Universiteit Twente

faculteit der
informatica

Een Architectuur voor Multimedia Databases

E. van het Hof, A.P. de Vries, H.E. Blok en H.M. Blanken

{hof,arjen,blokh,blanken}@cs.utwente.nl

Samenvatting

Digitale multimedia bibliotheken kunnen niet worden ondersteund met traditionele monolithische database systemen. Voor het uitvoeren van queries op multimedia objecten maken we gebruik van meta-data. Eindgebruikers kunnen hun informatiebehoefte meestal niet in termen van deze meta-data formuleren. Om dit probleem op te lossen stellen we een iteratief query proces voor. Daarnaast stellen deelnemers aan digitale multimedia bibliotheken verschillende eisen aan de architectuur van een multimedia database. We presenteren een architectuur waarbij de multimedia objecten, de meta-data en de processen die meta-data opleveren, van elkaar worden gescheiden. Deze componenten verbinden we met elkaar door middel van een software bus. Hierdoor ontstaat een gedistribueerde architectuur voor een multimedia database. Om de bruikbaarheid van de opzet te testen, hebben we een prototype implementatie verzorgd.

1 Inleiding

Ontwikkelingen op het gebied van netwerken en opslagcapaciteit maken het mogelijk dat steeds meer gegevens digitaal worden opgeslagen. Deze gegevens bestaan niet alleen uit tekst, maar ook uit verschillende soorten multimedia. Een voorbeeld van deze ontwikkeling is de enorme groei van het Internet. Er komen per dag nieuwe WWW-pagina's, die verschillende soorten multimedia bevatten. Ook bedrijven creëren grote digitale bibliotheken.

In de projecten AMIS [Amis] en miRRor [deVries98d] ontwikkelen we een architectuur om grote hoeveelheden multimedia data op te slaan en te bevragen. Naarmate de hoeveelheid digitale multimedia informatie toeneemt, wordt ook de behoefte om te kunnen zoeken in die informatie groter. Helaas zijn huidige (relationele en object-relationele) databases niet voldoende in staat om met multimedia gegevens om te gaan. De multimedia gegevens kunnen weliswaar worden opgeslagen in zogenaamde Binary Large Objects (BLOBs), maar ze kunnen slechts op een machine-gerichte wijze worden geïndexeerd. Voor de eindgebruikers is er niet voldoende ondersteuning [deVries98a].

In sectie 2 bespreken we de eigenschappen van multimedia data en de beantwoording van vragen aan multimedia databases. Daarna zullen we, aan de hand van een voorbeeld, de gebruikers van multimedia databases identificeren. De eisen van deze gebruikers zullen we gebruiken om een probleemstelling te formuleren. Vervolgens analyseren we twee architecturen waarmee, ter ondersteuning van digitale bibliotheken, multimedia databases kunnen worden gebouwd. Nadat we in het kort de verschillende componenten van de architectuur hebben besproken, zullen we ingaan op het ontwerp van deze componenten. Om de bruikbaarheid van de opzet te testen hebben we een prototype implementatie gemaakt. Dit prototype bespreken we in sectie 6. We sluiten af met conclusies en aanbevelingen.

2 Achtergrond

Zoals we in de inleiding al hebben aangegeven, is het opslaan van multimedia objecten in BLOBs meestal niet voldoende om vragen over de multimedia objecten te kunnen beantwoorden. Dit komt omdat in de meeste database systemen multimedia objecten niet kunnen worden geselecteerd door middel van een conditie op de inhoud van een BLOB.

Om toch vragen over multimedia objecten te kunnen verwerken, kunnen we informatie opslaan over deze objecten, zogenaamde meta-informatie. Eén manier om meta-informatie te verkrijgen is door handmatige annotatie. Een nadeel van het gebruik van annotaties is, dat verschillende mensen dezelfde multimedia titel anders annoteren. Een annotatie kan ook niet alle details van een multimedia titel beschrijven.

Een andere vorm van meta-informatie bestaat uit zogenaamde features. Dit zijn door speciale algoritmen berekende gegevens, waarvan eigenschappen van de multimedia objecten kunnen worden afgeleid. Een voorbeeld van een feature is een kleurenhistogram. Aan de hand van een kleurenhistogram kunnen plaatjes worden geselecteerd op kleureigenschappen. Belangrijk is, dat features automatisch, zonder tussenkomst van mensen, kunnen worden berekend [Faloutsos96]. De vragen van de gebruikers kunnen worden beantwoord met behulp van deze features. De features en annotaties kunnen worden opgeslagen in een database.

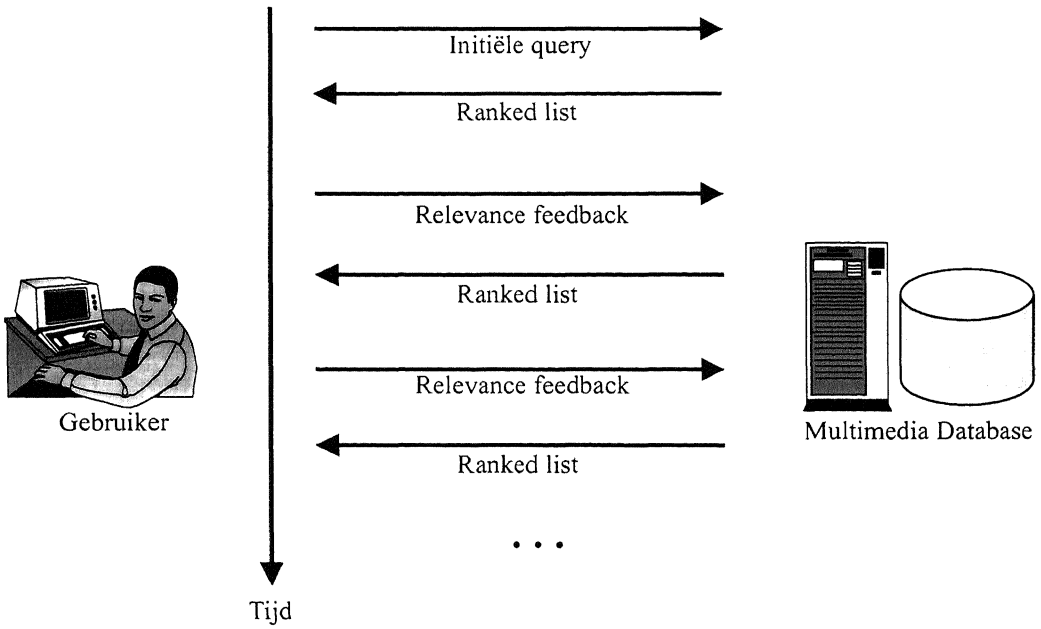
Een feature is een punt in een multidimensionale ruimte. Met behulp van een afstandsmaat in die ruimte, kunnen features met elkaar vergeleken worden. De afstand tussen de features van twee verschillende objecten kunnen we gebruiken als een maat van verschil tussen die objecten [Duda73]. Voorbeelden van een dergelijk gebruik van features zijn het Musclefish audio retrieval systeem [Wold96] en de Virage Image Engine [Virage98].

Een enkele feature, zoals een kleurenhistogram, zegt natuurlijk weinig over een multimedia object. Door verschillende soorten features van multimedia objecten op te slaan, ontstaan er verschillende representaties van deze objecten. Een vraag aan de multimedia database kan dan worden beantwoord door gebruik te maken van deze verschillende representaties van de multimedia objecten.

Door gebruik te maken van meerdere features, kan een systeem worden ontwikkeld waarmee kan worden gezocht op eigenschappen van multimedia objecten. Dit idee is voor het eerst toegepast in [Niblack93] als het QBIC systeem (Query By Image Content) [QBIC]. Door te kijken naar de overeenkomsten tussen features kunnen (op een bepaald gebied) overeenkomstige multimedia objecten worden opgezocht. Dit staat bekend onder de naam "similarity queries".

Wanneer een feature overeenkomt met een concept dat de eindgebruiker kent, kunnen we de gebruiker direct door de feature ruimte laten navigeren. Deze vorm van het beantwoorden van een vraag door de eindgebruiker noemen we "navigational queries" [deVries98a].

Eindgebruikers kunnen een (multimediale) informatiebehoefte vaak niet direct formuleren in queries op features. Er zit een groot conceptueel verschil tussen de waarneming van de gebruikers en de features in het systeem. Wij vinden dat de database de gebruiker in een interactieve dialoog moet helpen zijn informatiebehoefte uit te drukken op systeemniveau [deVries98a]. In miRRor bestuderen we hoe deze dialoog ondersteund kan worden met behulp van technieken ontwikkeld in tekst retrieval [deVries98c]. In figuur 2.1 is de dialoog tussen gebruiker en multimedia database schematisch weergegeven.



Figuur 2.1: Dialoog tussen een gebruiker en een multimedia database

De gebruiker start de dialoog door een initiële query aan te bieden aan de multimedia database, bijvoorbeeld met behulp van keywords of een voorbeeld object. De multimedia database levert vervolgens een gesorteerde lijst op met multimedia objecten die voldoen aan de initiële query. Deze lijst noemen we een ranked list. De gebruiker kan vervolgens aangeven welke elementen uit de ranked list wel en welke niet aan zijn informatiebehoefte voldoen. Deze informatie wordt aan de multimedia database teruggegeven. Het teruggeven van informatie over het resultaat van de voorgaande query noemen we "relevance feedback". De database levert dan, aan de hand van de relevance feedback, een nieuwe ranked list op. Dit proces kan net zolang doorgaan totdat de gebruiker tevreden is met het resultaat. In de volgende sectie presenteren we een voorbeeld van een dialoog tussen een gebruiker en een multimedia database.

2.1 Voorbeeld van een multimedia database

In miRRor richten we het ontwerp van een multimedia database op het ondersteunen van digitale bibliotheken. De eisen van de gebruikers van digitale bibliotheken leggen al snel een aantal architecturale punten vast.

Een voorbeeld van een multimedia data collectie is de Internet site van CNN [CNN]. Op de site van CNN worden verschillende soorten multimedia aangeboden: tekst, foto's, maar ook geluidsfragmenten. Een ander voorbeeld van een aanbieder van multimedia informatie is een betaal-TV zender. Een betaal-TV zender wil wel dat gebruikers van de multimedia database films uit hun bestand kunnen vinden, maar ze willen niet dat die gebruikers de films kunnen bekijken zonder dat ze ervoor betalen.

Voor het zoeken in de multimedia data, moet er een database met features en annotaties worden aangelegd (de meta-data database). Geen van de aanbieders van multimedia titels wil dat hun concurrent de meta-data database beheert, omdat ze dan het zoekproces niet kunnen controleren. De meta-data database moet dus door een onafhankelijke organisatie beheerd worden. We verbinden de meta-data database via een netwerk met de media servers (de servers met de eigenlijke multimedia data) van de aanbieders.

De gebruikers kunnen via het netwerk zoeken in de meta-data database. De gebruiker gebruikt de multimedia database nu als volgt. Een gebruiker formuleert een initiële query. Dit kan een "similarity query" of een "navigational query" zijn. In alle gevallen maakt de gebruiker contact met de meta-data database. De meta-data database levert een ranked list op. De ranked list bestaat uit een aantal representaties van de multimedia objecten.

Stel dat de gebruiker zoekt op de naam van een acteur (met als doel een film te vinden waarin de acteur speelt). Als resultaat levert de meta-data database een aantal representaties op. Bijvoorbeeld titels van nieuwsberichten, waarin de naam van de acteur voortkomt. Tevens kan er een aantal titels van films worden opgeleverd waarin de acteur speelt. In dit geval is de titel van de film ook een representatie van de film.

Nu kan de gebruiker eventueel een "ranked list" terugsturen naar de database. Omdat de zoektermen al vrij direct waren geformuleerd, is de gebruiker tevreden met het resultaat. Enkele nieuwsberichten geven aan hoe succesvol enkele films van de acteur zijn. Nadat de

gebruiker de berichten heeft gelezen vraagt hij een complete film op bij de media server van de TV-zender.

2.2 Gebruikersmodel

Uit het voorbeeld in de vorige sectie blijkt dat er verschillende groepen gebruikers van digitale bibliotheken zijn. Wanneer we een multimedia database ontwerpen om digitale bibliotheken te ondersteunen, moeten we rekening houden met die gebruikersgroepen. We onderscheiden voor ons ontwerp de volgende groepen gebruikers:

Eindgebruikers	Eindgebruikers zoeken in de meta-data database en vragen media titels op bij de media servers. De eindgebruikers willen dat hun vragen zo snel mogelijk en zo goed mogelijk worden beantwoord.
Database administrator	De database administrator moet naast de taken die een database administrator heeft in een normale relationele database ook contacten onderhouden en contracten sluiten met de verschillende deelnemende partijen.
Database application developers	Database application developers zijn geavanceerde gebruikers die voor eindgebruikers een applicatie op maat kunnen bouwen. De database application developer kan geen wijzigingen in de structuur en werking van de multimedia database aanbrengen.
Content providers	Content providers bezitten multimedia titels. Ze verkopen de multimedia titels of ze stellen de multimedia titels beschikbaar. Content providers willen weten welke eindgebruikers gebruik maken van hun content.
Feature engineers	Feature engineers zijn specialisten op het gebied van feature extractie algoritmen. Ze zijn continu bezig met het ontwikkelen van nieuwe features. We verwachten dat feature engineers de door hun ontwikkelde algoritmen per licentie willen verkopen. Hun doel is, dat zoveel mogelijk content providers met hun features willen werken.
Database engineers	Database engineers zijn specialisten op het gebied van index structuren voor het opslaan van multimedia objecten en features. Ze leveren onderdelen (plug-ins en modules) voor databases waardoor die databases kunnen worden gebruikt als meta-data databases.
Annotators	Annotators zijn mensen die multimedia objecten uit een multimedia database annoteren, dat wil zeggen dat ze de multimedia objecten bekijken en handmatig classificeren.

2.3 Gebruikerseisen

Het succes van een digitale bibliotheek hangt onder meer af van de inhoud. Voor gebruikers is de collectie alleen interessant als er veel multimedia objecten in het systeem zijn opgenomen. Content providers willen graag zelf het beheer over hun titels voeren. Om zoveel mogelijk content providers over te kunnen halen om mee te doen, moet deze eis worden ingewilligd.

De beheerders van de meta-data zijn niet de eigenaren van de multimedia objecten. Bovendien stelt het afspelen van multimedia objecten hoge eisen aan het platform waarop dat plaatsvindt. Dit maakt dat het voordelig lijkt de meta-data en de multimedia objecten op aparte systemen op te slaan. Met traditionele monolithische databasesystemen is niet goed mogelijk daaraan te voldoen. De content providers zijn gebaat bij een groot aantal gebruikers. De multimedia database moet daarom goed schaalbaar zijn.

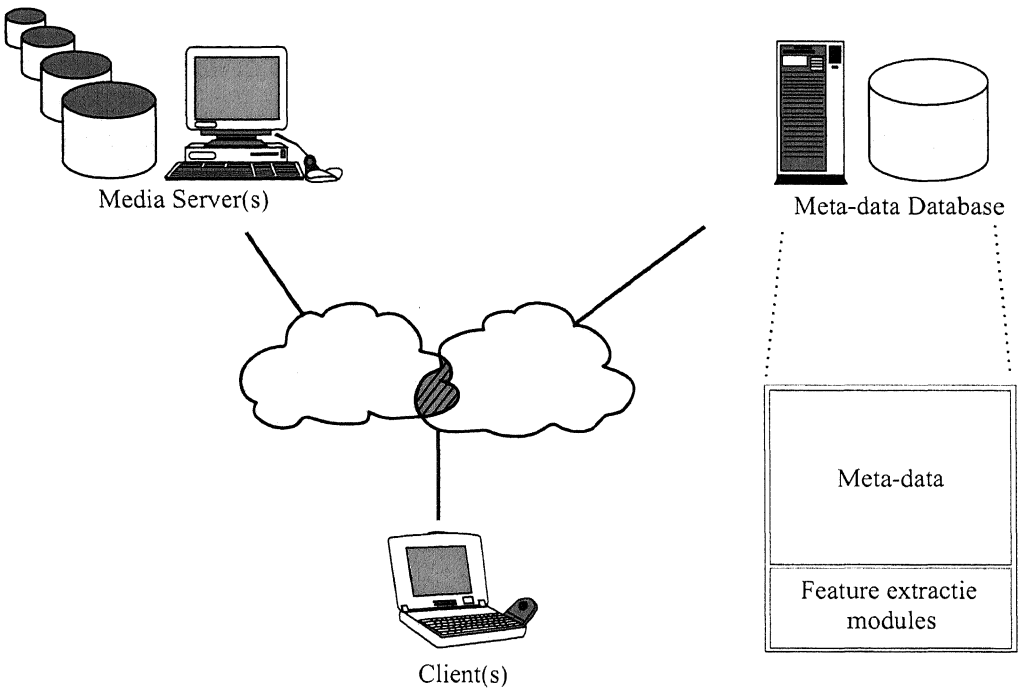
3 Probleemstelling en analyse

Aan de hand van de in de vorige sectie gepresenteerde achtergrond, formuleren we de volgende probleemstelling:

"Ontwerp een architectuur van een gedistribueerde multimedia database, ter ondersteuning van digitale bibliotheken, waarbij aan de eisen van de diverse gebruikersgroepen wordt voldaan. Het ontwerp moet een goede performance toestaan".

3.1 Architectuur

Wanneer we kijken naar de eisen van de gebruikers en de conclusies uit sectie 2.3, dan ligt een architectuur zoals weergegeven in figuur 3.1 voor de hand. Deze architectuur noemen we de centrale architectuur.



Figuur 3.1: Centrale architectuur

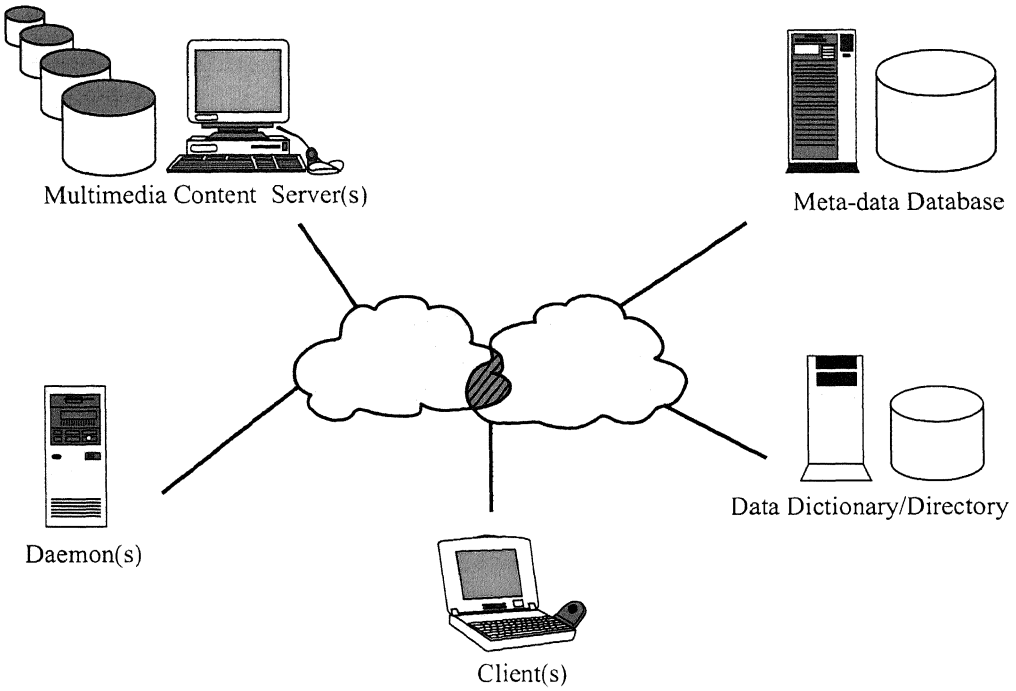
In figuur 3.1 is de splitsing aangegeven tussen de media servers (eigendom van de content providers) en de meta-data database. De gebruikers zijn in de eerste plaats gebruikers van de meta-data database. Pas nadat aan de hand van representaties een multimedia object is geselecteerd, wordt een media server aangesproken (zie het voorbeeld in sectie 2.1).

De feature extractie algoritmen kunnen als extensies in de meta-data database worden geladen. Gebruikers werken via een standaard client/server verbinding met de meta-data database.

De centrale architectuur heeft een aantal voordelen. Ten eerste wordt aan de belangrijkste gebruikerseis voldaan, namelijk de splitsing van multimedia objecten en meta-data. Alle feature extractie operaties worden intern in de meta-data database uitgevoerd. Voor de communicatie tussen deze operaties en de meta-data database hoeft het netwerk dus niet gebruikt te worden. Omdat feature extractie operaties een enorme rekenkracht kunnen vergen (doordat ze soms gebruik maken van complexe patroonherkenningsalgoritmen), is het moeilijk om een voorspelling te doen over het aantal gebruikers dat tegelijk met de meta-database kan werken. Sommige databases kunnen echter worden opgeschaald om meer gebruikers tegelijk met de database te kunnen laten werken. Een structurele oplossing is het opschalen niet, omdat, door de onregelmatige en onvoorspelbare belasting van de systemen door de feature extractie algoritmen, niet veel kan worden gegarandeerd met betrekking tot het aantal toegestane gebruikers.

Zoals we in sectie 2.2 hebben besproken, willen feature engineers dat zoveel mogelijk content providers met hun feature extractie algoritmen werken. In de centrale architectuur bepaalt de database administrator van de meta-data database welke feature extractie algoritmen worden gebruikt. De content providers zijn dus afhankelijk van de database administrator en ze kunnen niet investeren in algoritmen die alleen gebruikt worden om features van hun content te berekenen.

Om de deelnemers in de digitale bibliotheek meer vrijheid te geven en het systeem nog meer open te maken, introduceren we een volledig gedistribueerde architectuur. In figuur 3.2 is deze architectuur schematisch weergegeven. Een verschil met de centrale architectuur is, dat feature extractie algoritmen kunnen worden ondergebracht in aparte componenten die op een willekeurige plaats in het netwerk kunnen werken. Net als in [deVries98b] noemen we deze componenten daemons. Een operatie in een daemon heeft als invoer een multimedia object en als uitvoer een representatie (de representatie kan worden opgeslagen in de meta-data database) of een nieuw multimedia object (ingeval van bijvoorbeeld een compressie of conversie operatie). Een daemon kan worden beschouwd als een normale operatie die alleen wordt uitgevoerd op een andere locatie.



Figuur 3.2: Volledig gedistribueerde architectuur.

De daemons, de media-servers, de meta-data database en de clients zijn in de volledig gedistribueerde architectuur autonome componenten. Om van die componenten een gedistribueerde multimedia database te maken, moet er gemeenschappelijke kennis over het systeem worden bijgehouden. De gemeenschappelijke informatie bestaat uit kennis over welke media servers, media formaten, representaties en daemons in de gedistribueerde multimedia database aanwezig zijn. Deze informatie kan worden vergeleken met het schema van een relationele database. Het beheer van deze informatie maakt van de losse componenten een gedistribueerde multimedia database. Deze informatie wordt bijgehouden door de "data dictionary/directory".

Voor de communicatie tussen de media servers en de client applicaties, kan een voor het betreffende multimedia type geschikt stream protocol worden gebruikt. Een stream protocol is een protocol waarmee multimedia titels in real-time over een netwerk kunnen worden verzonden. De gedistribueerde database hoeft deze protocollen niet vast te leggen. Er kan door middel van overleg tussen een client applicatie en een media server overeen worden gekomen welk stream protocol gebruikt wordt. Voor de communicatie tussen de verschillende componenten in de gedistribueerde multimedia database kan gebruik worden gemaakt van een software bus. Er is veel onderzoek gedaan naar communicatie tussen componenten in gedistribueerde systemen en componentenarchitecturen [Omg98]. In sectie 4.1 wordt de software bus geanalyseerd.

Aan de volledig gedistribueerde architectuur kleven enkele nadelen ten opzichte van de centrale architectuur. In de gedistribueerde architectuur is bijvoorbeeld meer communicatie nodig tussen de verschillende onderdelen dan in de centrale architectuur. In de centrale architectuur vindt de communicatie tussen de feature extractie algoritmen – in de gedistribueerde architectuur de daemon componenten – en de meta-data database intern in de meta-data database plaats. Dit is aanzienlijk sneller dan de communicatie via een software bus. Ook is het beheer van een volledig gedistribueerd systeem lastiger dan het beheer van een centraal systeem.

De gedistribueerde architectuur kent echter ook enkele belangrijke voordelen. Ten eerste kunnen alle componenten op verschillende computers in een netwerk worden geïnstalleerd. Daemons die veel rekenkracht vergen, kunnen dan op zwaardere systemen worden geïnstalleerd, zodat de overige delen van het systeem niet worden vertraagd. Er is dus een zekere handmatige load-balancing mogelijk. Ten tweede kunnen er computers (en dus componenten) worden toegevoegd aan het systeem (door ze toe te voegen aan het netwerk). Op deze manier kunnen zeer grote, schaalbare, systemen worden gebouwd. Verder komt de volledig gedistribueerde architectuur goed overeen met de gebruikersanalyse. Iedere gebruikersgroep kan (met duidelijke specificaties van de interfaces van de componenten) zelfstandig componenten ontwikkelen. Feature engineers kunnen bijvoorbeeld de daemon componenten ontwikkelen. In de centrale architectuur zouden de feature engineers afhankelijk kunnen zijn van database engineers, omdat de feature extractie algoritmen dan geïntegreerd moeten worden in de meta-data database.

4 Componenten in de volledig gedistribueerde architectuur

We concluderen dat het volledig gedistribueerde ontwerp het beste aansluit bij het gebruikersmodel van de multimedia database. Hieronder bespreken we de componenten uit de volledig gedistribueerde architectuur.

4.1 Software bus

De software bus verzorgt de communicatie tussen de verschillende componenten van de gedistribueerde multimedia database. Een andere taak van de software bus is het met elkaar in contact brengen van de verschillende componenten door een name-space aan te bieden, waarvan de componenten gebruik kunnen maken. De software bus moet ook onder extreme situaties blijven werken.

4.2 Daemons

Een daemon component kan op eigen initiatief werk vragen aan de meta-data database. We noemen het opvragen van werk door een daemon een get_work query. Een get_work query heeft een drietal parameters: een multimedia type, een representatie en een conditie. De meta-data database levert alle objecten van het multimedia type op, waarvan de gegeven representatie niet in de meta-data database is opgenomen en waarvoor de conditie geldt. De daemon gaat dan van die multimedia objecten de betreffende representatie berekenen.

In [deVries98b] wordt onderscheid gemaakt tussen type-triggered en content-triggered daemons. Een type triggered daemon wordt geactiveerd voor elk multimedia object van een bepaald type. Een content-triggered daemon wordt geactiveerd voor multimedia objecten waarvoor een bepaalde voorwaarde op de inhoud geldt. Een gezichtherkenningsdaemon werkt bijvoorbeeld alleen op multimedia objecten waarop een gezicht te zien is.

4.3 Media servers

Media servers worden gebruikt door client- en daemon componenten. Deze componenten stellen verschillende eisen aan een media server. Ten eerste moeten media servers functies ondersteunen waarmee de multimedia objecten kunnen worden gelezen (als een random access file). De daemons lezen de multimedia objecten via de "random access" interface, omdat ze vaak kleine delen van het multimedia object nodig hebben en ze die niet real-time hoeven af te spelen. Ten tweede moeten de media servers ook multimedia objecten kunnen streamen (real-time versturen) naar client-componenten, zodat client componenten het multimedia object kunnen presenteren aan de eindgebruikers.

4.4 Meta-data database

Het ontwerp van een meta-data database is in ontwikkeling. In de meta-data database worden de locaties van de multimedia objecten op de media servers geregistreerd. Verder worden de bij de multimedia objecten horende features in de meta-data database opgeslagen.

4.5 Data dictionary/directory

De multimedia database kan niet goed functioneren zonder dat er gemeenschappelijke kennis wordt bijgehouden over de in het systeem aanwezige componenten en datatypen. De data dictionary/directory bewaart al deze gegevens. De data dictionary/directory werkt nauw samen met de meta-data database. Wanneer een nieuw multimedia datatype wordt toegevoegd aan de data dictionary/directory, maakt de data dictionary/directory nieuwe tabellen aan in de meta-data database.

4.6 Clients

Client componenten in de gedistribueerde multimedia database, hoeven geen bepaalde interface te implementeren. Er moet tijdens het ontwerpen en implementeren van clients wel rekening worden gehouden met het feit dat clients correct gebruik maken van de interfaces van de andere componenten in de gedistribueerde multimedia database.

5 Ontwerp

We kunnen de gedistribueerde multimedia database ontwerpen door gebruik te maken van gedistribueerde componententechnologie. De beschrijving van de eisen van de software bus komt sterk overeen met de specificatie van de CORBA architectuur [Omg98] [Orfali97]. Met CORBA is het mogelijk om op een hoog niveau operaties uit te voeren op de componenten. De data dictionary/directory en de meta-data database ontwerpen we aan de hand van een object-relationele database.

Voor de overige componenten is het voldoende om interfaces te definiëren in de CORBA Interface Definition Language (IDL). Programmeurs kunnen een interface implementeren en de implementatie opnemen in de multimedia database.

In deze sectie wordt als voorbeeld een specificatie gegeven van de meta-data database. In [Hof98] zijn ook de andere componenten gespecificeerd. Omdat de interfaces zijn ontworpen met de CORBA Interface Definition Language (IDL) [Omg98], kunnen de componenten worden geïmplementeerd in iedere programmeertaal die door CORBA wordt ondersteund.

5.1 Meta-data database

We gaan er in dit artikel vanuit dat de meta-data database gebouwd wordt als in [deVries98c]. De multimedia query processor bevindt zich dan tussen de interface en de onderliggende database. De interface is als het ware een "wrapper" om deze onderliggende database heen. Hieronder wordt de IDL code van de interface stap voor stap besproken. Op elk stukje specificatie volgt uitleg over de functionaliteit die daarmee ondersteund wordt.

```
typedef any media_object_descriptor;
```

Het type `media_object_descriptor` geeft de locatie van een media object op een media server aan. Het is niet belangrijk om te weten hoe een `media_object_descriptor` is opgebouwd. Het type `any` kan het type van een in IDL gespecificeerd type aannemen. Een voorbeeld van een `media_object_descriptor` is een URL.

```
struct query_result_item {
      any object;
      media_object_descriptor id;
};
```

De structure `query_result_item` geeft één resultaat object aan van een query. De gebruiker kan feedback geven op dit object. Niet het multimedia object zelf wordt opgeleverd, maar een representatie uit de meta-data database. De representatie kan van een willekeurig type zijn.

```
struct query_feedback_item {
      media_object_descriptor id;
      long  relevance;
};
```

Een `query_feedback` item wordt gebruikt door de client om relevance feedback te geven aan de database.

```
struct feature_query_item {
      string feature_type_id;
      any    value;
};
```

De structure `feature_query_item` wordt gebruikt om een directe query op een representatie uit te voeren. Hiermee kunnen clients, die kennis hebben over de gebruikte representaties, direct een representatie in de query gebruiken.

```
typedef sequence<query_result_item> query_result;
typedef sequence<query_feedback_item> query_feedback;
typedef sequence<media object descriptor> id_list;
```

Van de hierboven beschreven structures worden sequences gemaakt, zodat bij het uitvoeren van een query en het geven van relevance feedback meerdere gegevens tegelijk kunnen worden doorgegeven aan de database.

```
interface database_wrapper {

      query_result execute_similarity_query(in
                              id_list ids);
```

De methode `execute_similarity_query` voert een similarity query uit. De gegeven sequence van `media_object_descriptors (id_list)` geeft de objecten aan waarvoor gelijksoortige multimedia objecten moeten worden opgeleverd

```
      query_result relevance_feedback(in query_feedback f);
```

Met de functie `relevance_feedback` wordt door de gebruiker aangegeven welke multimedia objecten uit de voorgaande query-stap (de voorgaande `relevance_feedback` of de initiële query) relevant zijn. Aan de hand van de feedback berekent de meta-data database de query opnieuw.

De onderstaande vijf functies kunnen door clients worden gebruikt om door een feature ruimte te browsen. De functie `navigate_random` wordt gebruikt om op een willekeurig punt in de feature ruimte te beginnen.

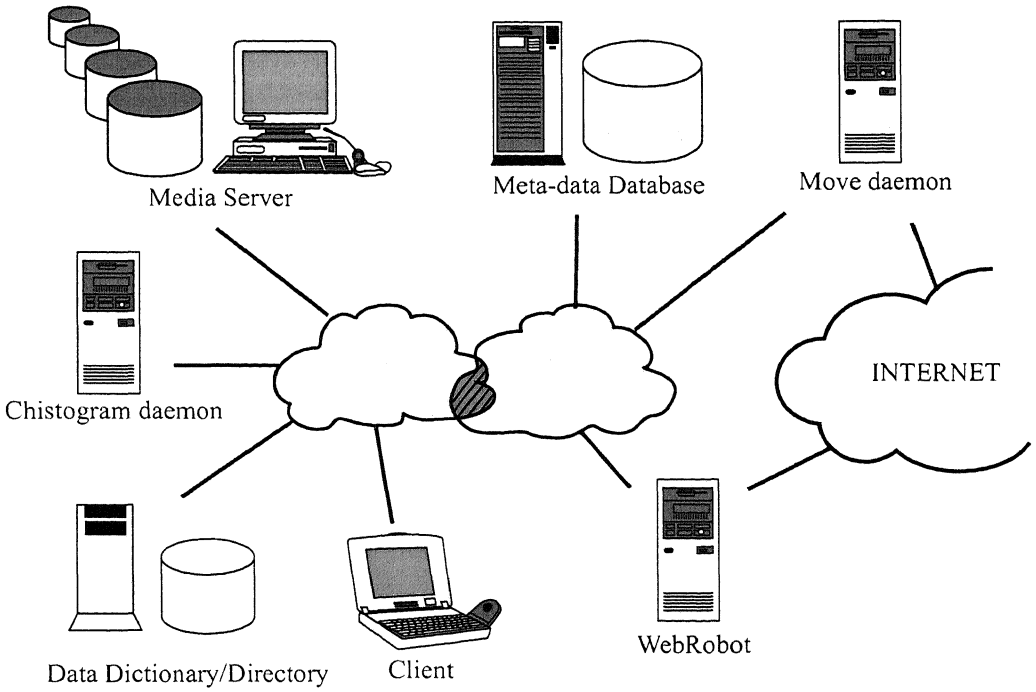
```
      void navigate_first(    in string representation_type_id,
                              out media_object_descriptor object,
                              out any representation);
      void navigate_next(     in long count,
                              in long representation_type_id,
                              out media_object_descriptor object,
                              out any representation);
      void navigate_prev(     in long count,
                              in long representation_type_id,
                              out media_object_descriptor object,
                              out any representation);
      void navigate_last(     in long representation_type_id,
                              out media_object_descriptor object,
                              out any representation);

      void navigate_random(   in long representation_type_id,
                              out media_object_descriptor object,
                              out any representation);

};
```

De gebruiker moet aangeven welke representatie gebruikt moet worden. Een initïele query wordt dus door de gebruiker ingevoerd door middel van een `navigate_*` of `execute_similarity_query`. De gebruiker kan vervolgens meerdere malen relevance feedback geven met de methode `relevance_feedback`.

6 Prototype

Om de bruikbaarheid van de opzet te testen, hebben we een prototype implementatie gemaakt van onze architectuur. In deze sectie bespreken we ons demonstratiesysteem.

In figuur 6.1 is het demonstratiesysteem schematisch weergegeven. De werking van het demonstratiesysteem is als volgt. De WebRobot daemon zoekt op het Internet naar plaatjes. De URLs van de plaatjes worden als representatie van het type "Image" in de meta-data database geplaatst. Dat wil zeggen dat voor ieder op het Internet gevonden plaatje de locatie van het plaatje op het Internet in de meta-data database wordt gezet. De URL is in dit geval een representatie en geen media_object_descriptor, omdat de WWW-servers op het Internet niet worden gebruikt als media servers. De WWW-servers worden gebruikt om multimedia data beschikbaar te stellen die in de gedistribueerde multimedia database wordt opgeslagen.



Figuur 6.1: Demonstratie gedistribueerde multimedia database

De "move-daemon" vraagt met bepaalde regelmaat aan de meta-data database of er Image objecten zijn waarvan wel een URL representatie is opgeslagen en geen media_object_descriptor. Dat zijn dus objecten die nog niet op een media server staan. De move-daemon kopieert deze plaatjes van het Internet naar een media server die deel uitmaakt van de gedistribueerde multimedia database. Vervolgens voegt de move-daemon een media_object_descriptor toe aan de meta-data database.

De Chistogram daemon vraagt met bepaalde regelmaat aan de meta-data database welke objecten wel in de meta-data database zijn opgeslagen met een media_object_descriptor en waarvan geen kleurenhistogram feature in de meta-data database aanwezig is. De Chistogram daemon leest die plaatjes van de media server en berekent het kleurenhistogram. Het kleurenhistogram wordt ingevoegd in de meta-data database.

De eindgebruiker kan via een client vragen stellen aan de meta-data database. De gebruiker kan afhankelijk van het resultaat van de vraag bij een media server het betreffende plaatje ophalen.
Om de interoperabiliteit van de verschillende CORBA compatibele ORBs te testen, hebben we een client geïmplementeerd als Java-applet. Bij de Netscape Navigator™ browser wordt standaard een CORBA compatibele ORB geleverd. Een applet kan gebruik maken van deze ORB. De door ons geïmplementeerde applet gebruikt de browser-ORB om contact te krijgen met de ORB van de gedistribueerde multimedia database. Zodoende kan een gebruiker via een browser en het Internet toegang krijgen tot de multimedia database.

De implementatie van de componenten is meestal eenvoudig, omdat de belangrijke delen van de componenten vaak al bestaan, zoals een database waarop de meta-data database wordt gebouwd en feature modules. Het implementeren komt dan neer op het gedistribueerd maken van deze componenten. Dit kan worden gedaan door een extra layer (wrapper) te bouwen tussen de gebruikte onderdelen en de software bus. Op deze manier hebben we de meta-data database geïmplementeerd door gebruik te maken van de Monet database server [Boncz97].

7 Conclusies en aanbevelingen

Om beter aan de eisen van gebruikers van digitale bibliotheken te voldoen, moet een multimedia database anders dan traditionele databases worden ontworpen. We hebben twee kandidaatarchitecturen met elkaar vergeleken en gekozen voor een volledig gedistribueerde architectuur. Met behulp van een CORBA software bus en de specificaties van de componenten kan een gedistribueerde multimedia database worden gebouwd. De onafhankelijkheden tussen de gebruikers blijven bestaan wanneer de gebruikersgroepen apart componenten kunnen maken voor de multimedia database. Dit kan doordat voor iedere gebruikersgroep een aparte IDL (interface) definitie beschikbaar is. Doordat, om een component te maken, alleen een interface hoeft te worden geïmplementeerd op (vaak) bestaande systemen, kunnen snel systemen worden gebouwd. Als voorbeeld is een demonstratie systeem gebouwd dat plaatjes kan verplaatsen van het Internet naar een media server en vervolgens meta-data van de plaatjes opslaat in een meta-data database.

Er zijn nog vele punten waaraan verder onderzoek besteed zou kunnen worden. Zo zou de meta-informatie die bekend is over de daemons kunnen worden gebruikt om de aanroepen van daemons te optimaliseren. Het eenvoudige model van een datatype met meerdere representaties kan worden uitgebreid naar een model dat is gebaseerd op een semantisch netwerk [deVries98d]. Er zou ook kunnen worden onderzocht of het mogelijk is om de verschillende representaties te classificeren. Misschien moet er onderscheid worden gemaakt tussen representaties die worden gebruikt om alleen te zoeken en representaties die (alleen) worden gebruikt voor presentatie aan gebruikers. De mogelijkheid om meerdere meta-data databases te gebruiken biedt misschien interessante perspectieven. Waarschijnlijk zorgt dit wel voor een grote toename van de complexiteit van het systeem. Misschien heeft het voordelen om speciale databases te maken met alleen meta-data van een bepaald multimedia type. De data dictionary/directory zou misschien ingebouwd kunnen worden in de database wrapper. De reden om het niet te doen is, dat de data dictionary/directory in principe een autonoom component is in het systeem. Een extra taak voor de data dictionary/directory zou het opstarten van het systeem kunnen zijn. Aan de andere kant is gebleken dat voor het implementeren van de data dictionary/directory de meta-data database kan worden gebruikt.

Daar komt bij dat de meta-data database altijd over alle informatie uit de data dictionary/directory moet kunnen beschikken om queries uit te voeren.

In een uitbreiding van dit ontwerp zou beter rekening moeten worden gehouden met de onderlinge relaties die kunnen ontstaan tussen verschillende componenten. Componenten van dezelfde leverancier zouden nog verborgen interfaces kunnen hebben. Componenten kunnen ook afhankelijk van elkaar zijn. De CORBA specificatie is nog steeds in ontwikkeling. Er worden steeds meer CORBA services gespecificeerd en geïmplementeerd. Veel van deze services lijken bijzonder geschikt om te worden gebruikt voor de implementatie van geavanceerde functies van de multimedia database.

8 Literatuur

[Amis] AMIS Project informatie:
http://www.cs.utwente.nl/~blokh/AMIS/

[Boncz97] Boncz, P., Wilschut, A. N., Kersten, M. L., 'Flattening an Object Algebra to Provide Performance', in *Fourteenth International Conference on Data Engineering*, pp. 568-577, (Orlanda, Florida), februari 1998.

[CNN] CNN Home Page
http://www.cnn.com

[Duda73] R.O. Duda, Hart, P.E., *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[Faloutsos96] Faloutsos, C., *Searching multimedia databases by content*, Kluwer Academic Publishers, Boston/Dordrecht/London, 1996.

[Hof98] Hof, E. van het, *Een Architectuur voor Multimedia Databases*, Afstudeerverslag, Universiteit Twente, 1998.

[Niblack93] Niblack, W., Barber, R., Equitz, W., Flickner, M., Glasman, E., Petkovic, D., Yanker, P., Faloutsos, 'The QBIC project: querying images by content using color, texture and shape'. Technical Report RJ 9203, IBM Research Division, 1993.

[Orfali97] Orfali,R., Harkey, D., *Client/Server Programming with JAVA and CORBA* John Wiley & Sons, 1997.

[Omg98] Object Management Group. *CORBA: The Common Object Request Broker: Architecture and Specification*, Release 2.2, maart 1998.

[QBIC] QBIC (Query By Image Content) informatie
http://wwwqbic.almaden.ibm.com/

[deVries98a] De Vries, A.P., Van der Veer, G., Blanken, H.M., 'Let's talk about it: Dialogues with multimedia databases. Database support for human activity', *Displays* **18** (4), pp. 215-220, 1998.

[deVries98b] De Vries, A.P., Eberman, B., Kovalcin, D.E., 'The design and implementation of an infrastructure for multimedia digital libraries', in *Proceedings of the 1998 International Database Engineering & Applications Symposium*, pp. 103-110, (Cardiff, UK), juli 1998.

[deVries98c] De Vries, A.P., Blanken, H.M., 'The relationship between IR and multimedia databases' in IRSG'98, (Autrans, France), maart 1998.

[deVries98d] De Vries, A.P., Blanken, H.M. 'Database technology and the management of multimedia data in Mirror', SPIE'98 (aangeboden).

[Virage98] Virage Image Engine
http://www.virage.com

[Wold96] Wold, E., Blum, T., Keisler, D., Wheaton, J., 'Content based classification, search and retrieval of audio'. *IEEE Multimedia*, 3 (**3**), 1996.