine kan op basis van het door hem beschikbare User model van de ingelogde gebruiker bepalen welk fragment door dit IFRAME getoond moet worden. Een voordeel van deze constructie is dat het gebruik maakt van universele HTML code. Verder is het ontwerpen van adaptieve pagina's eenvoudig, omdat deze HTML constructie in de huidige HTML editors terug te vinden is. Als nadeel, is het onoverbrugbare probleem van vaste hoogte en breedte welke Iframes hebben, aan te voeren. Er moet vooraf bepaald worden welke afmetingen een Iframe heeft. Dit is veelal browser afhankelijk. Indien een gebruiker een groot lettertype ingesteld heeft, kan dit leiden tot schuifbalken aan rechter en onderkant van het scherm. Indien een frame een ander frame insluit ontstaat er een groter probleem. Ieder parent frame moet exact weten wat het formaat van zijn kinderen is, de kinderen moeten dit ook weer weten van hun kinderen. Dit proces gaat door tot het laatste frame welke geen kinderen meer heeft.

3.3.2. SSI

SSI (Server Side Inclusions) is een extensie van een webserver. Deze extensie stelt de auteur in staat om middels de commentaar tag een bestand in te voegen. De syntax is:

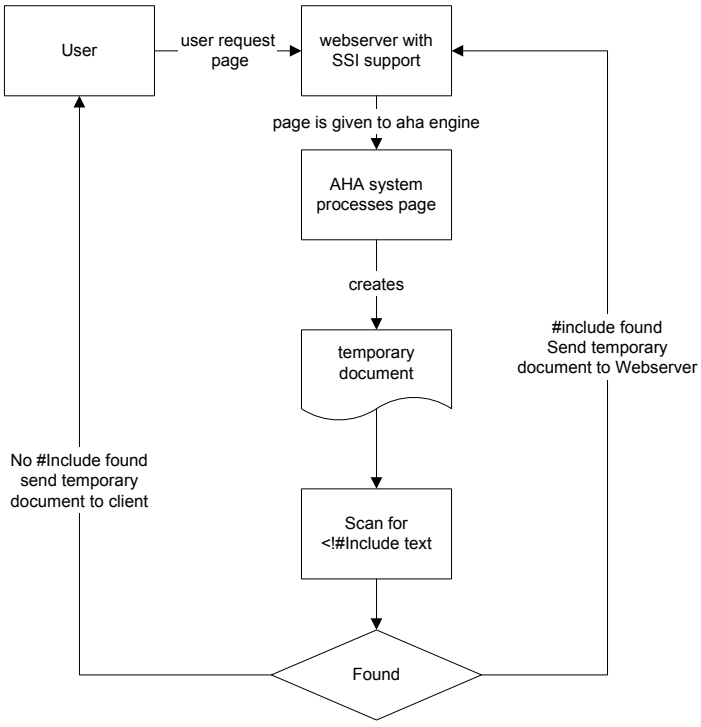
```
<!--#INCLUDE FILE="bestandsnaam.extensie"-->
```

Indien een webpagina de extensie shtml krijgt, controleert de webserver de pagina op de hierboven beschreven syntax. Indien de webserver deze syntax vindt, probeert hij het bestand "bestandsnaam.extensie" op de plaats van de declaratie in te voegen. De adaptieve engine vervangt net als de eerste oplossingmethode de bestandsnaam, in een door de engine goed bevonden fragmentnaam. Vervolgens wordt de pagina opnieuw aangeboden aan de webserver. Daarna wordt de pagina die de webserver op basis van deze pagina genereert opnieuw gecontroleerd.

Stap voor stap:

1. De gebruiker vraagt aan het systeem een pagina op
2. Pagina wordt aan het AHA! Systeem doorgegeven.
3. AHA! systeem creëert een tijdelijk document met daarin de juiste "#include" verwijzingen
4. Pagina wordt gescand op "#include" verwijzigen
5. Indien aanwezig, wordt de pagina opnieuw aan de webserver met SSI ondersteuning aangeboden
6. Dit proces herhaalt zich tot er geen include verwijzingen meer op de pagina aanwezig zijn. Dan wordt de pagina aangeboden aan de gebruiker.

Het schema op de volgende pagina beschrijft deze oplosmethode:



Figuur 11 Stroomschema SSI fragment verwerking

De webserver is geoptimaliseerd om snel bestanden op een pagina te plaatsen. Dit heeft als gevolg dat de performance van deze oplossing goed lijkt. Echter door veel communicatie tussen de webserver en de AHA! engine kan deze performancewinst teniet gedaan worden. Een voordeel is dat sommige HTML editors de mogelijkheid bieden om deze constructies toe te voegen, wat prettig is voor de auteur. Het verplichten van een webserver met SSI ondersteuning is als een onoverbrugbaar probleem gekenmerkt. Het gebruik van de commentaar tag is wel door W3C gestandaardiseerd maar er is geen betekenis aan gegeven.

3.3.3. AHA! definitie

De derde oplosmethode is een fragment identificeren door een nieuwe syntax te definiëren.
Binnen XHTML bestaat de mogelijkheid om commentaar te plaatsen. De inhoud van het commentaar hoeft niet te voldoen aan enige syntax. Dit lijkt op de SSI methode van fragment identificatie. Commentaar ziet er in XHTML als volgt uit: "<!-- commentaar -->" Deze commentaar constructie zou vervolgens gebruikt kunnen worden om een fragment te identificatie. Bijvoorbeeld:

```
<!–AHAconditionalfragment.nr:=”1209”>
```

De adaptieve engine bepaalt vervolgens welke informatie er op de positie van deze declaratie gesubstitueerd moet worden. Een schema van verwerking is te vinden in de volgende oplosmethode. Deze is identiek qua verwerking, enkel de identificatie van het fragment is anders.

3.3.4. Object Tag constructie

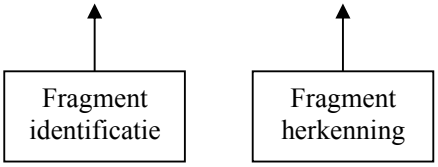
De laatste oplosmethode is gebruik maken van een algemene (X)HTML constructie, de <object> tag. Deze tag is in de toekomst de vervanger van de IMG, APPLET, EMBED en BGSOUND tag. Deze tag identificeert objecten zoals applets, plaatjes en geluidsfragmenten. Door aan te geven welk type het object heeft, bepaalt de browser welke aan de browser gekoppelde viewer hij gebruikt, om het object af te beelden/verwerken.

De syntax van de object tag is de volgende:

```
<OBJECT NAME="naam van object" DATA="bestand.ext"
type="objectype" width="int" height="int"></OBJECT>
```

Het object type heeft net als de IFRAME constructie ook een hoogte en een breedte. Echter de hele tag inclusief de attributen wordt door de AE vervangen door het in gevoegde basisfragment. In tegenstelling tot de IFRAME constructie welke blijft staan op de pagina, wordt de Object constructie compleet vervangen door de AE. Voor de AHA! 3.0 fragment identificatie, zijn niet alle velden nodig. Er is gekozen om het "name" veld te gebruiken om het fragment mee te identificeren. Om er voor te zorgen dat de browser (indien er geen AHA! Engine wordt gebruikt) het object niet probeert te tonen moet er in het type veld een mime type staan welke niet bekend is bij de browser. Het Mime type "aha/text" is op dit moment niet gekoppeld aan een viewer, dus bruikbaar binnen deze vorm van fragment herkenning.
Een voorbeeld van een AHA! object is:

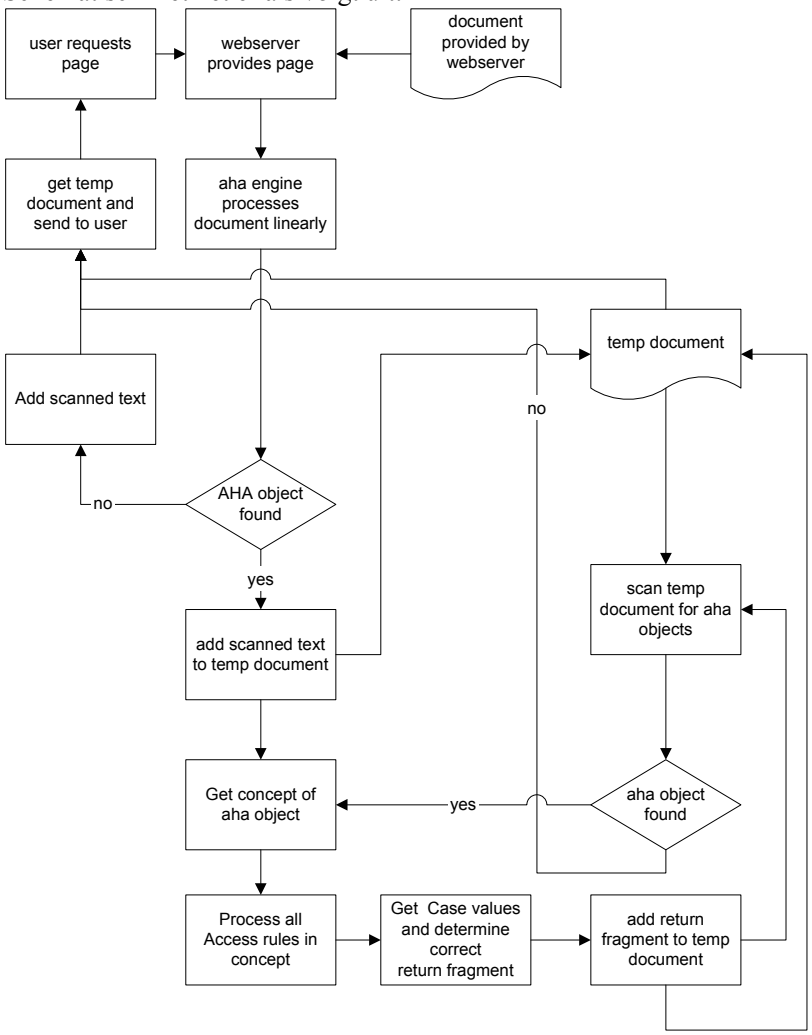
```
<OBJECT NAME="fragment_naam" type="aha/text" </OBJECT>
```



Het schema om met deze manier van fragment identificatie verwerking te werken is de volgende:

1. Gebruiker vraagt een pagina aan de webserver
2. De webserver geeft deze pagina aan de AHA! engine
3. De AHA! engine scant deze pagina van boven naar beneden op <object> tags
4. Indien er een AHA! object tag gevonden wordt, worden alle UM mutaties beschreven in het concept uitgevoerd
5. Het in te voegen basisfragment wordt bepaald op basis van condities die een uitspraak doen over het UM en beschreven staan in het concept van dit fragment.
6. Het return fragment wordt gecontroleerd op nieuwe fragment declaraties, indien niet aanwezig, wordt het return fragment in het document gesubstitueerd op de plaats van de fragment declaratie
7. Indien er een nieuwe fragment declaratie wordt gevonden in het fragment wordt het proces bij stap vier herhaald met dit fragment als basisfragment
8. Dit recursieve proces wordt herhaald tot er geen fragment declaraties meer gevonden worden
9. Pagina met alle fragmenten presenteren aan de gebruiker.

Schematisch ziet het er als volgt uit:



Figuur 12 stroomschema object tag verwerking

Indien er in een ingesloten fragment een nieuwe object declaratie voorkomt, wordt dit op die positie verwerkt en wordt er eventueel een nieuw fragment ingevoegd. Dit proces houdt op wanneer er geen nieuwe fragmenten declaraties in het document aanwezig zijn. Deze oplosmethode heeft als voordeel dat het object type standaard in HTML editors aanwezig is wat prettig is voor de auteur. Verder is deze manier van fragment afhandeling erg snel. Het systeem hoeft geen onodige communicatie met andere partijen te ondergaan. Het gebruik van de object tag is opgenomen in de W3C standaard van XHTML. Er worden dus geen obscure tags gebruikt of misbruikt.

3.4. Keuze van oplosstrategie

Globaal zijn de oplosmethoden onder te verdelen in twee groepen. De eerste groep laat de fragment substitutie over aan een webserver. Groep twee substitueert het gekozen fragment zelf. De eerste heeft de voorkeur, immers een webserver is geoptimaliseerd om dit proces uit te voeren. Zelfgeschreven code zal in regel minder efficiënt zijn.
Oplossing één heeft een niet te overbruggen probleem. In een IFRAME constructie moet worden gedefinieerd welke afmetingen dit frame heeft. Een frame welke zelf zijn grootte bepaalt is niet mogelijk. Indien een frame vervolgens een ander frame insluit moet het frame waarin dit geopend wordt zijn grootte aanpassen om wederom geen schuifbalken te krijgen. Verder worden IFRAME's op verschillende browsers verschillend gepresenteerd. Om deze reden alleen, is de IFRAME oplossing afgevallen. SSI razendsnel omdat hiervoor speciaal geoptimaliseerde code is geschreven. Dit brengt meteen een nadeel naar boven. De webserver moet deze speciaal geoptimaliseerde code bevatten. Er moet SSI ondersteuning op de webserver aanwezig zijn. Een ander nadeel is het (mis)gebruik van de commentaar tag. Indien een auteur besluit om in zijn commentaar "#include voorbeeld" te schrijven wordt dit door de webserver herkent en verwerkt als zijnde SSI instructies. Verder moet de pagina bij elke declaratie aan de webserver worden aangeboden. De webserver zal iedere keer deze pagina verwerken totdat de AHA! engine geen #include declaraties meer tegenkomt. De manier waarop de webserver met niet geldige declaraties omgaat is niet duidelijk en ook niet getest.
De twee laatste oplossingen zijn op de identificatie van het fragment na hetzelfde. De engine bepaalt op grond van het gedeclareerde fragment welk document hij moet invoegen op de plaats van declaratie. Dit invoegen gebeurt door de AHA! engine. De AHA! engine controleert dit fragment vervolgens op nieuwe fragment declaraties. Deze methode heeft als groot voordeel dat er geen onnodige communicatie tussen de AHA! engine en de webserver hoeft plaats te vinden.
De zelfgedefinieerde syntax heeft hetzelfde nadeel als de SSI oplossing. De <object> tag oplossing heeft uiteindelijk de voorkeur gekregen omdat de meeste HTML editoren standaard een voorziening voor deze constructie bieden. Het is de meest logische beschrijving welke in de HTML taal gebruikt kan worden. Een nadeel van de object tag, is het probleem dat andere XML talen deze constructie niet herkennen. Andere talen zoals SMIL (**S**ynchronized **M**ultimedia **I**ntegration **L**anguage) bieden geen ondersteuning voor de object tag. Er wordt een uit die taal gekozen tag gebruikt om een fragment te herkennen. Dit wordt op dezelfde manier gedaan als van de object tag gebruik gemaakt is. Er wordt een tag gekozen waar minimaal twee velden vrij invulbaar zijn, een voor de fragment identificatie, een voor de fragment herkenning. In hoofdstuk 4 wordt over SMIL meer geschreven. Verder wordt er uiteengezet welke tag hiervoor gekozen is.

In de volgende tabel is aangegeven welke voor en nadelen de bekeken oplosstrategieen hebben.

Criteria/Oplossing	IFRAME	SSI	AHA! Definitie	Object TAG
Ondersteuning in auteurs tools	++	0	--	++
Snelheid van verwerking	0	0	+	+
Onoverbrugbare problemen	-- (1)	-- (2)	++	++
Flexibiliteit voor complexe structuren	-	++	++	++
W3C standaard	++	--	--	++
beheersbaarheid	--	++	++	++

(1): vaste hoogte breedte frames
(2): eis van gebruik SSI ondersteuning in webserver

3.4.1. Conclusie keuze oplosstrategie

Er is gekozen om de <object> constructie te gebruiken als fragment identificatie. Deze HTML constructie komt het dichtst bij de toepassing waarin wij fragmenten gebruiken. Verder zijn de nadelen van die bij deze constructie horen minimaal. Een probleem wat opdook na de implementatie van dit oplosschema, is dat het maximale aantal fragmenten op een pagina gedefinieerd moet worden, teneinde het systeem niet te laten crashen. Dit is een implementatietechnisch probleem en heeft niets te maken met de object tag structuur maar met de keuze om de AHA! engine zelf fragmenten in te laten sluiten. De Iframe en SSI oplossing vallen af om de problemen van “framegrootte bepalen” en de eis van een een webserver met SSI ondersteuning. De keuze tussen de zelfgedefinieerde en <object> tag is minder triviaal. De commentaar tag gebruiken heeft grote voordelen indien er verschillende XML georiënteerde talen gebruikt gaan worden. In dat geval worden er geen voor die talen vreemde tags gebruikt. Dit voordeel weegt niet op tegen het nadeel dat er geen gebruik gemaakt kan worden van standaard HTML editors. Een HTML editor kan commentaar tag’s genereren maar daarna moet de gebruiker zelf alles specificeren. Indien een gebruiker de Object tag in een HTML editor wil plakken, bied de HTML editor meer hulp bij het invullen van velden. Een gebruiker hoeft dus niet in de HTML code te duiken om een AHA! object in een HTML pagina toe te voegen. Dit moet hij wel doen indien hij gebruik wil maken van de commentaar tag. Een object tag kan dus simpel met een muis op de gewenste pagina geplaatst worden.

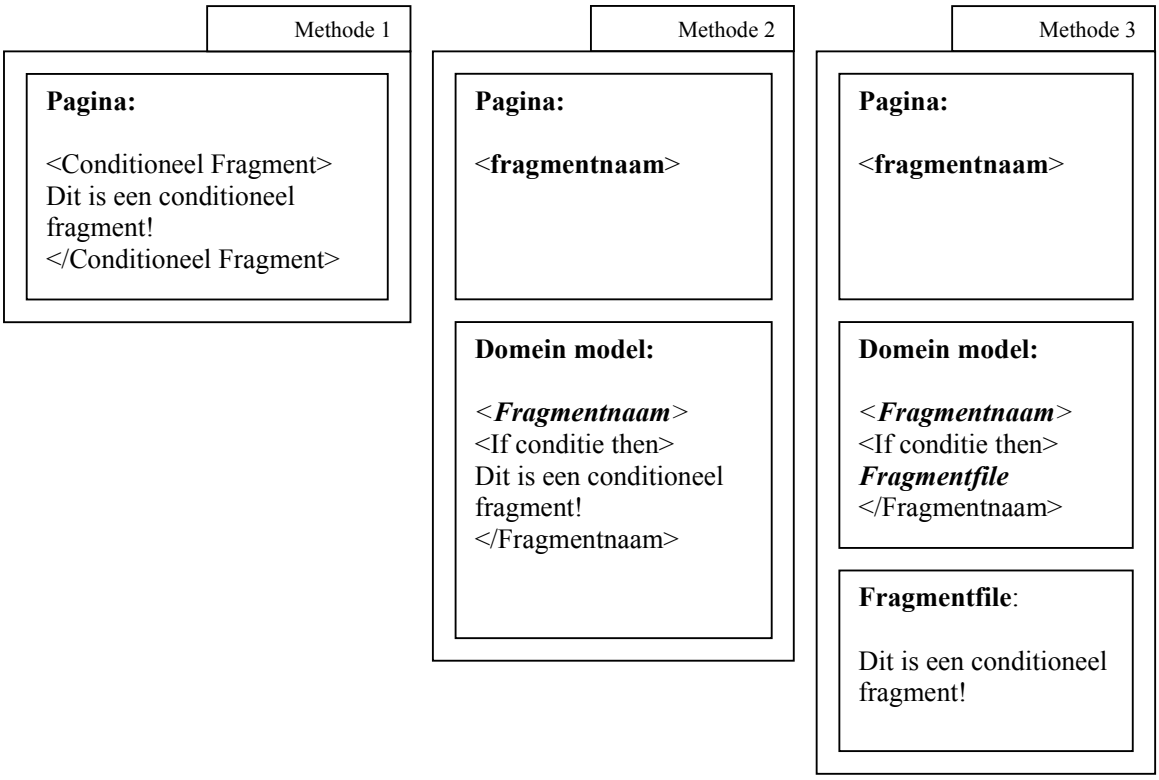
3.5. Fragment opslag door DM/content splitsing

Het laatste op te lossen deelprobleem was de plaats waar de fragment content (basisfragment) opgeslagen moet worden. De content van een fragment kan op verschillende plaatsen worden opgeslagen:

1. In een XHTML pagina
2. In het domein model
3. Zelfstandig binnen het Filesysteem of ander opslagsysteem

In AHA 1.96 staat de fragment content opgeslagen in de XHTML pagina. Dit is ongewenst om architectuur en onderhoud redenen: Het breekt de modulaire opbouw van een AHS systeem af. De tweede methode; content in het domein model opslaan. Dit is niet gewenst daar dit een niet te onderhouden situatie oplevert. Als er content in het domein model wordt opgeslagen, wordt het domein model onnodig zwaar en groot. Het domein model beschrijft op een abstracte manier de werkelijkheid en bevat niet de werkelijkheid. In het domein model staan wel verwijzingen naar de locatie waar het document daadwerkelijk opgeslagen is. Dit is meteen de derde en laatste methode. De fragment content onderbrengen in het filesysteem gebruikmakend van losse bestanden per fragment of een andere dataopslag methode waarbij de content losgekoppeld is van DM eigenschappen. Een database zou ook geschikt zijn als fragment opslag systeem.

Hieronder schematisch de drie methoden uiteengezet:



Figuur 13 fragment content opslag

Wat direct opvalt, is dat er naarmate de methode verfijnder wordt, de overhead en aantal lagen groter wordt. Een voordeel van deze extra lagen is natuurlijk het ontrafelen van fragment content en fragment adaptatie uit zowel de XHTML pagina's alsmede het Domein Model. Er is gekozen voor methode #3. Ondanks de toename aan overhead maakt dit een AHS modulair. Het verweven van componenten in een AHS verminderen de kracht van specificatie, onderhoud en constructies welke niet meer mogelijk zijn door deze verweving.

4. Theoretische consequenties

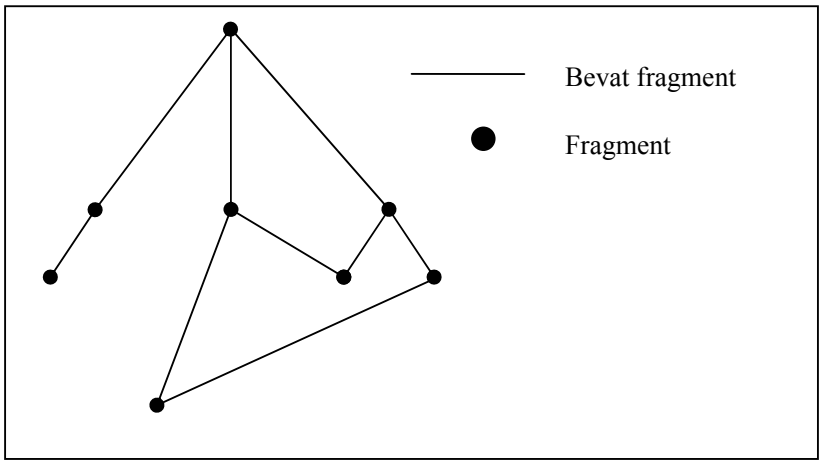
4.1. Inleiding

Uit hoofdstuk twee is gebleken dat de object structuur gebruikmakend van de <object> tag de meest geschikte is voor het gebruik binnen AHA!. Dit hoofdstuk beschrijft de consequenties van deze structuur binnen het AHA! Systeem. Doordat objecten andere objecten kunnen insluiten is het gebruik van deze nieuwe structuur niet zonder gevaren. Oneindige insluiting van objecten is een reëel gevaar. Hoe deze en andere effecten, van de nieuwe structuur, passen in het grote beeld van AHA!, wordt in dit hoofdstuk uiteengezet.
De volgende consequenties worden nader bekeken:

1. structuurverandering meerdimensionale object structuur
2. gevaar oneindige recursie en terminatieproblemen
3. verminderd overzicht op een applicatie
4. AHA! meets SMIL
5. Hergebruik van fragmenten
6. Autonoom gedrag van een fragment

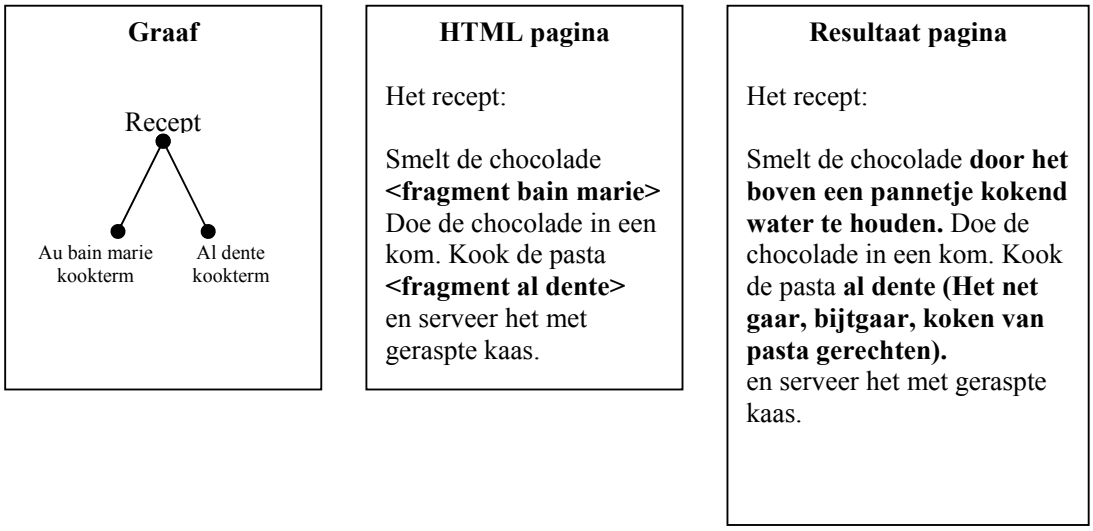
4.2. Meerdimensionale object structuur

Onze gekozen oplosstrategie staat het toe, dat fragmenten andere fragmenten insluiten. Een fragment is, zoals beschreven in hoofdstuk een, een XHTML document. In dit XHTML document kan een object verwijzing worden geplaatst met de Object tag. Deze structuur is vervolgens als een graaf te visualiseren.



Figuur 14 node fragmenten graaf

Iedere graaf moet van boven naar beneden gelezen worden en bestaat uit lagen. Uit bovenstaande graaf is duidelijk te zien dat de applicatie zich niet als een boom gedraagt. De graafdiepte en complexiteit zijn niet direct af te leiden. Een fragment kan op meerdere plaatsen gebruikt worden. In een adaptieve onderwijs applicatie kan een definitie op meerdere plaatsen voorkomen. Onderstaand voorbeeld geeft een reële voorstelling van een applicatie waarbij uiteindelijk zelfinsluiting tot de mogelijkheden behoord.
Om deze notatiemethode verder toe te lichten hieronder een kort voorbeeld van een conceptuele graaf met de fragment structuur, de HTML pagina met fragment structuur en daarnaast de resultaat pagina welke aan de gebruiker wordt getoond.



Figuur 15 voorbeeld concept→ structuur → presentatie

Om recursieve problemen te illustreren volgt hieronder een groter voorbeeld met alle componenten om een probleem applicatie te ontwerpen.

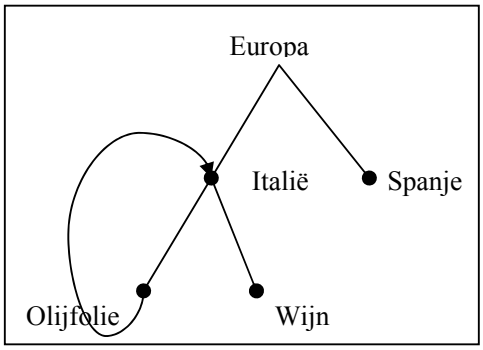
Stel een onderwijs applicatie voor, handelend over Europa. De volgende kennis is gedefinieerd:

- Italië
- Olijfolie
- Wijn

Met als definities:

- Italië: Italië, land aan de middellandse zee. Bekend om de grootschalige productie van **[wijn]** en **[olijfolie]**.
- Olijfolie: Olijfolie wordt bereid uit de persing van olijven. Olijven zijn terug te vinden in het Middellandse-Zeegebied en specifiek **[Italië]**. Olijfolie geeft een specifieke zuiderse smaak aan uw gerechten.
- Wijn: het product dat uitsluitend verkregen is door gehele of gedeeltelijke alcoholische vergisting van verse druiven of van druivenmost.

Dit is uit te drukken in onderstaande graaf:

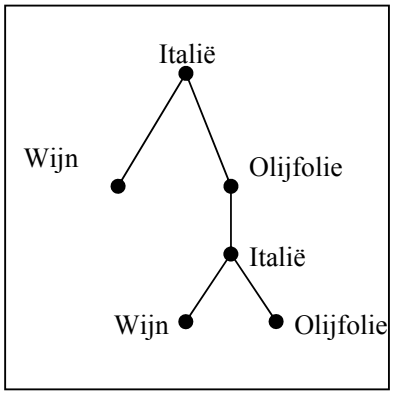


Figuur 16 structuurgraaf onderwijsapplicatie Europa

Dit zou de analoog aan het voorbeeld in het begin van dit hoofdstuk de volgende hypertext pagina kunnen opleveren:

Italië: Italië, land aan de middellandse zee. Bekend om de grootschalige productie van **[wijn] (**Wijn: het product dat uitsluitend verkregen is door gehele of gedeeltelijke alcoholische vergisting van verse druiven of van druivenmost.) en **olijfolie (**Olijfolie: Olijfolie wordt bereid uit de persing van olijven. Olijven zijn terug te vinden in het Middellandse-Zeegebied en specifiek **[Italië]** (Italië: Italië, land aan de middellandse zee. Bekend om de grootschalige productie van **[wijn] (**Wijn: het product dat uitsluitend verkregen is door gehele of gedeeltelijke alcoholische vergisting van verse druiven of van druivenmost.) en **olijfolie (**Olijfolie: Olijfolie wordt bereid uit de persing van olijven. Olijven zijn terug te vinden in het Middellandse-Zeegebied en specifiek **[Italië]**. Olijfolie geeft een specifieke zuiderse smaak aan uw gerechten.)). Olijfolie geeft een specifieke zuiderse smaak aan uw gerechten.).

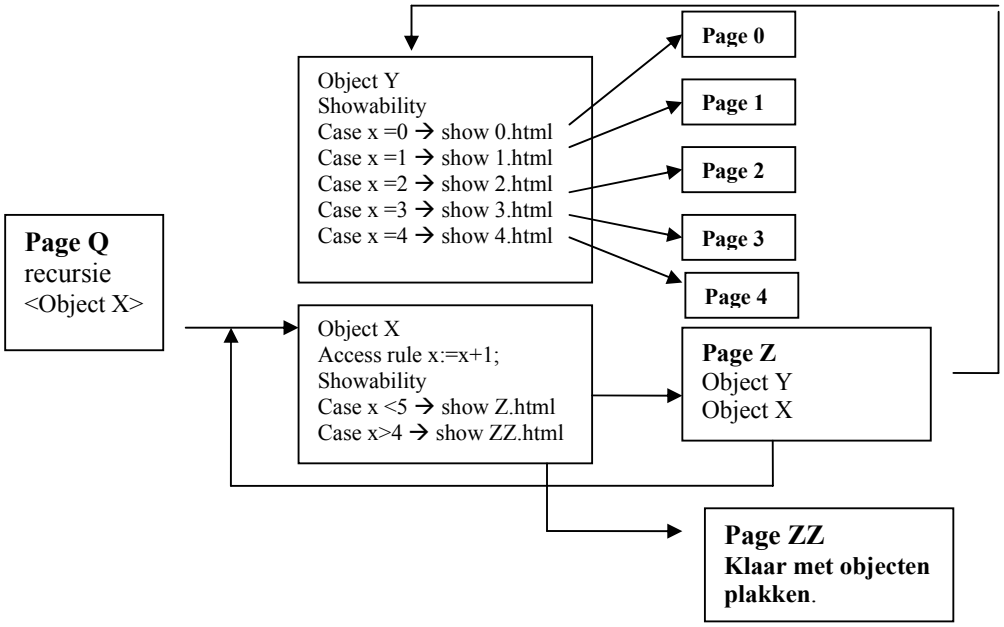
In een graaf ziet dit er als volgt uit:



Figuur 17 Document boom onderwijs applicatie Europa

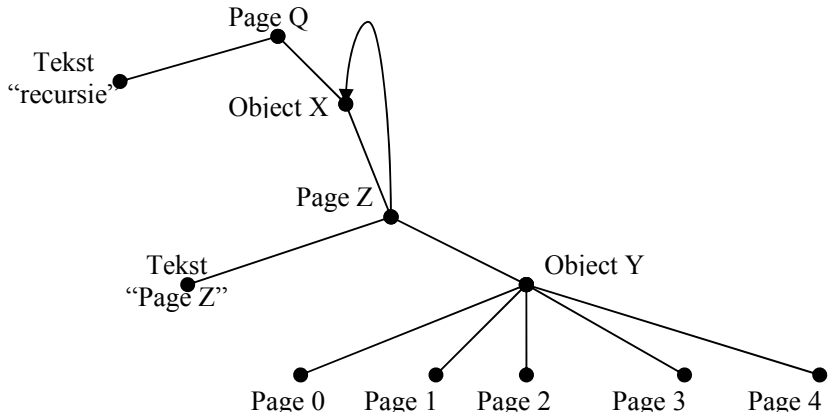
Dit proces herhaalt zich indien er geen maatregelen getroffen worden. De volgende paragraaf beschrijft welke maatregelen er getroffen kunnen worden om dit te voorkomen en hoe er gedetecteerd kan worden dat deze situatie ontstaat. Het niet afbreken van een recursieve structuur kan te wijten zijn aan het ontbreken van een variante functie. Indien er tijdens het opbouwen van de resultaatboom geen wijzigingen (in brede context) in het UM ontstaan, is een oneindige constructie het gevolg. Fragmenten blijven elkaar insluiten omdat er zoals bovenstaand voorbeeld toont geen uitweg in deze constructie is.

Het volgende voorbeeld is een voorbeeld van een recusieve applicatie, welke wel correct gebruik maakt van recursief insluiten van objecten. Dit, om aan te tonen dat er correct/zinnig gebruik gemaakt kan worden van recursieve fragmenten.



Figuur 18 Recursief fragment schema

Dit is ook met een graaf te visualiseren, analoog aan de methode besproken in 4.2.



Figuur 19 structuurgraaf recursief fragment schema

Een gebruiker vraagt aan het AHA! systeem pagina Q op. De engine vind vervolgens op pagina Q object X. Object X toont pagina Z indien de UM waarde x kleiner is dan 5 en verhoogt de UM waarde x met 1. Pagina Z toont object Y en vervolgens object X. Object Y retourneert afhankelijk van de waarde van x Page 0, page 1, pag e2, page 3 of page 4 als basisfragment. Dit herhaalt zich 5 keer (x is geïnitieerd met waarde 0). De tussentijds geretourneerde tekst wordt op een tijdelijke pagina geplakt. Deze tijdelijke pagina wordt aan het einde aan de gebruiker getoond. De vet gedrukte tekst, is de tekst die uiteindelijk aan de gebruiker getoond wordt.

Dit levert als resultaat op:

Resultaat pagina
Recursie **Page Z** **Page 0** **Page Z** **Page 1** **Page Z** **Page 2** **Page Z** **Page 3** **Page Z** **Page 4** **Page ZZ** **Klaar met objecten plakken.**

Het praktische nut van dit soort applicaties moet zich nog bewijzen. Te denken valt aan enkele malen hetzelfde object te plaatsen. Een top 10 van geschikte cd's zou een toepassing kunnen zijn. "Page 0" zou dan nummer 1 zijn etc. Echter "page 0" zou zelf moeten bepalen, welke cd voor die gebruiker het meest geschikt is. Dit zijn in AHA! query technisch ingewikkelde constructies. De volgende paragraaf beschrijft verschillende terminatie mogelijkheden welke binnen het AHA! systeem geïmplementeerd kunnen worden om oneindige recursie te detecteren en af te breken.

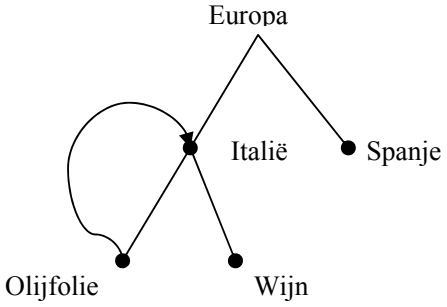
4.3. Terminatie bij recursieve objectstructuren

Uit het voorafgaande paragraaf is duidelijk geworden dat recursie een reeel gevaar voor het functioneren van een AHS kan zijn. Deze paragraaf beschrijft drie terminatie algoritmen om deze ongecontroleerde manier van pagina's genereren af te breken. De volgende drie methode zijn bestudeerd:

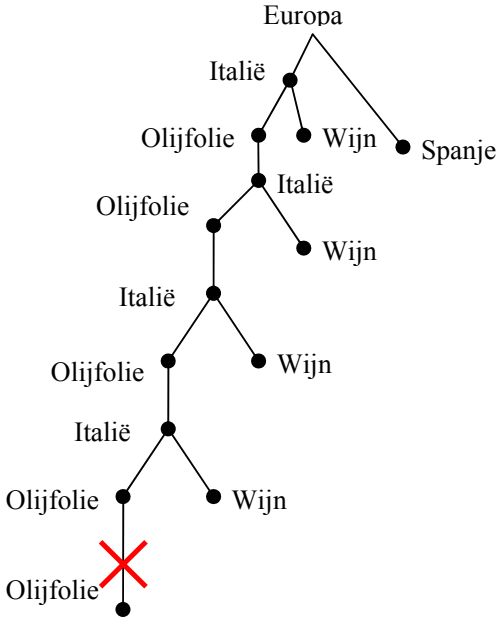
1. Beperken van diepte boom
2. Monotone UM mutaties
3. Patroonherkenning in UM mutaties

4.3.1. Restrictie boomdiepte

Een beperking op de diepte van de boom is de meest eenvoudige manier van het afbreken van mogelijke oneindige recursie. Er wordt gedefinieerd hoeveel fragmenten er maximaal in een boom aanwezig mogen zijn. Op dit moment is dit maximum, gedefinieerd op 500. Dit is formeel gezien niet de boomdiepte maar de boomgrootte. In het voorbeeld van Italië en olijfolie wordt na een aantal keer olijfolie en Italië ingevoegd te hebben, de recursie afgebroken en de pagina getoond. Indien er meer dan 500 fragmenten op een pagina geplaatst worden, is het in ieder geval dubieus of dit de bedoeling was. Deze manier is niet de meest correcte manier van recursie afbreken doch een zeer doeltreffende.



Figuur 20 structuurgraaf onderwijsapplicatie Europa



Figuur 21 afgebroken documentboom

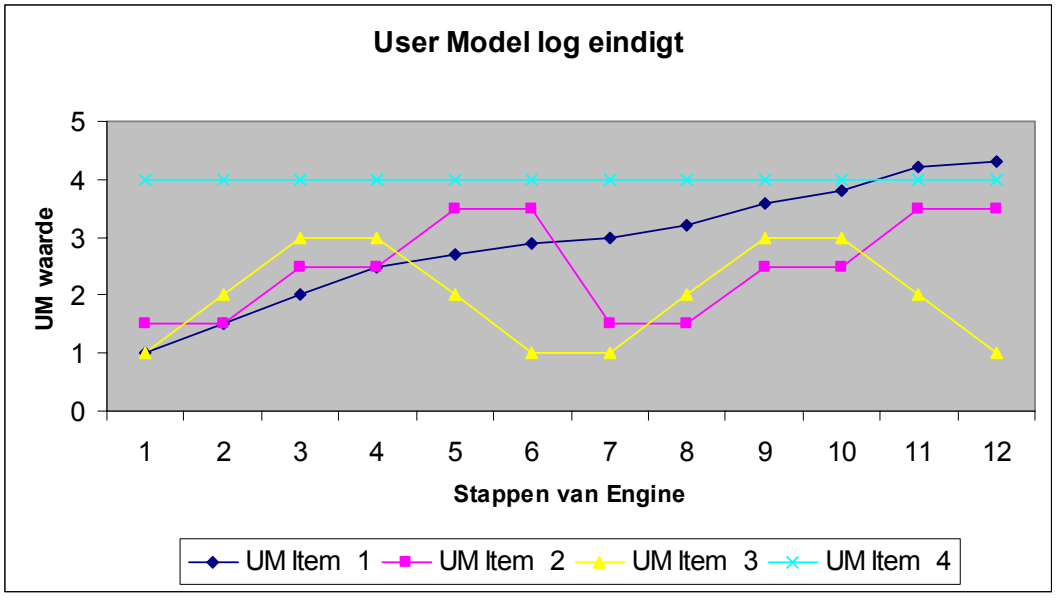
Figuur 19 en figuur 20 geven duidelijk weer waar de boom wordt afgebroken. Dit is echter alleen om het systeem te beschermen tegen dit soort ongewenste constructies. In de toekomst zouden dit soort constructies gedetecteerd moeten worden door auteursgereedschap. Dit is dus geen theoretisch correcte manier van cycle detectie en terminatie maar een botte bijl methode om het systeem tegen zichzelf te beschermen.

4.3.2. Monotone UM mutaties

De tweede methode van oneindige recursie herkennen, is de eis dat een UM mutatie slecht een kant op mag bewegen. Positief of negatief en niet alternerend. Indien een UM item niet een kant op beweegt kan de recursie beëindigd worden en de pagina gepresenteerd worden. Deze methode van recursie interceptie is een zeer beperkte. Het beperkt zowel de auteur in zijn vrijheid, evenals die van het systeem. Het is nu nog mogelijk om oneindig grote recursieve structuren te genereren welke niet door dit algoritme worden herkend en afgebroken. Deze methode is ook een niet aan te bevelen methode van recursie detectie en terminatie. Het is te zwak in het herkennen van recursie patronen.

4.3.3. Patroon herkenning

Zoals eerder gedefinieerd: Indien er tijdens het opbouwen van de resultaatboom geen wijzigingen (in brede context) in het UM ontstaan, is een oneindige boom het gevolg. Deze wijzingen moeten gezien worden als patroon. Indien een fragment het UM wijzigt en een ander fragment deze wijziging ongedaan maakt kan geconcludeerd worden dat er geen vooruitgang geboekt wordt in het behalen van een eindsituatie. Een variante functie moet afnemen of toenemen richting een terminatie toestand. Deze terminatietoestand kan zijn x<5 zoal in het recursieve voorbeeld (fig 18) getoond werd. Om deze methode van patroon herkenning te illustreren nemen we een User Model met 4 items, item 1 veranderd niet in een patroon en dus kan deze pagina correct zijn.



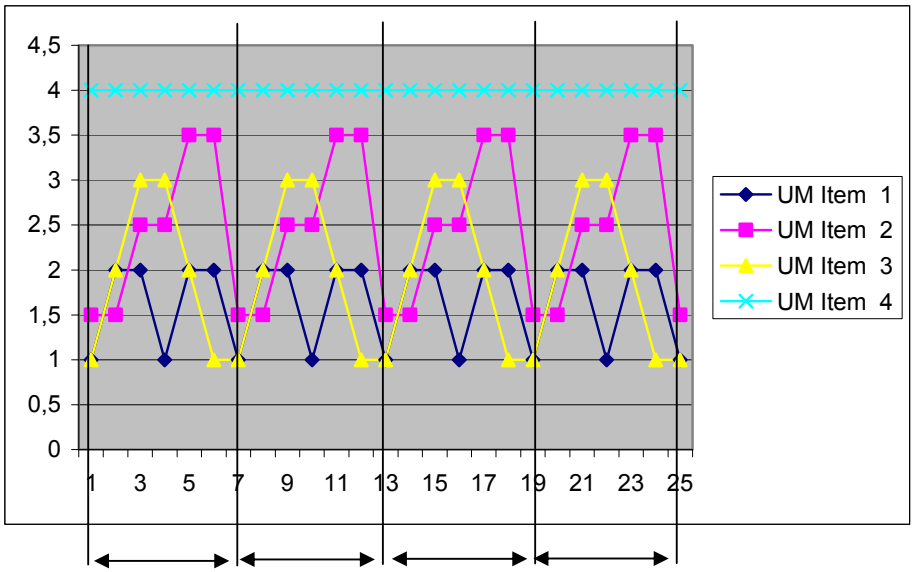
Figuur 22 terminatie van oneindige recursie met patroonherkenning

Nogmaals, deze gegenereerde pagina kan correct zijn. UM item 1 kan ook oneindig naar boven blijven lopen zonder een fragment structuur af te breken. Patroon herkenning is het zoeken naar een herhaling in UM mutaties. Per UM mutatie moet gekeken worden of de staat van het UM al eerder in die sessie (tijd dat een gebruiker is ingelogd) is voorgekomen. Indien dan de voorgestelde UM mutatie dezelfde is als in eerdere situatie kan deze UM mutatie geweigerd

worden en is er sprake van een patroon. Een bekent patroon is een sinus, indien de periode van een sinus bekent is, is de volgende periode ook bekend. Dit zal dan, hoeveel periodes er ook bekeken worden, niet veranderen.

Indien een pagina wordt opgebouwd en er tijdens de opbouw bij elk nieuw toegevoegd fragment (en bijbehorende UM mutaties) niets verandert, is dit een teken van oneindige recursie.

Een periode met UM mutaties duidt op ongecontroleerde recursie:



Figuur 23 terminatie van oneindige recursie met patroonherkenning II

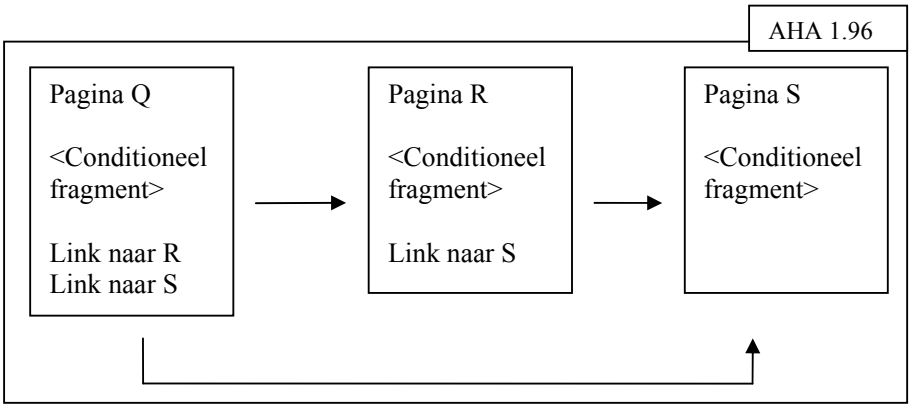
Beschouw de bovenstaande grafiek, tussen fragment 1 en 7 gebeuren er unieke UM mutaties. Vanaf positie 8 herhalen de UM mutaties zich in periode's van 7 stappen. Indien een patroon ontstaat over ALLE UM entries, dan en slechts dan is er sprake van een oneindige recursie. Er is immers niets variant meer. Het niet voorkomen van patronen is geen garantie voor het ontbreken van oneindige recursie zoals op de vorige pagina is uitgelegd. Zolang er geen patroon gedetecteerd wordt, wordt het window waardoor gekeken wordt om een patroon te herkennen groter.

4.3.4. conclusie terminatie

Om goede terminatie van oneindige fragmentstructuren af te breken cq herkennen zou er naast methode drie van recursie terminatie, altijd een controle op de resultaat boomgrootte moeten plaatsvinden. Indien de boom te groot wordt zou de boom afgebroken moeten worden. (methode 1). Een combinatie van verschillende methoden is daarom aan te bevelen. Methode drie als eerste wapen om de engine te beschermen tegen oneindige recursie. Indien methode drie geen patroon herkent blokkeert methode 1 de engine na 500 fragmenten invoegen. Methode 1 is dus het laatste redmiddel voor de engine om niet oneindig door te lopen (met als technisch gevolg een stack overflow, welke de server automatisch afsluit of een error pagina aan de gebruiker toont).

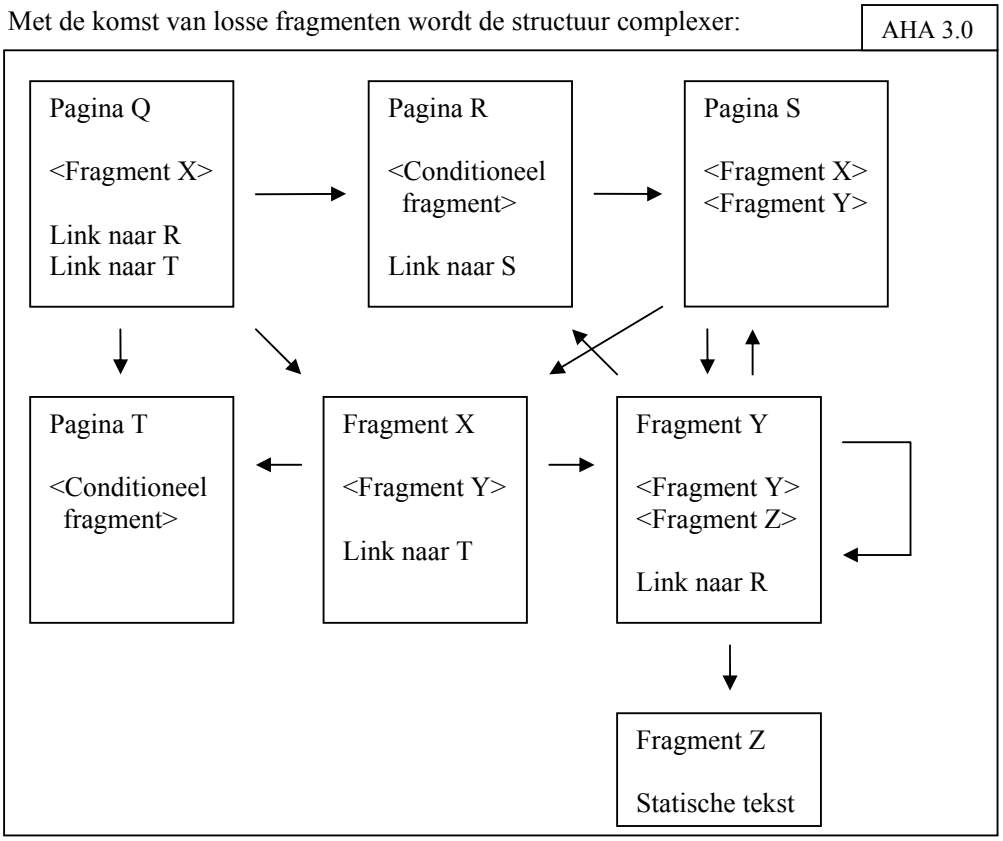
4.4. Verminderd overzicht AHA! applicatie

De derde consequentie van deze losse fragmentstructuur is het verminderde overzicht op het systeem. In AHA! 1.96 werden alléén pagina's adaptief getoond, er was een relatie tussen de pagina's door middel van links.
De figuren hieronder tonen de relaties tussen fragmenten en pagina's. Een pijl geeft aan dat er een relatie is tussen beide. Een pijl tussen 2 pagina's geeft bijvoorbeeld aan dat er middels een link van de ene naar de andere pagina genavigeerd kan worden. Een pijl tussen een pagina en een fragment geeft aan dat een pagina een fragment bevat. Indien van een fragment naar een ander fragment een pijl gaat, geeft dit aan dat dit fragment het andere fragment insluit.



Figuur 24 complexiteit van structuur in AHA 1.96

Met de komst van losse fragmenten wordt de structuur complexer:



Figuur 25 complexiteit van structuur in AHA! 3.0.

De link structuur van een website presenteren kunnen de meeste Web ontwikkelomgevingen. Deze omgeving genereert een graaf waarin alle links worden getoond door een lijn tussen twee knopen. Indien ook de structuur tussen fragmenten en pagina's en fragmenten met fragmenten gevisualiseerd moet worden is dit een complexer probleem. De vakgroep "Computer Graphics" heeft ervaring met het 3D visualiseren van complexe graven. Een AHA! applicatie visualiseren zou een nuttige toevoeging voor een auteur kunnen zijn. Zie [HEESSEN2002] voor meer informatie over 3D visualiseren van grafen met cycles.

4.5. AHA! MEETS SMIL

Nu het domein model los is gekoppeld van de pagina's, is de opbouw van een pagina minder relevant. Immers de fragmenten zijn uit de pagina's verhuisd naar een eigen data opslag en de adaptatie regels zijn verhuisd naar het DM. Aangezien AHA! 3.0 standaard XML (eXtensible Markup Language) pagina's verwerkt, maakt het niet uit in welke taal deze XML pagina's geschreven zijn. Op dit moment zijn een groot aantal documentstructuren in XML opgemaakt. XML is een in 1998 door het W3C gespecificeerde taal. Doordat de object structuur een standaard tag is, kan deze gebruikt worden in ieder XML document. SMIL ofwel **S**ynchronized **M**ultimedia **I**ntegration **L**anguage (spreek uit smile) is een taal welke XML gebruikt als structuur. SMIL specificeert interactieve multimedia voor het Web. SMIL pagina's worden afgespeeld met speciale SMIL players. RealOne player van Real is een SMIL 2.0 player. In een SMIL document worden eigenschappen van multimedia elementen beschreven. Deze eigenschappen kunnen bijvoorbeeld zijn:

- Speel videoclip1.avi af na 5 seconden
- Speel geluidfragment2.wav af na 2 seconden tegelijkertijd met videoclip3.avi
- Indien het scherm 100 * 100 pixels bevat, toon picture_small.jpg, anders picture_big.jpg

Deze eigenschappen worden in het XML formaat beschreven.
Om adaptieve elementen toe te voegen in een SMIL presentatie, kan er een <object> tag worden geplaatst op een SMIL pagina. Echter de DTD (Data Type Definition) van SMIL document heeft geen ondersteuning voor de <object> tag. Een DTD is een verzameling spelregels waaraan een XML document moet voldoen. Aan een XML document zijn een aantal eisen te stellen: Goede opmaak van het document (well formedness), syntactisch correctheid en semantisch correctheid. Een document is wellformed als de basis-structuur van een document correct is. Of een document syntactisch correct is, wordt gecontroleerd middels het bijbehorende DTD. Hierin wordt, bijvoorbeeld, gedefinieerd, dat na het opslaan van een naam, de woonplaats opgeslagen moet worden. Verder staat in een DTD hoe deze naam in een XML document geïdentificeerd moet worden. De DTD van SMIL ondersteunt geen gebruik van de Object tag. Daarom is in SMIL een andere tag gebruikt om adaptieve fragmenten te herkennen en identificeren. Dit is de REF tag geworden. De REF tag is een SMIL tag welke naast de Object tag door de AE herkend en verwerkt wordt.
De relatie tussen een ref tag en de object tag is de volgende:

```
<ref        src="ahacook.smilobject"  type="aha/text">   </ref>
                      ↕                       ↕
<object     name=”ahacook.smilobject" type="aha/text">   </object>
```

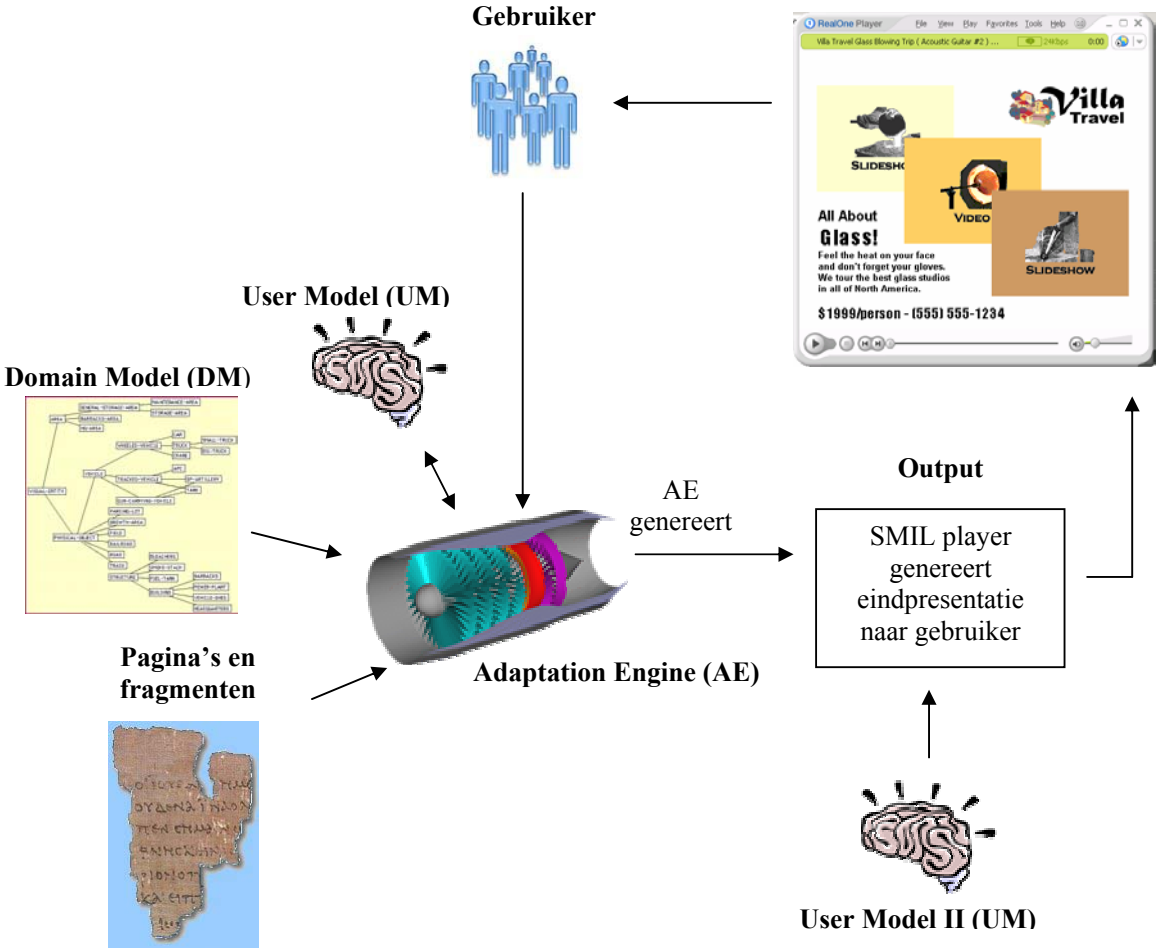
Een voorbeeld SMIL document staat hieronder:

```
1      <!DOCTYPE smil SYSTEM "SMIL10.dtd">
2      <smil xmlns="http://www.w3.org/2001/SMIL20/Language">
3      <head>
4        <layout>
5             <region id="main" top="10" left="20" height="200" width="200" z-6
               index="3"></region>
6        </layout>
7      </head>
8        <body>
9         <seq repeat="1">
10        <ref dur="4s" fill="remove" repeat="1" src="textfile.txt" > </ref>
11        <ref dur="4s" fill="remove" repeat="1" src="textfile1.txt" > </ref>
12        <ref   dur="4s"   fill="remove"   repeat="1"   src="ahacook.smilobject"
          type="aha/text"></ref>
13        </seq>
14       </body>
15     </smil>
```

Dit voorbeeld laat de structuur van een SMIL document zien. Op regel 10 tot 13 worden fragmenten getoond. Het eerste fragment is een op het systeem aanwezig tekstbestand textfile.txt. Deze wordt gedurende 4 seconde getoond aan de gebruiker. Regel 12 toont een adaptief fragment. Deze wordt door de AHA! engine verwerkt. De AHA! engine substitueert hiervoor een ander fragment. Dit fragment kan er bijvoorbeeld uitzien als regel 11.
Het bovenstaande voorbeeld laat slechts een heel klein beperkt deel van SMIL zien. SMIL is een ongelofelijk grote taal welke vele mogelijkheden bevat [HOSCHKA1998]. Het is niet de bedoeling om een volledig overzicht van SMIL te geven. Dat valt buiten de scope van dit verslag. Eveneens is de term XML en DTD vrij beperkt uitgelegd. Indien een lezer hier en over SMIL meer informatie wenst, in de referentielijst staat een lijst met goede boeken over XML [POEL2003] en SMIL.

4.5.1. Conclusie AHA MEETS SMIL

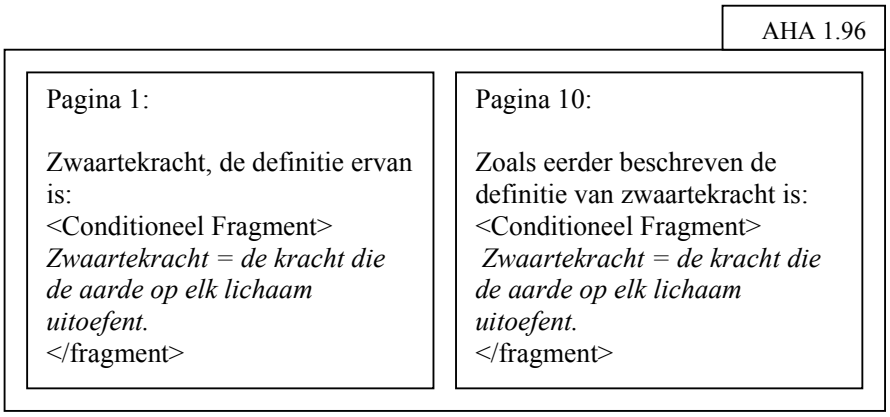
Doordat SMIL een XML georiënteerde taal is, welke middels een SMIL browsers met een webserver communiceert, is het mogelijk adaptieve onderdelen aan te brengen in een SMIL document. Dit is universeel bij elke op XML gebaseerde taal welke met een webserver communiceert. De taal SMIL biedt adaptiviteit aan de gebruiker in de vorm van omgevingsadaptatie. Wat voor een netwerk heeft de gebruiker? Welke schermresolutie heeft een gebruiker? Deze antwoorden heeft een SMIL player bij het bepalen van de presentatie voorhanden. Echter, permanente opslag, over welk fragment/pagina een gebruiker gezien heeft en hoe vaak hij deze pagina's/fragmenten gezien heeft, ontbreken in de taal SMIL (en deze is hiervoor ook niet bedoeld). Het AHA systeem vult het adaptief gedrag van SMIL perfect aan.
Dit levert het volgende systeemoverzicht op:



Figuur 26 Structuur overzicht van een AHS in samenwerking met SMIL

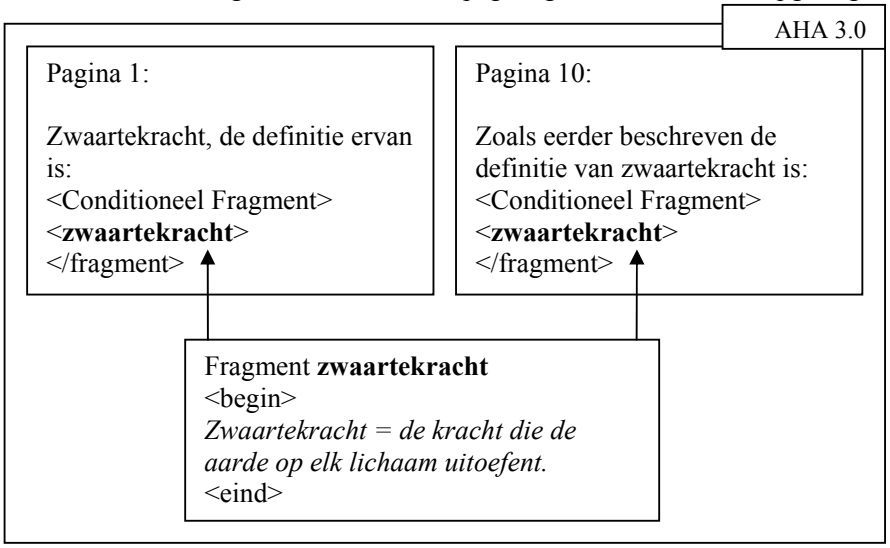
4.6. Hergebruik fragmenten in AHS

De vijfde onderzochte consequentie is dat in AHA 1.96 de fragment content wordt opgeslagen op een pagina. Indien vaker hetzelfde fragment op verschillende pagina's gebruikt wordt, heeft dit tot gevolg dat de content van een fragment onnodig op verschillende pagina's wordt opgeslagen. Door de fragment content uit de pagina's te halen, behoort redundant opslaan van fragment content tot het verleden. Dit kan bij grote informatiesystemen een aardige besparing aan gegevensopslag betekenen. Een ander groot voordeel van deze manier van werken is onderhoudbaarheid. Indien een definitie die als fragment is opgeslagen in het AHA! systeem gewijzigd moet worden, hoeft dit slechts een keer te gebeuren.



Figuur 27 Fragment opslag AHA! 1.96

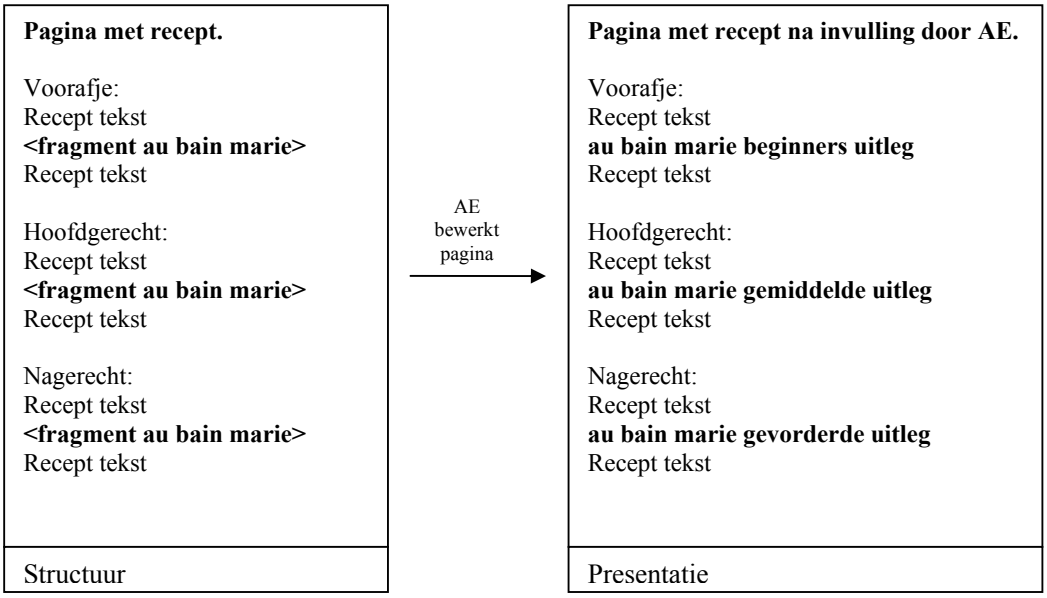
In AHA 3.0 is de fragment content uit de pagina gehaald en centraal opgeslagen.



Figuur 28 Fragment opslag AHA! 3.0

4.7. Autonoom gedrag van een fragment

De laatste consequentie welke onderzocht is handelt over het autonoom gedrag van een fragment. Een fragment in AHA! 3.0. is zelf in staat om het user model te wijzigen. Indien een pagina een AHA! 3.0 fragment bevat, wordt hiervan het concept opgehaald en de bijbehorende regels uitgevoerd. Deze regels kunnen het User Model aanpassen. De kennis van een onderwerp kan bijvoorbeeld verhoogd worden. Ieder fragment fungeert als een black box. Een fragment bepaalt volkomen autonoom op welke manier hij zichzelf wil presenteren. In AHA! 1.96 was het niet mogelijk om drie dezelfde fragmenten op een pagina te plaatsen en deze drie fragmenten alle een verschillende presentatie te geven. Ik illustreer dit aan de hand van een voorbeeld:
Stel op een kookpagina wordt een recept uitgelegd. Dit recept gebruikt drie maal de term au bain marie. In het vervaardigen van het voorafje het hoofdgerecht en het nagerecht wordt deze kookterm gebruikt. Het fragment au bain marie kan zich op drie manieren presenteren, beginners gemiddeld en ervaren kennis over de kookterm. Iedere keer dat de gebruiker de kookterm gezien heeft schuift de gebruiker een niveau op en verandert de presentatie van het fragment. Structureel zou dit er als volgt uitzien:

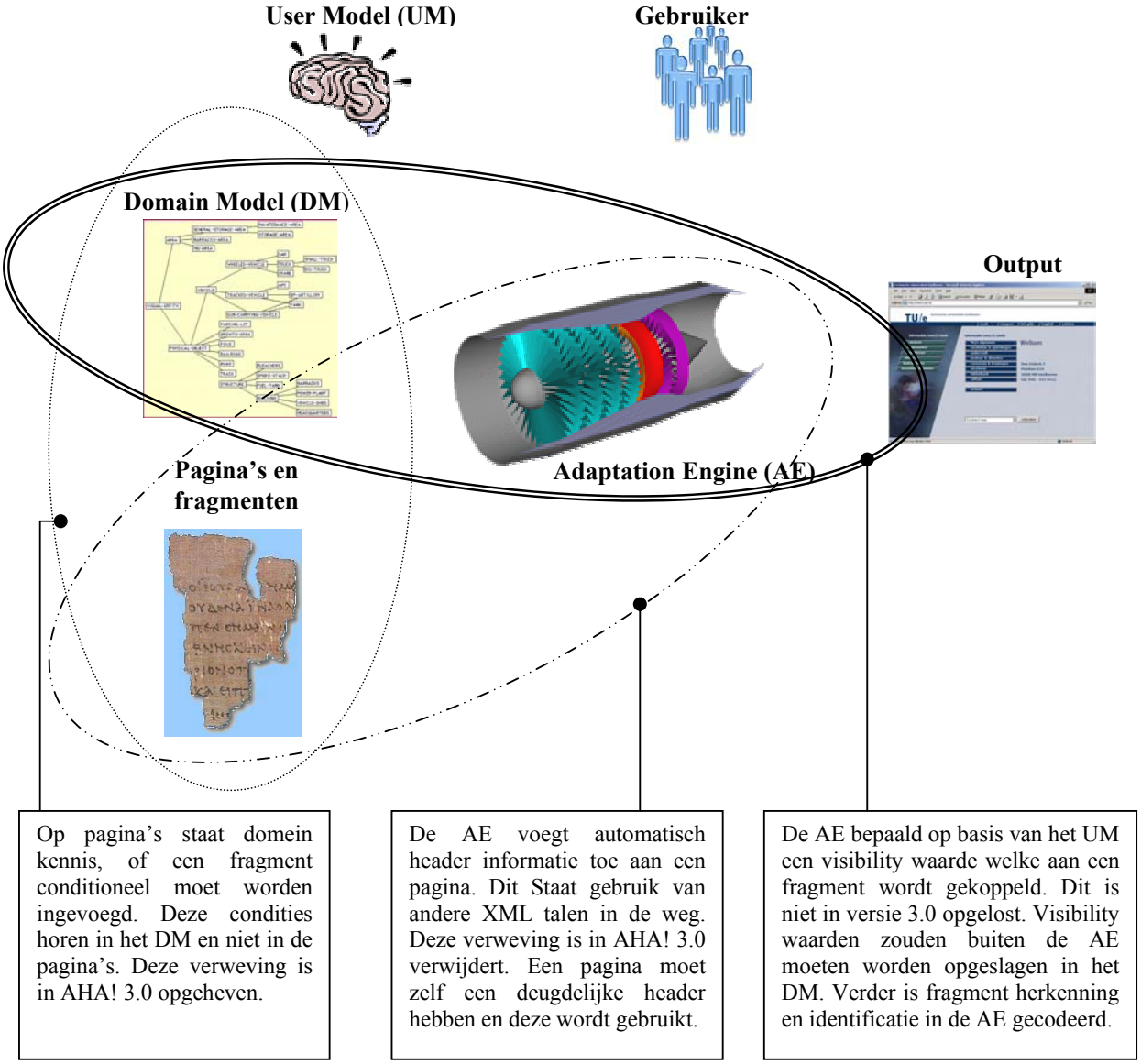


Figuur 29 autonoom fragment gedrag

In AHA! 1.96 is het niet mogelijk met identieke fragmenten zo een structuur te maken.

4.8. Modulaire opbouw AHA! systeem

Het AHA! 1.96 heeft zoals in dit hoofdstuk beschreven veel koppelingen tussen DM en content liggen. Om AHA! zo modulair mogelijk te maken geeft onderstaand figuur iedere verweving tussen de diverse componenten in AHA! 1.96. Er is bij iedere verweving beschreven hoe deze indien mogelijk ontrafelt is.



Twee van de drie verwevingen zijn opgeheven met de nieuwe structuur zoals beschreven in deze scriptie. De verweving welke overblijft, is minimaal te noemen. Er wordt op basis van het UM een showability waarde gegenereerd. Deze waarde zou normaal opgezocht moeten worden in het DM, maar wordt nu geclassificeerd. Indien de waarde tussen 0 en 9 zit wordt er een showability waarde van 0 gesubstitueerd, tussen 10 en 19 een waarde van 10 etc. Strikt genomen zou deze conversie in het DM gedefinieerd moeten zijn. Dit wordt nu eigenhandig door de AE uitgevoerd. De AE behoort geen DM kennis te bezitten.

5. Technische consequenties

5.1. Inleiding

De nieuw ingevoerde structuur heeft naast theoretische ook technische gevolgen. Dit hoofdstuk beschrijft kort welke problemen/veranderingen er ontstaan zijn en hoe hiermee om te gaan.
De volgende veranderingen worden beschreven:

1. performance problemen bij grote structuren
2. verandering in concept definitie
3. verandering in Auteurs gereedschap

5.2. Performance problemen

Door het toestaan van recursieve fragment structuren zijn complexe applicaties te ontwikkelen welke veel tijd van het systeem vergen. Om een indicatie te geven van tijd is de volgende testcase uitgevoerd:

Maximaal aantal fragmenten:	500
Test systeem:	Intel Celeron 500 met 192 Mb geheugen
Tijd	8 Minuten verwerking
Fragmentgrootte:	Enkele regels acsii tekst
Recursiestructuur:	Object X sluit object Y in welke object X insluit etc..

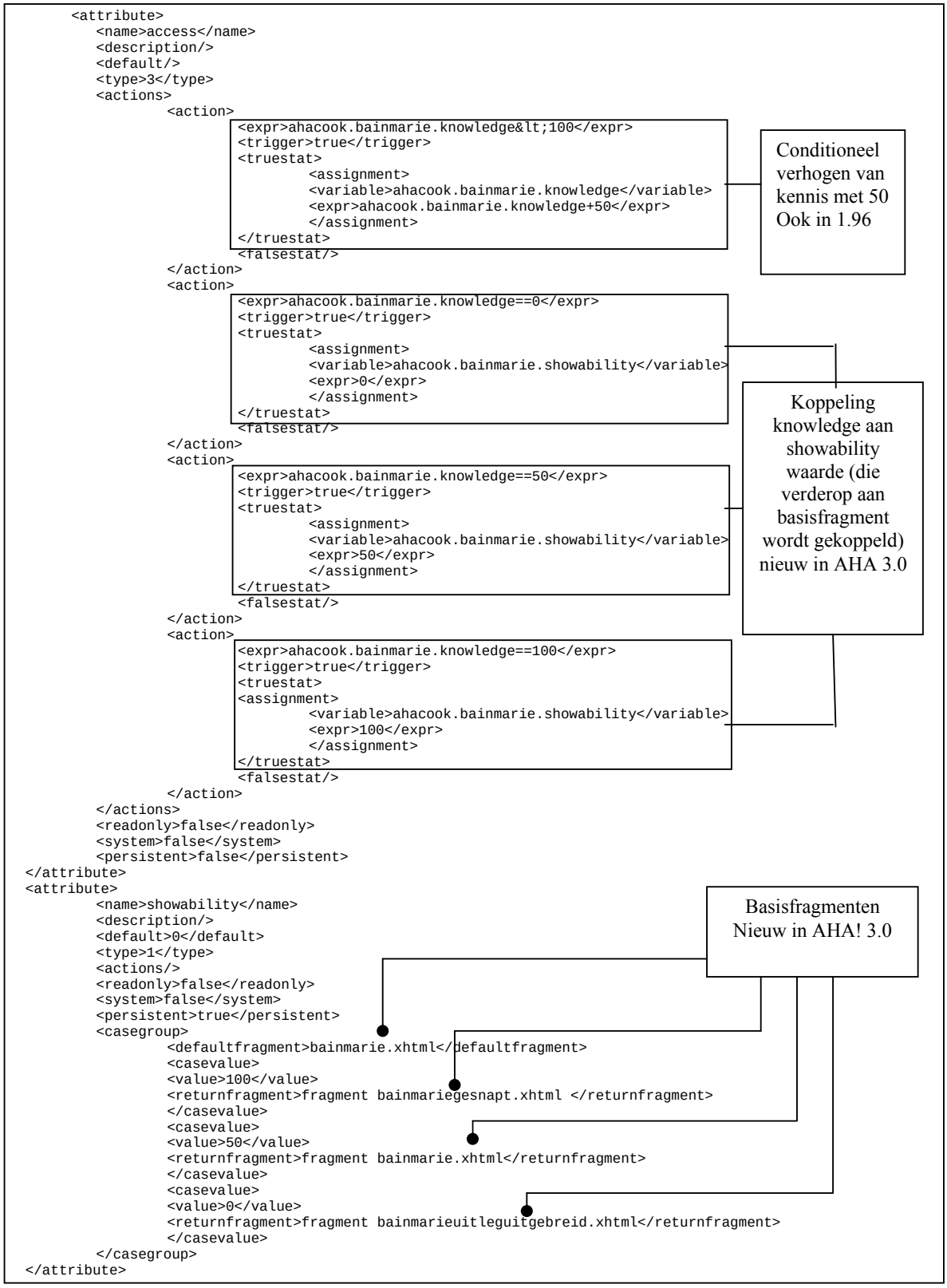
Als bijverschijnsel kunnen tijdens deze tijd andere gebruikers geen gebruik maken van het AHA systeem. Dit druist tegen het principe van een webserver welke meerdere aanvragen parallel verwerkt in. De grootte van het fragment had minimale invloed op de tijd welke het systeem nodig had om de pagina te genereren. De grens van 500 fragmenten is in het systeem aan te geven (is een variabele in de XMLHandler.java file). Indien dit op meer dan 500 werd gezet crashte de server. Deze crash kon op twee manieren gebeuren. Bij een oudere tomcat versie (1.0_01) gaf het een stack overflow terug aan de gebruiker. Een nieuwere versie (1.1.24) stopte de tomcat server waarna geen pagina aanvragen meer gedaan mochten worden. Dat dit ook bij een snelle server reëel is, wordt middels de volgende reële case duidelijk gemaakt:

Gegevens: Aantal gebruiker simultaan aanwezig op systeem:	15
Gemiddeld te genereren aantal fragmenten per pagina:	6
Gemiddelde bezoektijd van een pagina door gebruiker:	2 minuten
Capaciteit van server uitgaande van bovenstaande case:	1 fragment/seconde
Uitkomst: Dit levert de volgende schatting van resultaten op:	
Gemiddeld aantal aanvragen per gebruiker per seconde:	0,05
Gemiddelde load (in procent) van het systeem:	75%

Bij 20 gebruikers is het systeem 100% belast en ontstaat er een wacht rij welke niet meer oplost. (indien gemiddeld 20 gebruikers actief zijn). Indien de capaciteit vertienvoudigd wordt levert dit een capaciteit op van 200 gebruikers. **Performance is een issue!** De hierboven berekende waarden zouden getest moeten worden op correctheid. Het is een schatting welke reëel lijkt te zijn. Een beperking van het aantal gebruikers kan ook een oplossing bieden. Indien uit statistische gegevens blijkt hoeveel gebruikers wanneer ingelogd zijn, en hoeveel pagina's deze opvragen, zou met deze heuristische gegevens de toegang tot het systeem voor nieuwe gebruikers beperkt kunnen worden.

5.3. Verandering in de concept definitie (domein model)

In AHA! 1.96 werden fragmenten middels een <if> constructie al dan niet op een pagina ingesloten. In AHA! 3.0 is gekozen om ieder fragment net als een pagina aan een concept te koppelen. Hiervoor moet het huidige pagina concept worden uitgebreid met een onderdeel welke beschrijft welke conditie aan welk basisfragment gekoppeld moet worden. Omdat de presentatie van een fragment los staat van bijvoorbeeld de kennis van een bepaald item is er gekozen om een nieuw attribuut showability te introduceren. De showability wordt in het access gedeelte van een concept bepaald. Daarna wordt middels een case constructie bepaald welke showability waarde er aan welk basisfragment moet worden gekoppeld. Op de volgende pagina staat een concept welke bij au bain marie hoort. Er is een extra laag toegevoegd




```
      <attribute>
         <name>access</name>
         <description/>
         <default/>
         <type>3</type>
         <actions>
                 <action>
                         <expr>ahacook.bainmarie.knowledge&lt;100</expr>
                         <trigger>true</trigger>
                         <truestat>
                                 <assignment>
                                 <variable>ahacook.bainmarie.knowledge</variable>
                                 <expr>ahacook.bainmarie.knowledge+50</expr>
                                 </assignment>
                         </truestat>
                         <falsestat/>
                 </action>
                 <action>
                         <expr>ahacook.bainmarie.knowledge==0</expr>
                         <trigger>true</trigger>
                         <truestat>
                                 <assignment>
                                 <variable>ahacook.bainmarie.showability</variable>
                                 <expr>0</expr>
                                 </assignment>
                         </truestat>
                         <falsestat/>
                 </action>
                 <action>
                         <expr>ahacook.bainmarie.knowledge==50</expr>
                         <trigger>true</trigger>
                         <truestat>
                                 <assignment>
                                 <variable>ahacook.bainmarie.showability</variable>
                                 <expr>50</expr>
                                 </assignment>
                         </truestat>
                         <falsestat/>
                 </action>
                 <action>
                         <expr>ahacook.bainmarie.knowledge==100</expr>
                         <trigger>true</trigger>
                         <truestat>
                         <assignment>
                                 <variable>ahacook.bainmarie.showability</variable>
                                 <expr>100</expr>
                                 </assignment>
                         </truestat>
                         <falsestat/>
                 </action>
         </actions>
         <readonly>false</readonly>
         <system>false</system>
         <persistent>false</persistent>
</attribute>
<attribute>
         <name>showability</name>
         <description/>
         <default>0</default>
         <type>1</type>
         <actions/>
         <readonly>false</readonly>
         <system>false</system>
         <persistent>true</persistent>
         <casegroup>
                 <defaultfragment>bainmarie.xhtml</defaultfragment>
                 <casevalue>
                 <value>100</value>
                 <returnfragment>fragment bainmariegesnapt.xhtml </returnfragment>
                 </casevalue>
                 <casevalue>
                 <value>50</value>
                 <returnfragment>fragment bainmarie.xhtml</returnfragment>
                 </casevalue>
                 <casevalue>
                 <value>0</value>
                 <returnfragment>fragment bainmarieuitleguitgebreid.xhtml</returnfragment>
                 </casevalue>
         </casegroup>
</attribute>
```

Figuur 30 Concept wijzigingen 1.96 → 3.0

In de DTD zijn de veranderingen zwart gemarkeerd:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT concept (name, description, expr, attributes, resource?)>
<!ELEMENT attributes (attribute)*>
<!ELEMENT attribute (name, description, default, type, actions, readonly, system,
persistent, stable?, stable_expr?,casegroup?)>
<!ELEMENT actions (action)*>
<!ELEMENT action (expr, trigger, truestat, falsestat)>
<!ELEMENT truestat (assignment)*>
<!ELEMENT falsestat (assignment)*>
<!ELEMENT assignment (variable, expr)>
<!ELEMENT casegroup (defaultfragment, casevalue*)>
<!ELEMENT casevalue (value,returnfragment)>
<!ELEMENT defaultfragment (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT returnfragment (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT default (#PCDATA)>
<!ELEMENT expr (#PCDATA)>
<!ELEMENT resource (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT readonly (#PCDATA)>
<!ELEMENT system (#PCDATA)>
<!ELEMENT persistent (#PCDATA)>
<!ELEMENT stable (#PCDATA)>
<!ELEMENT stable_expr (#PCDATA)>
<!ELEMENT trigger (#PCDATA)>
<!ELEMENT variable (#PCDATA)>
```

Conceptueel werkt het systeem als volgt: Een fragment wordt benaderd. Middels de access rules wordt de showability waarde bepaald. Vervolgens wordt deze showability waarde gebruikt bij het bepalen van het returnfragment (basisfragment welke in de pagina wordt geplakt). Het return-frament is een pointer naar een basisfragment welke een los opgeslagen fragment is.

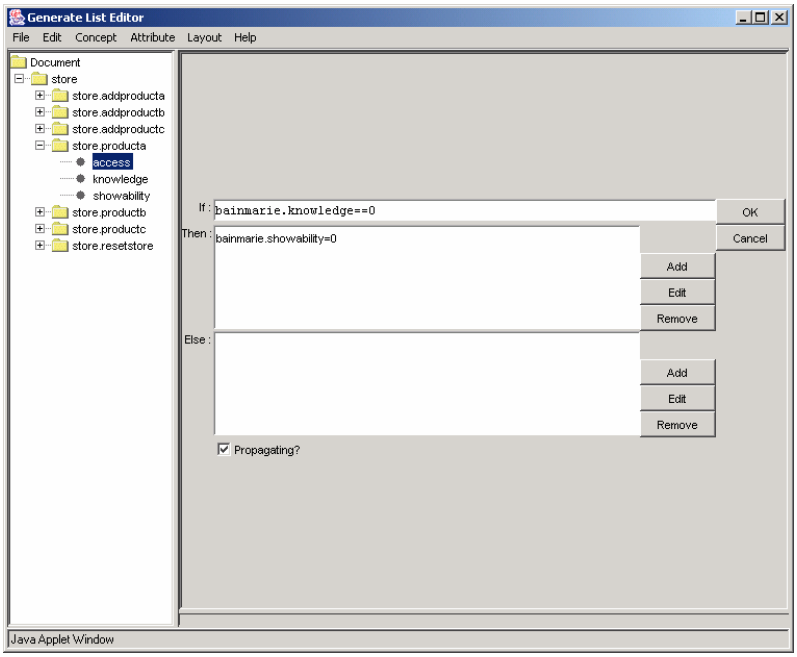
5.4. Verandering auteurstools

Door het veranderen van de concept structuur (5.3) moeten ook de auteurs tools deze nieuwe concepten kunnen genereren. Op dit moment wordt er binnen het AHA! systeem met twee auteurs gereedschappen gewerkt:

- Generate List Editor ook wel de Concept editor genoemd [stageverslag RUITER2001]
- Graph Author [afstudeerverslag ROUSSEAU2003]

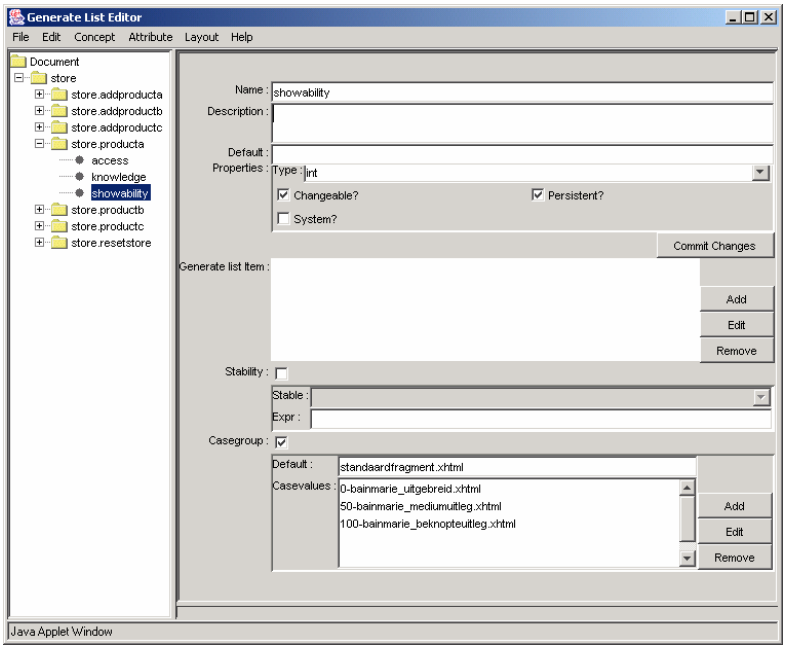
Beide auteursomgevingen zijn aangepast door B. Berden [afstudeerverslag BERDEN2003] tijdens mijn en zijn afstudeerperiode. De technische wijzigingen in de software laat ik buiten beschouwing. De focus zal vooral liggen in de functionele veranderingen die zijn gemaakt aan de software om de nieuwe fragment structuren te ondersteunen. Tijdens de ontwikkeling van het auteursgereedschap is er constant goed overlegd en getest of het ontwikkelde aansloot bij de wensen van de auteurs. De schrijver stelde zich op als tester en gebruiker.

De concept list editor is aangepast in de lijn van het product. Eerst wordt zoals in §5.3. beschreven, de showability waarde bepaald bij het benaderen van het fragment (zie Figuur 31).



Figuur 31 showability bepalen in Concept List Editor

Vervolgens wordt deze showability waarde gekoppeld aan een basisfragment (zie Figuur 32).

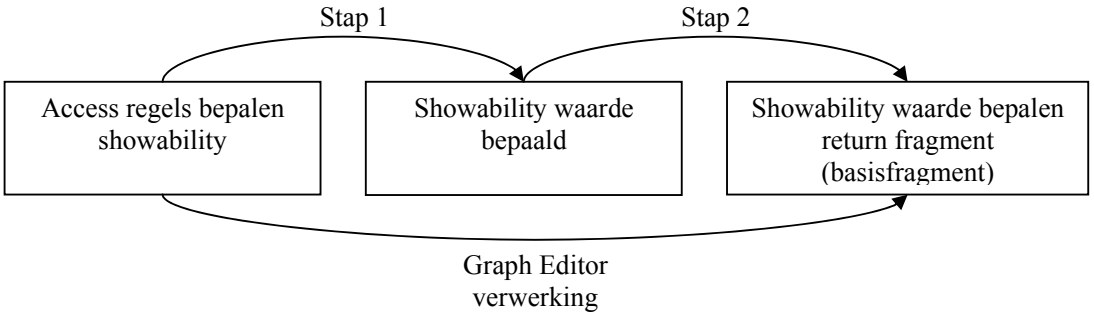


Figuur 32 basisfragmenten koppelen aan showability in concept list editor

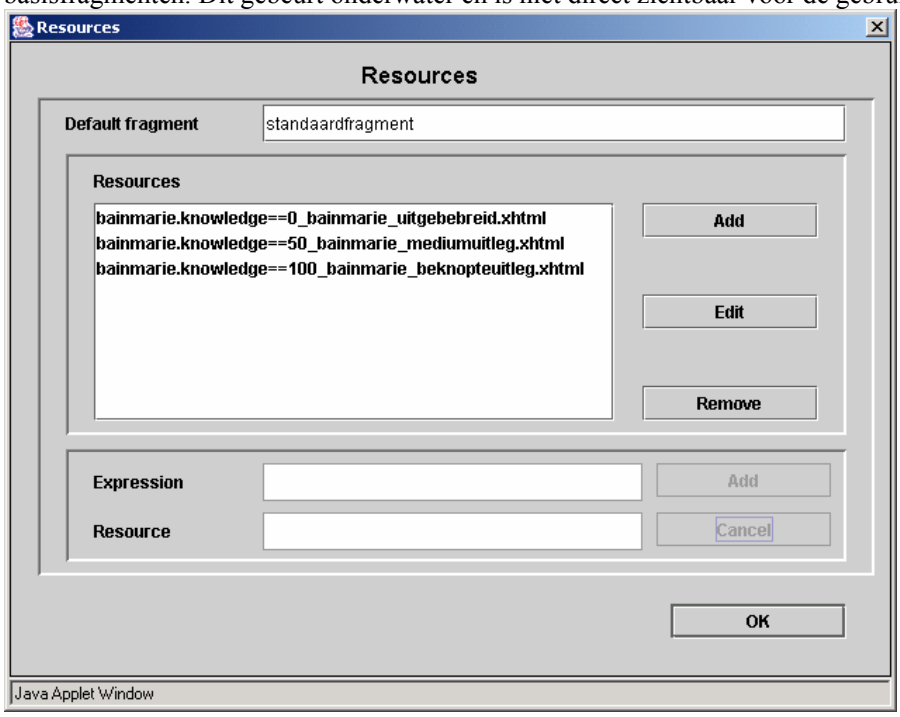
Graph author is het 2de tool welke is aangepast. Graph Author is in beginsel niet bedoeld om op laag niveau concept concept eigenschappen aan te passen. Het is gemaakt om relaties tussen concepten aan te geven. Aangezien de gebruiker in de meeste gevallen wil dat de fragment keuze bepaald wordt op het moment dat het fragment ingevoegd wordt, is een stap uit het proces verdwenen. In Generate List Editor worden steeds dezelfde stappen ondernomen:

1. Access regels zetten, op basis van een UM toestand, de showability waarde
2. Showability waarde wordt gekoppeld aan een basisfragment

Dit pad kan voor een gebruiker ingekort worden. In Graph Author hebben we ervoor gekozen om deze meestal onnodige tussenstap uit de procedure weg te laten en op basis van een toestand van het UM een basisfragment te koppelen. Schematisch:



In de Figuur 33 is de oplosmethode gevisualiseerd. Er is aan een expressie een fragment gekoppeld. Graph Author genereert intern de showability waarden en koppelt deze aan basisfragmenten. Dit gebeurt onderwater en is niet direct zichtbaar voor de gebruiker.



Figuur 33 Graph author fragment toewijzing

5.5. Verandering in de AE

5.5.1. Inleiding

Om de nieuw gedefinieerde structuren te kunnen gebruiken welke in het DM en UM zijn aangebracht, moesten enkele componenten van de AE gewijzigd worden. De engine communiceert met zowel het DM als het UM. Nieuwe velden moeten gedefinieerd worden om de engine de nieuwe structuur te laten begrijpen. Een overzicht van alle veranderde componenten en het bijbehorende functionaliteit is in onderstaande tabel weergegeven:

Componentnaam	Functionaliteit
XMLHandler.java	**Object en Ref tag herkenning en afhandeling (H3)**
	UM lock principe (H6) (1)
	Visibility van fragmenten (H2)
XMLConceptDB.java	**Object en Ref tag afhandeling (H3) (structuur beschrijving)**
AHAmime.java	**Herkenning SMIL documenten (H5)**
Case.java	**Object en Ref tag afhandeling (H3) (structuur beschrijving)**

(1) Het UM locken ofwel afsluiten voor mutaties is noodzakelijk om correcte afhandeling van parallelle aanvragen te garanderen. Hoofdstuk 6 zal dieper ingaan op deze problematiek. Het hoort niet specifiek bij de opdrachtdoelstelling maar was wel een noodzakelijke uitbreiding om correctheid te garanderen.

Hoe deze componenten zijn veranderd wordt uitgelegd in hoofdstuk 6 “Implementatie techniek”.

6. Implementatie techniek

6.1. Inleiding

Dit hoofdstuk beschrijft hoe de fragment object structuur geïmplementeerd is. De volgende implementatie onderdelen worden beschreven in dit hoofdstuk:

1. Object structuur herkenning en afhandeling in de XMLHandler
2. Terminatie garanderen door afbreken te grote structuren
3. Correct parallellisme garanderen door UM lock

Deze punten worden een voor een beschreven in dit hoofdstuk. In de bijlage(n) staan enkele software schema's, hierna wordt verwezen in dit hoofdstuk. Punt drie dient nadere toelichting; doordat auteurs pagina's kunnen construeren waar min of meer tegelijkertijd fragmenten aan het systeem worden aangevraagd (Frames in HTML) kunnen deze fragmenten elkaar beïnvloeden en correcte werking van het systeem in de weg staan. Daarom is dit probleem ook opgelost. §6.4 beschrijft dit probleem uitvoerig.

6.2. Object structuur herkenning en verwerking

Om de in hoofdstuk 3 gekozen <object> tag te herkennen en verwerken, moest de AHA Engine worden aangepast. De AHA engine bekijkt de aanvraag van de gebruiker en bepaalt wat voor een soort pagina er wordt aangevraagd. Voor ieder type aanvraag heeft de AHA engine een Handler beschikbaar. Mijn afstudeeropdracht heeft zich primair gefocust op XML aanvragen. Vandaar dat de XML-handler aangepast is in de AHA engine. De XMLHandler krijgt van de AHA engine een volledig X(HT)ML pagina. Deze wordt vervolgens omgezet naar een boom. Dit kan alleen indien het document een well-formed XML document is. Een well-formed XML document betekent dat het een bepaalde structuur heeft. Iedere knoop wordt in een XML document gestart door de knoop naam tussen <> te zetten en beeindigd door een "/" voor de knoopnaam te plaatsen en de knoop naam weer tussen <> te zetten. Knopen mogen in elkaar voorkomen maar verweven van knopen mag niet. Onderstaand voorbeeld geeft een correcte en niet correcte XML structuur weer:

Correcte XML structuur (well-formed)

```
<a>
      <b>
            <c>
            </c>
      </b>
</a>
```

Niet correcte XML structuur (well-formed)

```
<a>
      <b>
            <c>
      </b>
            </c>
</a>
```

Figuur 34 Well formed XML document

Deze boom wordt doorlopen door de XMLHandler en iedere knoop wordt geinspecteerd. De syntax van een object was:
`<object name="ahafragmentnaam" type="aha/text"> </object>`
En de knoop object heeft dus twee parameters te weten "name"en "type". De XMLHandler controleert de knoopnaam en het type van de knoop. Indien deze "object" en "aha/text" zijn wordt de knoop als een aha object afgehandeld.

De volgende acties worden dan ondernomen:

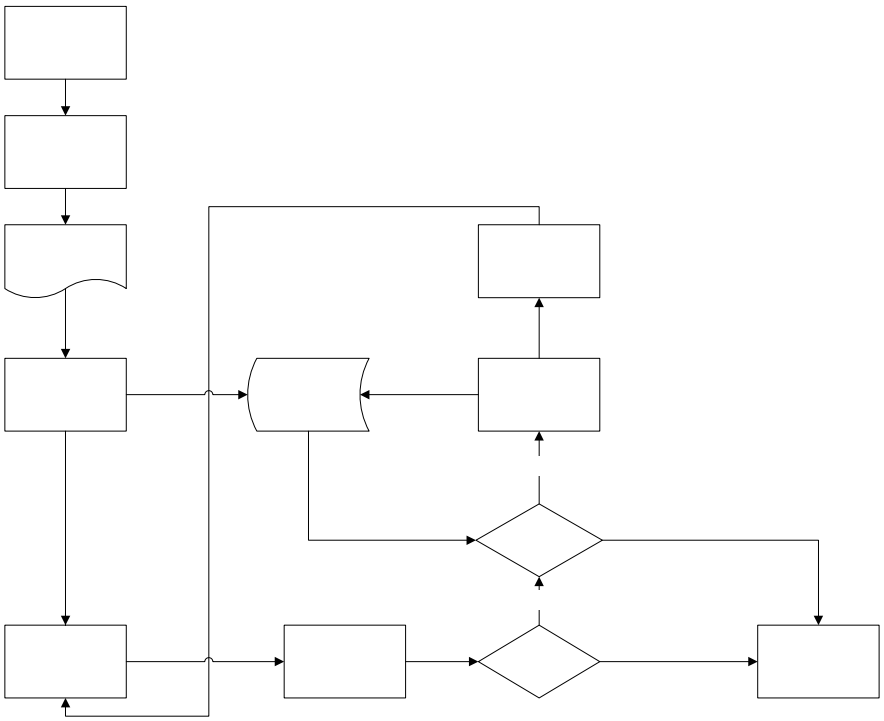
1. Het concept welke gekoppeld is aan het fragment wordt opgehaald
2. De access regels worden uitgevoerd van het desbetreffende fragment
3. De showability waarde wordt uit het UM gelezen van dat fragment
4. Er wordt gecontroleerd of aan die showability waarde een basisfragment te koppelen valt. Zo niet dan wordt het default-fragment gekoppeld aan het returnfragment.
5. Het returnfragment wordt omgezet naar een boom en deze wordt op dezelfde manier gecontroleerd als het hoofddocument.
6. Indien er geen combinaties van knoopnaam=object en knooptype=”aha/text” meer gevonden wordt, wordt de pagina gepresenteerd aan de gebruiker.

Deze wijzingen zijn alle gemaakt in de XMLHandler. Daar het systeem een showability waarde uit het UM moet kunnen lezen, moeten deze waarden wel beschikbaar zijn. Hiervoor zijn enkele uitbreidingen gedaan in de basisconcept structuur. § 5.3. beschrijft de wijzingen in de concept structuur. Deze wijzingen werden in de volgende modules van het systeem gewijzigd:
XMLConceptDB, hierin wordt opgeslagen hoe de interne structuur van een concept eruit ziet. Aangezien de interne structuur uitgebreid is, moest deze functie aangepast worden om deze uitbreiding te kunnen verwerken.
Case module, in de nieuwe concept structuur heeft het showability attribuut een default-fragment en casevalue’s. De Casevalues is een lijst van cases. Een case is een showability waarde gekoppeld aan een returnfragment. Het systeem kent het type case in versie 1.96 niet, daarom is deze aangemaakt op deze plaats. De grootste wijzigingen zijn gemaakt in de XMLHandler.

6.3. Terminatie garanderen

In § 4.3.1. wordt een terminatie algoritme beschreven welke het construeren van een boom welke groter dan 500 knopen is verhindert. Dit terminatie algoritme is geïmplementeerd door de XMLHandler uit te breiden. Een document X(HT)ML wordt opgebouwd als boom. Vervolgens verwerkt een functie “traverse” deze boom tot een document welke aan de gebruiker getoond wordt. Direct nadat het XML document omgebouwd is tot boom, wordt een teller “aantalfragmenten” op nul gezet. De boom wordt verwerkt zoals beschreven in 6.2. Iedere keer dat een “AHA! object” wordt gevonden, wordt de teller “aantalfragmenten” met één opgehoogd. Indien de teller 500 bereikt worden nieuwe object knopen van het type “aha/text” niet meer verwerkt en wordt de pagina getoond aan de gebruiker. De keuze om een nieuwe recursieve stap te maken wordt versterkt met de eis dat het totaal aantal AHA! object knopen die door de AE op die pagina geplaatst zijn minder dan 500 moet zijn.

Dit eenvoudige terminatie algoritme zorgt ervoor dat de AE niet kan vastlopen. Het is zoals in hoofdstuk 4 is uitgelegd een vrij beperkte manier van oneindige recursie herkenning.



Figuur 35 terminatie algoritme

Processresource

6.4. Correct parallellisme garanderen

In het AHA systeem zou het mogelijk moeten zijn om per gebruiker parallel twee (of meer) pagina's op te vragen. Parallel pagina's aanvragen kan voorkomen indien men een frame gebruikt. De afhandeling van deze frames is niet een eenduidig vastgelegd. Buiten de volgorde van pagina verwerking is er ook nog een volgorde waarmee de UM updates worden verwerkt. Het afhandelen van het UM gebeurt op dit moment op een niet een eenduidige manier. Dit levert twee deelproblemen op:

1. inconsistentie
2. livelock

Convert XHTML document to tree

Document in tree format

6.4.1. Inconsistentie

Als er parallel UM mutaties worden uitgevoerd kunnen deze update processen elkaar beïnvloeden.
Voorbeeld (1,2 A en B zijn atomaire acties die door het systeem worden uitgevoerd).

```
X=0
Proces 1.
1     X:=X+10
2     If x=10 → showfragment 1
Proces 2
A     X:=x+20
B     If x=20 → showfragment 2
```

De volgorde van afhandeling maakt veel uit. Dit levert echter alleen incorrecte resultaten op en geen live-lock. De volgorde 1,2,A,B levert fragment 1 op. De volgorde 1A2B levert geen enkel fragment op.

6.4.2. Livelock

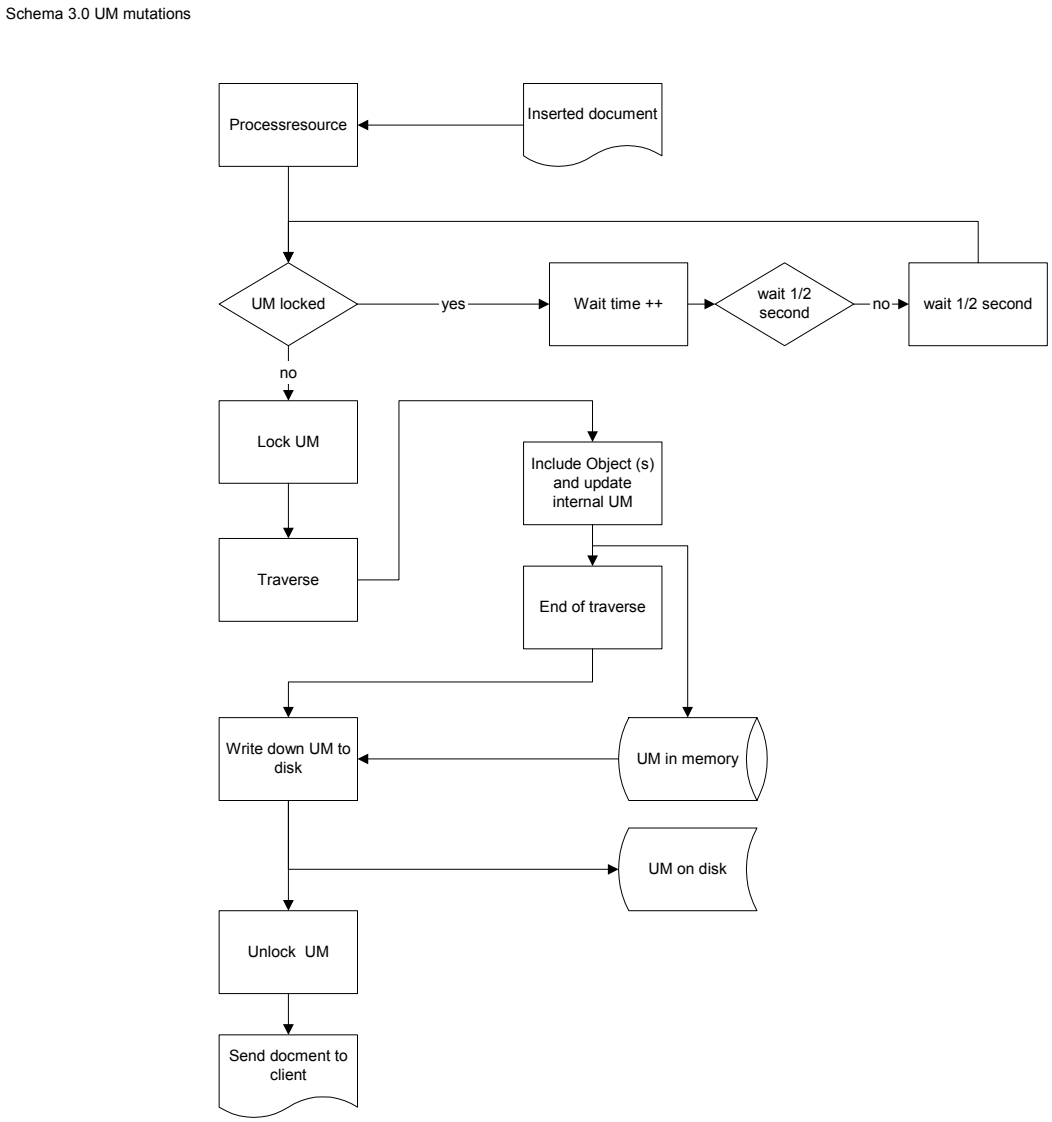
Livelock is het ongecontroleerd doorwerken aan een niet eindigende recursie constructie.
Dit kan alleen optreden in combinatie met de vanaf AHA! versie 3.0. mogelijke recursieve fragmenten.
Beschouw het volgende voorbeeld. Er wordt gebruik gemaakt van pseudo code; in AHA! bestaat geen while do constructie. Deze is echter middels zelfinsluiting te simuleren.§ 4.2 geeft zo een constructie als voorbeeld.

```
X=0
Proces 1.
while x<10 do
      Showfragment
      X+1
      End
Proces 2.
While x>10 do
      Showfragment
      X-1;
      End
```

Doordat proces 1 variabele x verhoogt en proces 2 de variabele x verlaagt kan de situatie ontstaan waarbij bij beide processen niet termineren. Deze blijven fragmenten tonen. In de praktijk zal een van de twee processen ooit eindigen doordat zijn eindconditie bereikt wordt (er zal nooit precies een 50-50 verdeling van processor tijd zijn). De volgorde van afhandeling is echter niet deterministisch. En dit kan zeer ongewenst zijn.

6.4.3. Oplossing voor correct parallellisme in AHA!

Als oplossingstrategie is gekozen voor de semaforen techniek van Dijkstra [DIJKSTRA1967]. Voordat een pagina verwerkt wordt, wordt het UM vergrendeld. Zolang het UM vergrendeld is, kan er geen andere pagina aanvraag van die gebruiker verwerkt worden. De andere pagina's worden netjes in een queue gestopt (automatisch door het systeem, omdat bij iedere pagina aanvraag een nieuwe thread wordt gestart door de webserver). Indien de AHA engine vastloopt op een ongedefinieerde wijze ("stekker uit het systeem methode") kan het UM mogelijk in een vergrendelde situatie achterblijven. Indien dit gebeurt moet de AHA engine opnieuw opgestart worden. Het UM wordt vergrendeld m.b.v. een niet persistente sleutel in het UM. Bij het opstarten van de engine worden alle niet persistente sleutels automatisch op hun standaard waarde gezet. (welke "niet vergrendeld" is). Hierdoor is het niet mogelijk dat een gebruiker vergrendeld blijft na een herstart van het systeem. Dit vergrendelingschema is in Figuur 36 weergegeven.



Figuur 36 UM lock mechanisme

7. Conclusie en aanbevelingen

7.1. Conclusies

De nieuwe fragment structuur welke in dit verslag gepresenteerd is, biedt vele nieuwe mogelijkheden. Zoals: recursieve fragment structuren, hergebruik van fragmenten, autonoom gedrag van fragment (UM mutaties) en de DM/content splitsing. Al deze mogelijkheden bieden auteurs meer mogelijkheden en een gebruikersvriendelijke methode om adaptieve applicaties te maken en maken het Adaptief Hypermedia Systeem transparanter en gestructureerder. Vooral gestructureerd omgaan met pagina's, fragmenten en regels is een groot voordeel. Fragmenten hangen niet meer losjes in diverse pagina's, maar zijn centraal opgeslagen en gedefinieerd. Dit biedt een betrouwbare en consistente manier van informatie presenteren. Deze manier van informatie verwerking en opslag is ook in brede zin bruikbaar. Niet alleen in adaptieve applicaties zijn fragmenten identificeerbare brokken informatie. Informatie systemen, in het algemeen, maken gebruik van fragmenten, die hergebruikt worden. Indien een informatie systeem volgens deze fragmentstructuur opgezet wordt, biedt dit voordelen in onderhoudbaarheid (op een plaats wijzigingen doorvoeren) en beheersbaarheid van informatiehoeveelheid (op één plaats opslaan). Verder biedt deze nieuwe structuur de mogelijkheid om in het User model bij te houden welke fragmenten gezien zijn. Bij meerdere fragmenten op een pagina kan zo ieder fragment zijn eigen presentatie bepalen op basis van de eerdere fragmenten op dezelfde pagina.

Verder kan geconcludeerd worden dat de nieuwe constructie enkele zeer gevaarlijke neveneffecten heeft. Oneindige recursie is er hier een van. Indien een auteur een zeer complex informatie systeem maakt, waarbij veel fragmenten zelf ook fragmenten bevatten (insluiten), kan dit tot cycles leiden. Een cycle is niet per definitie fout maar wel verdacht. Een adaptief systeem op cycles controleren is niet moeilijk en zou vooraf gedaan kunnen worden. Indien een auteur toch cycles wil toestaan is zijn informatie-systeem, zijn er in het systeem enkele veiligheidsmaatregelen te treffen om oneindige recursie te voorkomen. Hiermee is oneindige recursie te voorkomen. Deze beveiligingen zijn in AHA! 3.0 ingebouwd en getest.
Een andere conclusie die getrokken kan worden, is dat het AHA! systeem een rijker domein model heeft gekregen, de ontkoppeling tussen de content en het domein model is eindelijk gerealiseerd. In versie 1.96 was het domein model deels terug te vinden in de content. Versie 3.0 ontkoppelt fragment informatie, domeinmodel en fragment content van de pagina's. Deze ontkoppeling maakt het gebruik van algemene XML code in samenwerking met de AHA! engine mogelijk, voor iedere XML taal welke met een webserver kan communiceren. Er is geëxperimenteerd met de taal SMIL. Verder valt te denken aan talen zoals SVG (Structured Vector Graphic) een taal welke tekeningen middels vectors (lijnen) beschrijft, waar bijvoorbeeld voor iedere lijn een adaptief fragment getoond kan worden, op die manier kan een gebruiker aangeven in welke kleur de lijnen getoond moeten worden. Door de definitie van een lijn als een adaptief fragment in een SVG document te plaatsen, is dit mogelijk.

Adaptieve systemen zullen in de toekomst een steeds prominentere rol in de maatschappij gaan spelen. Elektronisch leren is een groeiende markt en verwacht wordt dat in de toekomst steeds meer studenten elektronisch cursussen gaan volgen. Dit gaat de docent niet vervangen maar breidt het mogelijke studiemateriaal uit. Sommige cursussen zullen volledig zonder direct contact met een docent te volgen zijn. De Universiteit van Liverpool maakt hier al intensief gebruik van (1500 studenten) in hun KIT E-learning systeem. Ook de universiteit van Eindhoven heeft al enkele jaren een online adaptieve cursus "Hypermediastructuren en –systemen 2L690" in hun vakkenaanbod zitten. Naast educatieve systemen zijn alle informatie systemen adaptief te maken. Een gebruiker kan liever een blauwe achtergrond op een nieuwspagina wensen dan een rode. Alle gebieden welke met (web)presentatie en persoonlijke voorkeuren te combineren zijn, kan in het licht van adaptieve Hypermedia gezien worden.

AHA! 3.0 is een Adaptief Hypermedia systeem welke op diverse manieren de presentatie kan aanpassen aan de wensen/doelen/ervaring van een gebruiker.

7.2. Aanbevelingen

Na 9 maanden onderzoek binnen het AHA! project, zijn diverse ideeën geboren welke AHA! naar een hoger plan zou kunnen tillen. Gesterkt door de ervaring met het begeleiden van AIO's S.G. Loeber en P. Barna bij het opzetten van een prototype van een adapatief museum, waag ik me aan het doen van enkele aanbevelingen.

7.2.1. Aanpassen editors

De editors welke gemaakt zijn om adaptieve applicaties mee te ontwerpen en te genereren zijn niet voldoende getest tevens is verzuimd om serieus te kijken naar de GUI/gebruiksvriendelijkheid van deze editors. De editors zitten op sommige vlakken onlogisch in elkaar. Gebruikers verwachten andere reacties van deze editors wanneer zij enkele vrij basale acties ermee willen uitvoeren. Dat de editors veelal in stage en afstudeeropdrachten in elkaar zijn gezet, verklaart dit gebrek aan volwassenheid. Toch zou het wenselijk zijn dat vaste werknemers van het AHA! project dit oppakken en eventueel dit zelf aanpassen of nieuwe stageopdrachten genereren welke de GUI/gebruiksgemak zouden inspecteren, aanpassen en de functionaliteit zouden testen. Een ander probleem bij de editors is het gebrek aan foutmeldingen. De editors vinden allerlei foute constructies prima, terwijl het AHA! systeem er vervolgens op vastloopt (bijvoorbeeld twee concepten met zelfde resource). Er zou meer feedback moeten komen vanuit de editors indien de gebruiker iets fouts doet. Verder zou de gebruiker gewaarschuwd kunnen worden voor de meest triviale cycles (waarbij fragmenten, fragmenten insluiten die vervolgens dezelfde fragmenten insluiten).

7.2.2. Versiebeheer systeem

Diverse malen is tijdens het afstuderen veel tijd verloren met het synchroniseren van AHA! versie 2.0 met AHA! versie 3.0. De noodzaak van een versiebeheer systeem is pas laat onderkend en geïmplementeerd. Ook aan het eind van de afstudeerperiode blijven deze versie problemen het AHA! team als een spook achtervolgen. Onvoldoende communicatie tussen ontwikkelaars, gebrek aan changelogs en algemene regels betreffende software ontwikkeling binnen het AHA! project, zijn hier de oorzaak van. Dit gebrek aan volwassen software ontwikkelen gaat goed zolang er niet meer dan 1 a 2 personen ontwikkelen. Het wordt een fiasco indien meer dan 2 mensen gaan ontwikkelen en bug-fixen in dezelfde modules van het AHA!. Dit gebrek aan volwassen software ontwikkelen staat de ontwikkeling en betrouwbaarheid van het AHA! systeem ernstig in de weg.

7.2.3. Recursieve structuren afbreken en visualiseren

In AHA! 3.0 worden recursieven structuren op een nogal eenvoudige manier afgebroken, de in § 6.3 geopperde oplossing zou geïmplementeerd kunnen worden. Verder onderzoek naar recursie in o.a. kennis updates (propagation) wordt op dit moment (mei 2003) gedaan door D. Smit als afstudeeropdracht binnen het AHA! team. Het visualiseren van recursieve structuren en de verbindingen tussen fragmenten en pagina's zou een toevoeging zijn voor een auteur. Snel en overzichtelijk kunnen zien welk fragment waar gebruikt wordt kan bij het aanpassen van een fragment erg nuttig zijn. Hiermee wordt de impact van de wijziging duidelijk.

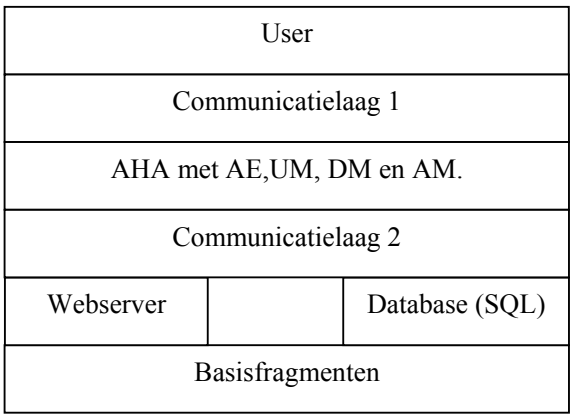
7.2.4. Fragment identificatie extern opslaan

Op dit moment controleert het AHA! systeem een document op <object> tags. Dit is in de code verweven. Indien er een nieuwe taal gebruikt wil worden welke geen ondersteuning biedt voor <object> tags zou het wenselijk zijn, om de AHA! engine een nieuwe tag te laten herkennen. Indien er extern buiten het systeem een tabel gedefinieerd wordt welke deze fragment herkenning en identificatie beschrijft zou dit andere XML talen in staat stellen snel ook van het AHA! systeem gebruik te maken. Z'n tabel ziet er als volgt uit:

Taal	Fragment herkennings tag	Fragment identificatie veld	Discriminatieveld t.o.v. regulier tag-gebruik
XHTML	Object	Name="fragmentnaam"	type="aha/text"
SMIL	Ref	name="fragmentnaam"	type="aha/text"

7.2.5. Andere XML toepassingen exploreren

Nu het AHA! 3.0 systeem in staat is om generiek XML te verwerken zou gekeken kunnen worden of de eis dat AHA! met een webserver communiceert overbodig gemaakt kan worden. Er zou gekeken kunnen worden naar een universele manier van aanvraag/verwerking/presentatie met behulp van het AHA systeem. Beschouw onderstaand schema:



Figuur 37 universele AHA! XML koppeling

Dit zou bijvoorbeeld adaptieve databases mogelijk maken. Een gebruiker doet een query op een database waar sommige parameters als fragmenten worden gezien. De AHA! engine verandert deze parameters in voor die gebruiker als meest geschikte code en stuurt dit verzoek door naar de database. Een voorbeeld query zou dan kunnen zijn: (de gebruiker die deze query aanvraag doet verdient 3000 euro per jaar).

```
Select     name,price
From       scooterproductlist
Where      price < [fragment user.salary]
```

De AHA! Engine verandert dit in de volgende query:

```
Select     name,price
From       scooterproductlist
Where      price < 3000
```

Een andere toepassing zou kunnen zijn dat het AHA! systeem bijvoorbeeld tussen een applicatie en het filesysteem nestelt. Stel een gebruiker gebruikt het tekenpakket AutoCad (een in de technische industrie veel gebruikt tekenprogramma) en vraagt een tekening aan het systeem om op te laden. Deze aanvraag wordt naar AHA! gestuurd, AHA! bewerkt de tekening adaptief op basis van het gebruikersprofiel (UM) en retourneert de tekening aan AutoCad.
Adaptieve elementen in een technische tekening kunnen bijvoorbeeld notatie van maten (tolerantie notatie), kleur van lijnen, gewenst type hartlijn zijn. Als er een interface gemaakt kan worden tussen het AHA! systeem en applicaties zou dit adaptiviteit op meerdere gebieden beschikbaar maken.
Een iets minder ambitieuze doelstelling is om te beginnen met het onderzoeken naar andere toepassingen (SMIL SGV) welke met een webserver communiceren en als structuur XML gebruiken.

8. Verklarende woordenlijst

AE	Adaptive Engine; programmatuur welke in een AHS wordt gebruikt om te presentatie te verzorgen.
AHA!	Adaptive Hypermedia for All
AHA	Adaptive Hypermedia Architecture
AHS	Adaptief Hypermedia Systeem; systeem welke zijn presentatie aanpast aan de gebruiker
Boom	representatie van een structuur middels lijnen en knopen waarbij cylces niet zijn toegestaan.
DM	Domein Model; model van systeem
DTD	Document Type Definition; document met spelregels waaraan een XML document zich moet houden.
Fragment	Identificeerbaar stuk informatie welke mogelijk meermalig in een systeem gebruikt wordt.
Graaf	representatie van een structuur middels lijnen en knopen waarbij cycles zijn toegestaan
Hypertext	structuur van een pagina waarbij in een pagina een directe verwijzingen te volgen valt naar andere pagina's, veel toegepast op het internet waarbij "links" deze directe verwijzingen zijn.
Hypermedia	Uitbreiding van Hypertext met multimedia componenten
Recursie	Zelf insluiting; programmeer/specificeer techniek waarbij een object zichzelf includeerd.
SMIL	Synchronized Multimedia Integration Language; multimedia taal
SVG	Scalable Vector Graphics; tekenformaat waarbij lijnen als wiskundige functies worden beschreven en daardoor oneindig vergroot kunnen worden zonder pixels te zien. (tegenhanger van bitmap tekening).
Tag	knoop in een XML document
TU/e	Technische Universiteit Eindhoven
UM	User Model; model van een gebruiker waar gebruikers speficifieke kenmerken in staan
XML	eXtensible Markup Language; gestructueerde taal
XHTML	eXtensible HyperText Markup Language; uitbreiding van HTML; HTML pagina's voldoen nu aan de eisen van XML

9. Onderzoeksbronnen

9.1. Referenties:

[BOYLE1994] C. Boyle, A.O. *Encarnacion. Metadoc: An Adaptive Hypertext Reading System.* User Modeling and User-Adapted Interaction 4, 1-19, 1994

[BRAY1998] T. Bray, J. Paoli, C.M. Sperberg-McQueen. *Extensible Markup Language (XML).* W3C Recommendation januari 2003 http://www.w3c.org/tr/rec-xml.html.

[BRUSILOVSKY1996] Brusilovsky, P. *Methods and techniques of Adaptive Hypermedia*. User Modeling and User-adapted interaction. 6: 87-126 1996 Kluwer Acadamic Publishers

[BRUSILOVSKY2001] Brusilovsky, P. *Adaptive Hypermedia*. User Modeling and User-adapted interaction. 11: 87-110 2001 Kluwer Acadamic Publishers

[DEBRA1996] De Bra, P.M.E, *Teaching Hypertext and hypermedia through the Web*. Journal of Universal Computer Science 2 (12) 797-804

[DEBRA2003a] P.M.E. De Bra, , N Stash, B.H. de Lange. *AHA! Adding Adaptive behavior to Websites.* Conferentie over Webapplication, Ede Nederland, Mei 2003

[DEBRA2003b] P.M.E. De Bra, , A. Aerts, B.A. Berden, B.H. de Lange *Escape from the Tyranny of the Textbook: Adaptive Object Inclusion in AHA!*, Conferentie Elearn 2003, Phoenix Arizona USA, Nov 2003

[DIJKSTRA1967] E.W. Dijkstra, *The structure of the "THE"-multiprogramming System"*. ACM Symposium "operating system principles" 1967 (tevens EWD196)

[HOSCHKA1998] P. Hoschka (ed.) *Synchronized multimedia Integration Language.* W3C Recommendation, Juni 1998 htttp://www.w3c.org/tr/rec-smil/

[RAGGETT1999] D. Raggett (ed.) *HTML 4.01 Specification*. W3C recommendation december 1999 http://www.w3.org/TR/html401/

[SHNEIDERMAN1989] B.Shneiderman, G.Kearsley. *Hypertext Hans-On!: An introduction to a New Way of Organizing and Accessing Information.* 1989 Addison Wesley.

9.2. Afstudeerverslagen

Alle afstudeerverslagen zijn in de faculteitsbibliotheek van wiskunde en informatica te vinden.

[BERDEN2003] B.A. Berden. *Stabiliteit binnen AHA!* Afstudeerscriptie, Technische Universiteit Eindhoven Mei 2003

[POEL2003] B.A. van der Poel, *XML Editor from design to implementation* Afstudeerscriptie, Technische Universiteit Eindhoven februari 2003

[HEESSEN2002] W.H. Heessen, *MUD layout generation and visualisation.* Afstudeerscriptie, Technische Universiteit Eindhoven augustus 2002

[ROUSSEAU2003] B.J.P. Rousseau. *Graph Author een AHA! Auteurstool,* Afstudeerscriptie, Technische Universiteit Eindhoven februari 2003

9.3. Stageverslagen

Deze afstudeerverslagen zijn bij de vakgroep IS van de faculteit wiskunde en informatica te verkrijgen

[ANSEMS2002] M.L.A. Ansems. *AHA Zichtbaarheid van fragmenten*. Technische Universiteit Eindhoven december 2002

[RUITER2001] J.P. Ruiter. *Generatelist Editor.* Technische Universiteit Eindhoven juni 2001

9.4. Literatuur

[1] Jan Willem van den Brandhof, *Gebruik je hersens*, Verba 1998 ISBN 90-5513-339-6

[2] Hongjing Wu, *A reference Architecture for Adaptive Hypermedia Applications* TU/e 2003 ISBN 90-386-0572-2

[3] J. Jaworski, *JAVA 1.2 Unleashed* SAMS 1998 ISBN 1-57521-389-3

[4] L.Cassady-Dorion *Industrial Strength JAVA* 1997 1-56205-634-4

[5] L.Calvi, A.I.Cristea. *Toward Generic Adaptive System: Analysis of a Case Study* TU/e 2001 paper.

[6] *Adaptive Hypermedia and Adaptive Web_based Systems.* Proceedings of International Conference. Springer AH 2000, Trento, Italy, augustus 2000.

9.5. Lijst van figuren

Figuur 1 Hypertext *3*
Figuur 2 Taxonomy technologieën van een Adaptieve Hypermedia Systemen *3*
Figuur 3 fragment en pagina voorbeeld *3*
Figuur 4 conceptueel overzicht pagina-fragment-basisfragment *3*
Figuur 5 conceptueel overzicht pagina AHA! 1.96 ..... *3*
Figuur 6 Conceptueel overzicht AHS systeem *3*
Figuur 7 Taxonomy van de in AHA! 1.96 geboden technologieën *3*
Figuur 8 Architectuur AHS AHA! 1.96 ..... *3*
Figuur 9 Taxonomy van de in AHA! 3.0 geboden technologieën *3*
Figuur 10 Architectuur AHA! 3.0. ..... *3*
Figuur 11 Stroomschema SSI fragment verwerking *3*
Figuur 12 stroomschema object tag verwerking *3*
Figuur 13 fragment content opslag *3*
Figuur 14 node fragmenten graaf *3*
Figuur 15 voorbeeld concept → structuur → presentatie *3*
Figuur 16 structuurgraaf onderwijsapplicatie Europa *3*
Figuur 17 Document boom onderwijs applicatie Europa *3*
Figuur 18 Recursief fragment schema *3*
Figuur 19 structuurgraaf recursief fragment schema *3*
Figuur 20 structuurgraaf onderwijsapplicatie Europa *Figuur 21 afgebroken documentboom* *3*
Figuur 22 terminatie van oneindige recursie met patroonherkenning *3*
Figuur 23 terminatie van oneindige recursie met patroonherkenning II *3*
Figuur 24 complexiteit van structuur in AHA 1.96 ..... *3*
Figuur 25 complexiteit van structuur in AHA! 3.0. ..... *3*
Figuur 26 Structuur overzicht van een AHS in samenwerking met SMIL *3*
Figuur 27 Fragment opslag AHA! 1.96 ..... *3*
Figuur 28 Fragment opslag AHA! 3.0 ..... *3*
Figuur 29 autonoom fragment gedrag *3*
Figuur 30 Concept wijzigingen 1.96 → 3.0 ..... *3*
Figuur 31 showability bepalen in Concept List Editor *3*
Figuur 32 basisfragmenten koppelen aan showability in concept list editor *3*
Figuur 33 Graph author fragment toewijzing *3*
Figuur 34 Well formed XML document *3*
Figuur 35 terminatie algoritme *3*
Figuur 36 UM lock mechanisme *3*
Figuur 37 universele AHA! XML koppeling *3*

10. Dankwoord

Graag wil op deze plaats een aantal mensen bedanken die zowel mijn afstudeeropdracht en mijn tijd op de TU/e gestimuleerd bespoedigd en aangenamer hebben gemaakt.

Allereerst wil ik de mensen in mijn vakgroep bedanken, en specifiek prof.dr.Paul DeBra voor de kans die ik gekregen heb om in het AHA! Project aan mijn afstudeeropdracht te werken en zijn kritische en realistische kijk op oplossingen, dr.Ad Aerts voor zijn kritische en theoretische visie op de oplossingen welke voorgedragen werden en zijn hulp bij het schrijven van de scriptie. Bart Berden en Brendan Rousseau voor de vele brainstorm sessies waardoor menig idee (terecht) de prullenbak is ingegaan en gezellig tijd binnen en buiten ons IBBB hokje ☺

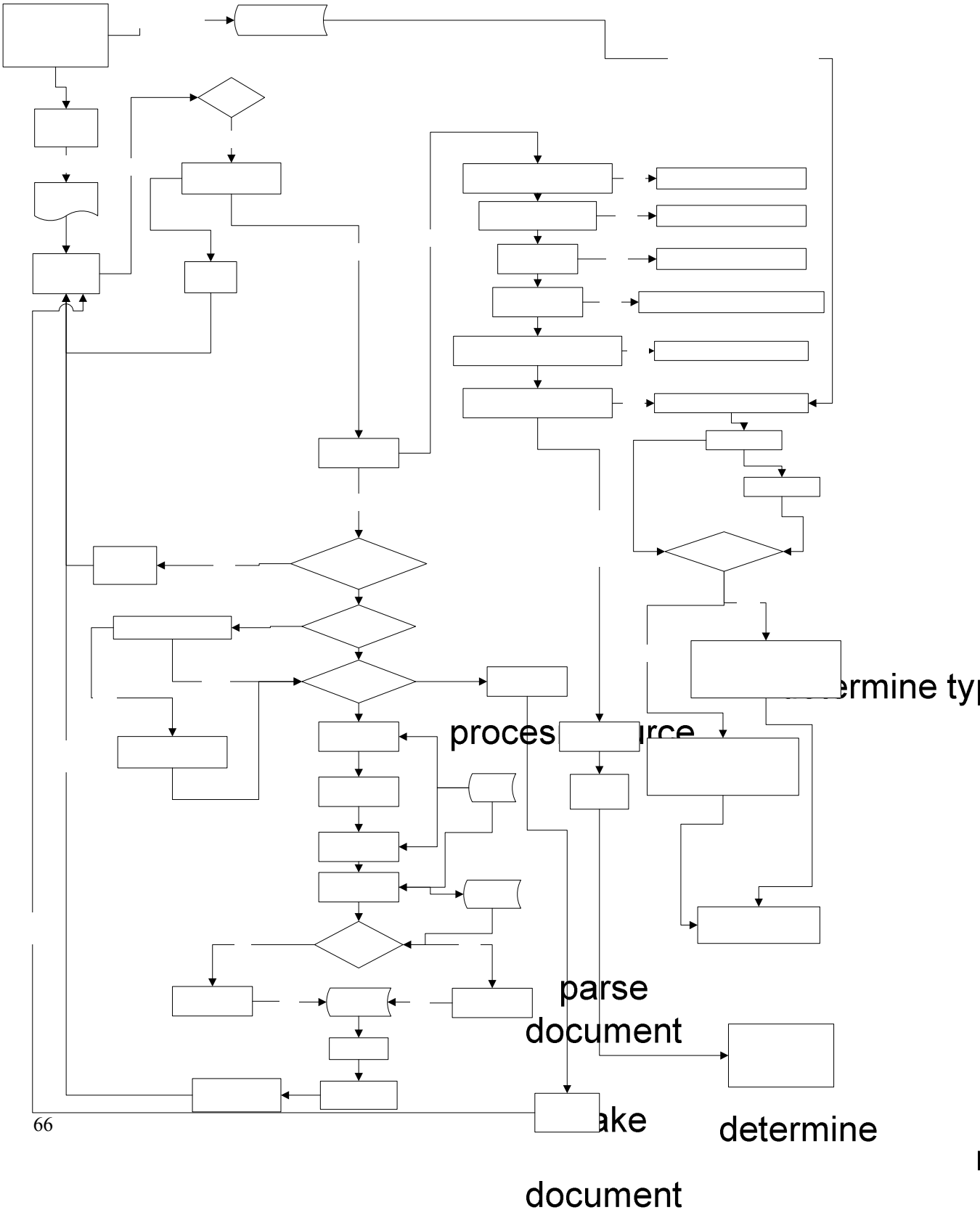
Verder wil ik mijn vrienden: Oscar, Rowenna, Anneguus, Bart, Erik en Wouter bedanken voor de prettige samenwerking tijdens de projecten en voorbereiding op tentamens gedurende de gehele periode op de TU/e. Als mede de eindeloze gesprekken over het afstuderen de toekomst, maar vooral de gezellige tijd in en om de Universiteit.

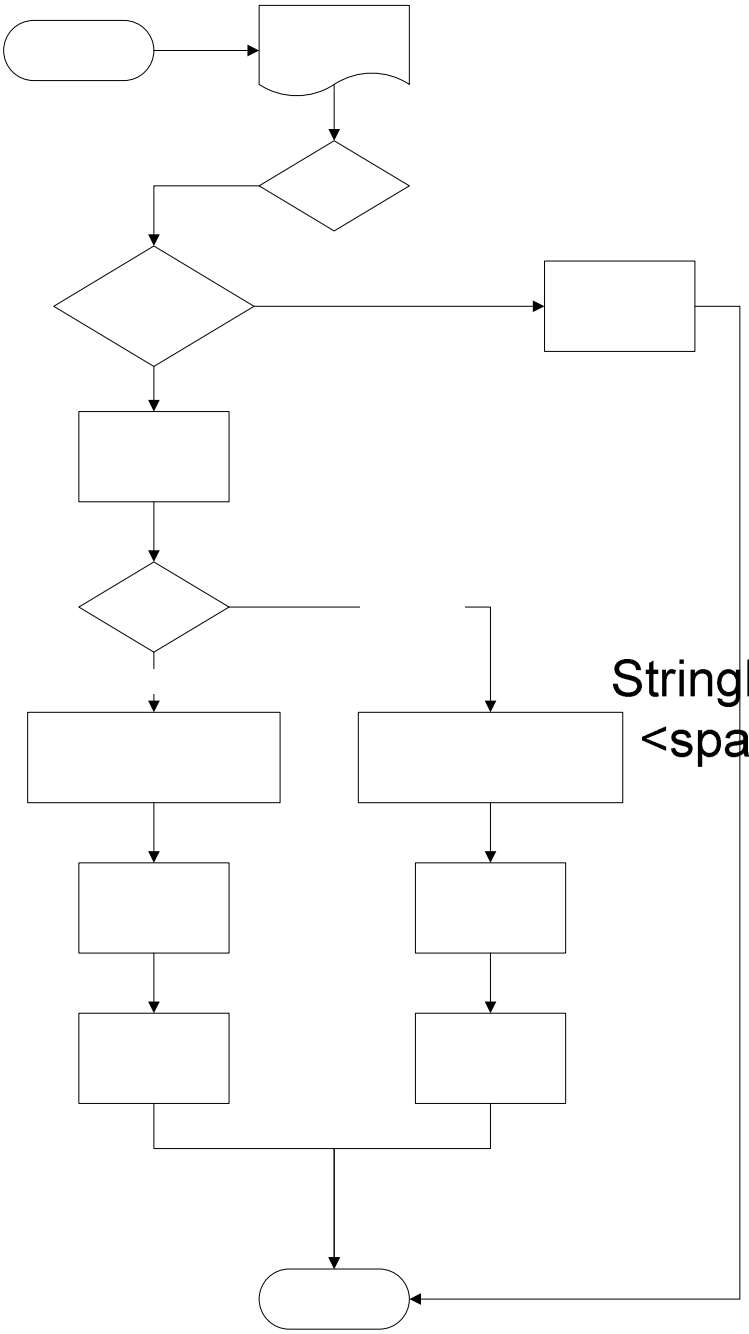
Als laatste gaat mijn dank uit naar mijn familie, Floris; voor de vele gesprekken waardoor ik een ruimer beeld van de wereld heb gekregen, Marieken voor het vertrouwen welke je altijd in mij gehouden hebt maar vooral mijn ouders Ria en Hans; jullie steun vertrouwen en altijd geïnteresseerde houding ten opzichte van mijn studie, heeft er zeker voor gezorgd dat ik zover ben gekomen om deze studie te kunnen beginnen. Zonder jullie positieve houding en financiële steun, zou dit resultaat niet mogelijk geweest zijn, bedankt hiervoor!

Barend

Bijlagen

bijlage I software stroomschema's





Stringbuffer with
 node

if class=conditional

Bijlage II papers