Sammendrag

Denne oppgaven ser på hvordan en XML-database kan brukes til indeksering av og søking i hierarkiske metadata. Dette inngår som en del av arbeidet med å gjøre informasjon fra forskjellige samlinger tilgjengelig for informasjonssøkere.

Problemstillingen det har vært arbeidet med er delt i to. Den ene delen var å finne ut hvordan en XML-database kan brukes som lokal indeks for metadata på et hierarkisk format. Den andre delen var å finne ut hvordan et søkegrensesnitt kan utvides til å utnytte hierarkiske søkemuligheter uten at det samtidig går ut over brukervennligheten i søkegrensesnittet.

Det er funnet ut at XML-databasen eXist egner seg godt til indeksering av og søking i hierarkiske metadata. Dette er funnet ut ved å utvikle en prototyp som basert på indeksering av tidligere innhøstede metadata fra forskjellige kilder tilbyr en felles søketjeneste. Datagrunnlaget i prototypen er tidligere innhøstede metadatabeskrivelser som kommer fra tre av Nasjonalbibliotekets samlinger. Totalt utgjør dette om lag 1400 metadatabeskrivelser fra Digitalt Radioarkiv, Galleri NOR og Mavis. Metadataene er beskrevet i formatet MODS (Metadata Object Description Schema). Etter en omforming, blir metadataene lagt inn i eXist. Søkingen gjøres i et webbasert søkegrensesnitt, som kobler seg til eXist.

Konklusjonen er at XML-databasen eXist kan anvendes som lokal indeks i en søketjeneste for informasjon fra mange forskjellige datakilder.

Oppgavetekst

Indeksering av og søking i hierarkiske metadata i XML-database

Informasjon fra forskjellige samlinger er ofte tilgjengelig for søking, men gjennom ulike søkegrensesnitt og ved hjelp av forskjellige søkemåter. Informasjonssøkere som søker i mange samlinger vil ofte være interessert i å kunne gjøre dette på ett sted, gjennom et felles søkegrensesnitt. Det vil være aktuelt å bruke et felles søkegrensesnitt for søkere som ikke vet så mye om kildene de søker etter informasjon i, og som ikke vil begrense seg til samlinger av en bestemt type (som for eksempel bilde, lyd, tekst). Et felles søkegrensesnitt er mindre aktuelt for søkere som vet hva de er ute etter, på hvilket format det foreligger og hvilke samlinger det finnes i.

Oppgaven bygger på arbeidet Eskil Høyen Solvang gjorde i sin hovedfagsoppgave i informasjonsforvaltning: ”Enhetlig tilgang til heterogene metadatabaser - Interoperabilitet v.h.a. OAI-PMH” [1]. Indekserings- og søkesystemet i denne løsningen var basert på søkemotoren og søkeprogramvaren Fast Data Search fra Fast Search & Transfer. Dette innebar en omforming av metadataene til metadataformatet FastXML, som har en flat struktur. Det skal lages et alternativt forslag til indekserings- og søkesystemet som ble foreslått i den oppgaven, basert på et hierarkisk metadataformat.

Denne oppgaven tar utgangspunkt i at metadata er samlet inn fra forskjellige samlinger og foreligger på et hierarkisk metadataformat. Selve oppgaven går ut på å konvertere og lagre metadataene lokalt på et felles format for å kunne søke i dem fra et felles søkegrensesnitt. Det skal benyttes et metadataformat som støtter hierarkiske metadata, derfor er MODS valgt. MODS uttrykkes ved hjelp av XML. For å kunne benytte seg av søketeknikker som utnytter XML-formatet og den hierarkiske strukturen skal metadataene lagres i en XML-database.

Egnetheten til XML-databasen og indeksering og søking i denne skal vurderes opp mot bruk av Fast Data Search.

Oppgaven går ut på å:

1. Lage en prototyp på lagring av, indeksering av og søking i hierarkiske metadata i en XML-database. For søkingens del inkluderer det mellomvare mellom søkegrensesnitt og XML-database.
2. Lage en prototyp på utvidet søkegrensesnitt basert på prototyp utviklet som del av [1].
3. Vurdere prototypens egnethet for søking sammenlignet med eksisterende implementasjon ved hjelp av Fast Data Search som beskrevet i [1].

Forord

Denne oppgaven er en masteroppgave utført siste semester i sivilingeniørstudiet ved Institutt for datateknikk og informasjonsvitenskap ved NTNU. Oppgaven er utført fra slutten av januar 2005 til innlevering 16. juni 2005.

Fagområdet er digitale bibliotek og oppgaven handler om interoperabilitet mellom forskjellige samlinger.

Oppgaven er skrevet av Knut Bjørke Ingebretsen, masterstudent ved Institutt for datateknikk og informasjonsvitenskap.

Veileder for oppgaven har vært Ingeborg Sølvberg. Øyvind Vestavik har bidratt med teknisk hjelp og kommentarer til rapporten. Det rettes herved en stor takk til begge for god veiledning og hjelp med oppgaven. Takker også Eskil Solvang som har vært behjelpelig med å svare på spørsmål og som stilte koden til søketjenesten til disposisjon.

Trondheim, 16. juni 2005
Knut Bjørke Ingebretsen

Innhold

Figurliste

1 Introduksjon til oppgaven

Digitale bibliotek kan inneholde samlinger av mange forskjellige slag. Tilgang til flere heterogene samlinger er ofte bare mulig ved å bruke forskjellige tjenester som er laget spesielt for en eller flere homogene samlinger. Det er et mål å kunne tilby brukerne tjenester som gjør det mulig å bruke én søkesyntaks og forholde seg til ett søkegrensesnitt når de søker etter informasjon i heterogene samlinger. Denne type interoperabilitet kan oppnås på forskjellige måter. En av disse er metadatahøsting.

Hovedfagsoppgaven ”Enhetlig tilgang til heterogene metadatabaser - Interoperabilitet v.h.a. OAI-PMH” [1] hadde som hovedmål å gjøre informasjon fra flere forskjellige samlinger lettere tilgjengelig for brukerne. Den oppgaven ble utført i samarbeid med Nasjonalbiblioteket.

”Nasjonalbiblioteket har mer enn 80 forskjellige databaser/samlinger innen en rekke ulike emner. Beskrivelser av disse databasene og mulighetene for å søke i dem er tilgjengelig for publikum, blant annet gjennom world wide web (...). En vanlig bruker er normalt interessert i å finne informasjon - i form av dokumenter - uavhengig av hvordan dokumentene er beskrevet ved hjelp av metadata og hvor dokumentene er lagret. I dag er imidlertid objektene i de forskjellige databasene beskrevet ved hjelp av mange ulike metadataformater, og brukeren må dessuten selv finne ut hvilken database som er mest relevant å søke i før han/hun utfører selve søket. For uerfarne brukere kan det derfor være problematisk å orientere seg i Nasjonalbibliotekets databaselandskap.” [1]

Problemstillingen i [1] var hvordan man kan utforme en arkitektur som muliggjør samtidig søk i flere heterogene metadatabaser, og hvordan man kan bygge opp et søkegrensesnitt som nyttiggjør seg denne arkitekturen slik at man gjennom søkegrensesnittet får tilgang til ressursbeskrivelser fra alle de heterogene metadatabasene.

Arkitekturen som ble foreslått forutsatte bruk av teknologiene:

- OAI-PMH (Open Archives Initiative Protocol for Metadata Harvesting) [2]
- Fast Data Search [3]

Nasjonalbiblioteket fylte rollen som datatilbyder i OAI-PMH-terminologi. Hver enkelt av databasene/samlingene hos Nasjonalbiblioteket var en datatilbyder, mens søkegrensesnittet var en tjenestetilbyder.

Videre var arkitekturen basert på søkemotoren og søkeprogramvaren Fast Data Search fra Fast Search & Transfer [4]. En OAI-høster som samlet inn metadata fra tjenestetilbyderne lagret disse lokalt i filer. For å kunne søke i de innhøstede metadataene ble dataene indeksert av Fast Data Search. Fast sitt indekseringsprogram indekserer kun data som foreligger i formatet FastXML, som er et XML-format med en flat struktur. Det vil si at når metadata på det opprinnelige metadataformatet omformes til FastXML blir alle eventuelle hierarkiske strukturer konvertert til en flat struktur, noe som kan føre til tap av informasjon. Dette kom fram i evaluering og konklusjon i [1].

Fordelen med å bruke Fast Data Search er at det er et komplett system for indeksering og søking som har en mengde søkemuligheter. Dessuten har systemet høy søkeytelse. Problemet er at Fast Data Search ikke er spesielt laget for å søke i data som er strukturerte, slik data lagret i et metadataformat er. Fast har laget en løsning som effektivt skal kunne

gjenfinne informasjon i store samlinger med data, kan søke i mange forskjellige filformater og som er skalerbar. Fast Data Search er altså et kraftig verktøy for allsidig søk, men ikke spesielt beregnet på metadata.

Hensikten med denne oppgaven er å undersøke om det er mulig å bruke en XML-database som lokal søkeindeks i stedet for Fast Data Search, og vurdere fordeler og ulemper ved dette.

1.1 Problembeskrivelse

Hovedmålet med denne oppgaven er å foreslå en alternativ måte å implementere søk på, der metadataenes hierarkiske struktur kan utnyttes. Ved å bruke et allment tilgjengelig metadataformat uttrykt i XML åpnes muligheten for å bruke et stort utvalg av databaser til lokal indeksering av metadataene. Muligheten for å bruke en ”native” XML-database nevnes under alternative løsningsmetoder i avsnitt 7.5.2 i [1]. Her påpekes det at fordeler ved å bruke en XML-database er bedre utnyttelse av opprinnelig XML-struktur og en effektiv søkemekanisme. Søk hvor som helst i XML-dokumentene er effektivt i en XML-database.

En native XML-database har den fordelen at standardiserte språk slik som XQuery [5] og XPath [6] kan brukes til å søke i XML-dokumenter som ligger i databasen.

Problemstillingen kan oppsummeres slik:

- Hvordan kan en XML-database brukes til lokal indeksering av metadata på et hierarkisk format slik at det gir større muligheter for søk enn ved indeksering av metadata med flat struktur?
- Hvordan kan et søkegrensesnitt utvides til å utnytte disse mulighetene uten at det samtidig går ut over brukervennligheten til søkegrensesnittet?

Søkegrensesnittet utviklet i [1] er grunnlaget for søkegrensesnittet som brukes i denne oppgaven.

1.2 Forutsetninger

Følgende forutsetninger legges til grunn for oppgavens løsning:

- Innhøstede metadata foreligger i XML-filer med metadataposter på MODS-format [19].
- Metadataformatet MODS brukes til lokal indeksering av metadataene i en XML-database.

De metadataene som skal benyttes er de samme som ble brukt i [1]. Fordi disse metadataene allerede er høstet med en OAI-høster finnes de i XML-filer med metadataposter på MODS-format.

1.3 Rapportens oppbygning

Resten av rapporten består av:

- Kapittel 2 som tar for seg forskjellige teknologier brukt i oppgaven
- Kapittel 3 som går gjennom tidligere arbeid relatert til denne oppgaven
- Kapittel 4 som er systemspesifikasjonen, der kravene til prototypen etableres
- Kapittel 5 som beskriver implementasjonen av prototypen

- Kapittel 6 som består av testing av søketjenesten. Søketjenesten er en del av prototypen
- Kapittel 7 som er en evaluering av den utviklede prototypen
- Kapittel 8 inneholder konklusjon på oppgaven

2 Relevante teknologier brukt i oppgaven

Dette kapittelet beskriver teknologier som er brukt i oppgaven. Metadataformater er et grunnleggende begrep innen digitale bibliotek og anvendes i denne oppgaven for å ensrette tilgangen til metadatabeskrivelser fra forskjellige samlinger. Derfor er metadataformatet MODS beskrevet i kapittel 2.1. Interoperabilitetsprotokoller er en måte å hente inn metadata fra forskjellige kilder på slik at de kan brukes i ett system. OAI-PMH er en slik protokoll, den beskrives i kapittel 2.2. En del av begrepene som brukes i OAI-PMH brukes også andre steder i denne oppgaven, derfor er det på sin plass med en beskrivelse av protokollen. Til slutt i kapittelet kommer en beskrivelse av eXist [7], som er en XML-database. Her beskrives de egenskapene ved eXist som danner grunnlaget for avgjørelser tatt i kapittel 4.1.3.

2.1 MODS

Det skal brukes et felles hierarkisk metadataformat for lagring av metadata i XML-databasen. Med utgangspunkt i vurderingene i [1] ser det ut som MODS (Metadata Object Description Schema) er det metadataformatet som egner seg best. MODS bruker XML-syntaks for å uttrykke metadatabeskrivelser. XML egner seg bra til å uttrykke metadata i MODS-format fordi oppbygningen er hierarkisk. Nedenfor følger en nærmere beskrivelse av MODS.

MODS (Metadata Object Description Schema) inneholder et subsett av MARC-elementene. Det kan ses på som en forenklet utgave av MARC, som bruker språkbaserte tagger i stedet for tallbaserte tagger. MODS er i følge [8] spesielt godt egnet til digitale biblioteksobjekter som krever utfyllende metadatabeskrivelser som er kompatible med eksisterende beskrivelser i bibliotekskataloger, men likevel ikke så kompliserte som MARC.

Grunnen til at MODS ble utviklet er i følge [8] et behov for en forenklet XML-versjon av MARC 21. Det har blitt uttrykt behov for å gå over til XML for metadata i bibliotek og andre kulturelle institusjoner. MODS er i følge [8] ment å fylle behovet for et alternativ mellom et enkelt metadataformat med et minimum antall felt og liten understruktur, slik som Dublin Core, og et veldig detaljert format med mange dataelementer og komplisert struktur, slik som MARC 21. MARC har mange og kompliserte dataelementer. I tillegg brukes tallbaserte tagger som er vanskelig å huske, og derfor ikke spesielt godt egnet for manuell redigering. Dublin Core Metadata Initiative er eksempel på en løsning på dette problemet, men Dublin Core er ment å dekke et større spekter enn MARC 21. Som en videreutvikling av MARC lanserte Library of Congress MODS i juni 2002. Siste versjon per i dag, MODS schema version 3.0, kom i desember 2003. Imidlertid brukes versjon 2 i denne oppgaven, fordi metadataene som skal brukes følger den versjonen.

Oppbygningen av MODS er hierarkisk, med 19 elementer på toppnivå. 10 av disse har underelementer, mens 9 ikke har underelementer, bare attributter. Det er for en stor del underelementer og attributter som inneholder dataene. Toppelementer, underelementer og attributter er listet opp i en oversikt på [9].

Tabell 2-1 - Toppnivåelementer i MODS

Metadataelement	Definisjon
titleInfo	titleInfo er et innpakningselement (wrapper element) som

	inneholder alle underelementene relatert til tittelinformasjon
name	name er et innpakningselement for navn som er assosiert med ressursen
typeOfResource	typeOfResource beskriver ressurstypen, for eksempel tekst, lyd eller bilde
genre	genre beskriver mer spesifikt enn typeOfResource hvilken kategori ressursen tilhører
originInfo	originInfo er et innpakningselement som inneholder alle underelementene relatert til publikasjons- og opprinnelsesinformasjon
language	language beskriver språket ressursen benytter, dette kan også beskrives i hvert element i MODS ved å bruke xml:lang- og/eller lang-attributtene
physicalDescription	physicalDescription er et innpakningselement som inneholder alle underelementene relatert til fysisk beskrivelse av ressursen
abstract	abstract inneholder et sammendrag av ressursen
tableOfContents	tableOfContents inneholder innholdsfortegnelse for ressursen
targetAudience	targetAudience beskriver nivået til publikumet som ressursen er beregnet på
note	note inneholder enhver bemerkning relatert til ressursen
subject	subject er et innpakningselement som binder sammen underelementer
classification	classification inneholder klassifikasjonsnummeret til ressursen
relatedItem	relatedItem inneholder informasjon som identifiserer en relatert ressurs, der attributtet type spesifiserer typen relasjon
identifier	identifier inneholder et unikt standardnummer eller en standardkode som identifiserer ressursen
location	location identifiserer institusjonen eller datalageret som holder ressursen, eller en URL der ressursen er tilgjengelig
accessCondition	accessCondition inneholder informasjon om restriksjoner som er lagt på tilgang til ressursen
extension	extension inneholder ekstra informasjon som ikke dekkes av andre elementer i MODS
recordInfo	recordInfo er et innpakningselement som inneholder underelementer med informasjon som er nødvendig for administrering av metadata

Eksempel på en metadatabeskrivelse i MODS (namespace er utelatt for å øke lesbarheten):

```
<mods>
  <titleInfo>
    <title>
    Lundefuglbestanden på Røst. Intervju med ornitologen Gunnar Lid
    </title>
  </titleInfo>
  <name>
    <namePart>Røisli, Inger Johanne</namePart>
    <role>
      <text>programleder</text>
    </role>
  </name>
  <typeOfResource>sound recording</typeOfResource>
  <genre>Radioprogram</genre>
```

```
  <originInfo>
    <publisher>NRK</publisher>
    <dateCreated>1983-08-11</dateCreated>
  </originInfo>
  <physicalDescription>
    <extent>00:03:50</extent>
  </physicalDescription>
  <abstract>
  K598 - Fugler + K333.7 - Naturvern + K948.441 - RØST - Programleder Inger
  Johanne RØISLI. Om lundefugler. Gunnar LID (mv) (på Røst): Stor begivenhet: For
  første gang siden 1974 nok mat i havet til at lundeungene kan vokse opp.
  Observasjoner av den første ungen på vei til havet - flere vil komme.
  Kjønnsmodne først etter 5-6 år. Vanskelig å si om de vil overleve sultne måker
  og andre farer. Zoolog Tycho ANKER-NILSSEN (mv): Lundeungene lever av sil
  (tobis). På Røst går det med opptil 15 millioner sil pr. døgn. –
  </abstract>
  <identifier type="uri">URN:NBN:no-nb_drl_20284 3424</identifier>
  <accessCondition>NRK</accessCondition>
</mods>
```

2.2 Open Archives Initiative Protocol for Metadata Harvesting

Open Archives Initiative Protocol for Metadata Harvesting (forkortet OAI-PMH) [2] er en standard som definerer et rammeverk for metadatahøsting. Rammeverket inneholder to typer deltakere: Datatilbydere og tjenestetilbydere. Datatilbydere gjør metadata tilgjengelig for høsting. Tjenestetilbydere høster metadata og bruker disse som grunnlag for tjenester.

2.2.1 Begreper

OAI-PMH har noen grunnleggende begreper som ligger i bunn. Disse begrepene er viktige for å kunne forklare hvordan protokollen virker.

Høster

En høster (harvester) er en klientapplikasjon som sender ut OAI-PMH-henvendelser. Høsteren drives av en tjenestetilbyder og samler inn metadata fra datalagre hos datatilbyderen.

Datalager

Et datalager (repository) er en server som kan prosessere de seks OAI-PMH-henvendelsene som er beskrevet i kapittel 2.2.2. Datalagre styres av datatilbydere slik at metadata gjøres tilgjengelig for høstere. OAI-PMH skiller mellom tre forskjellige entiteter i forbindelse med metadata som tilgjengeliggjøres ved hjelp av OAI-PMH.

- *Ressurs (resource)* – En ressurs er objektet metadataene beskriver. Hvordan en ressurs er lagret er utenfor definisjonsområdet til OAI-PMH.
- *Dataelement (item)* – Et dataelement inngår i et datalager og inneholder eller dynamisk genererer metadata for en enkelt ressurs i ulike formater. Disse metadataene kan høstes som poster ved hjelp av OAI-PMH. Hvert element har en unik identifikator innenfor definisjonsområdet til datalageret det tilhører.
- *Post (record)* – En post er metadata i et spesifikt metadataformat. En post returneres som en XML-kodet bytestrøm som respons på en protokollhenvendelse om å hente metadata fra et dataelement.

Unik identifikator

En unik identifikator (unique identifier) identifiserer utvetydig et dataelement i et datalager. Identifikatoren brukes i OAI-PMH-henvendelser når metadata skal hentes ut fra dataelementer. Dataelementer kan inneholde metadata i flere ulike formater. Den unike

identifikatoren peker til dataelementet og alle postene deler den samme unike identifikatoren.

Formatet til den unike identifikatoren må stemme overens med formatet til URI-syntaksen (Uniform Resource Identifier). Unike identifikatorer brukes både i responser på henvendelser og i henvendelser der det spørres etter en post i et spesifikt metadataformat.

Post

En post (record) er som nevnt metadata i et spesifikt metadataformat. En post har en utvetydig definisjon i form av kombinasjonen av den unike identifikatoren til dataelementet posten er tilgjengelig fra, `metadataPrefix` som identifiserer metadataformatet til posten og datostempelet til posten. XML-kodingen av en post består av følgende deler:

- *Header* – Inneholder den unike identifikatoren til dataelementet og andre egenskaper som er nødvendige for selektiv høsting.
- *Metadata* – En enkelt manifestasjon av metadataene til et dataelement. Minimumskravet til et datalager er at det kan returnere poster der metadataene er uttrykt i Dublin Core, uten kvalifikasjon. I tillegg kan andre metadataformater tilbys, for eksempel MODS.
- *About* – Dette er en valgfri og repeterbar del av posten som holder data om metadata-delen av posten.

Sett

Et sett (set) er en valgfri inndeling av datalagrene, der hensikten er å gjøre det mulig med selektiv høsting.

Selektiv høsting

Selektiv høsting vil si at metadatahøstere kan avgrense henvendelsene sine til deler av metadataene som er tilgjengelig fra et datalager. Det finnes to typer høstekriterier i OAI-PMH, disse kan kombineres. Det er datostempel og settmedlemskap. Datostempel forteller når en metadatapost ble lagt til eller oppdatert. Dermed kan man velge å ikke høste de postene som har et datostempel som tilsier at posten ikke er forandret siden forrige høsting. Settmedlemskap forteller hvilket sett en metadatapost hører til. Brukes gjerne til å dele opp postene etter samlingene de kommer fra.

2.2.2 Protokollens henvendelser og responser

OAI-PMH har noen definerte henvendelser, kalt verb, som høsteren sender til datatilbyderen.

- GetRecord
- Identify
- ListIdentifiers
- ListMetadataFormats
- ListRecords
- ListSets

Nærmere beskrivelse av verbene følger for å forstå hvordan protokollen virker.

GetRecord

GetRecord brukes til å hente en enkelt metadatapost fra et datalager. Identifikatoren til posten og metadataformatet er påkrevde argumenter.

Identify

Identify brukes til å hente informasjon om et datalager. Verbet har ingen argumenter. Informasjonen man får er blant annet navn på datalageret, base-URL-en til datalageret og protokollversjonen av OAI-PMH som datalageret bruker.

ListIdentifiers

ListIdentifiers brukes til å hente bare headerene til postene og ikke selve metadataene. Metadataformat er påkrevd argument, mens fra, til og sett er valgfrie argumenter som brukes til å oppnå selektiv høsting.

ListMetadataFormats

ListMetadataFormats brukes til å hente informasjon om hvilke metadataformat som er tilgjengelig fra et datalager. Identifikatoren til en bestemt post kan gis som argument for å finne ut hvilke metadataformat den er tilgjenglig i.

ListRecords

ListRecords brukes til å høste poster fra et datalager. Man får da hele posten, både header og metadata. Valgfrie argumenter kan brukes for å få til selektiv høsting basert på settmedlemskap og/eller datostempel.

ListSets

ListSets brukes til å hente settstrukturen til datalageret. Man får da beskrivelser av settene som finnes i datalageret, dersom datalageret støtter sett.

2.3 eXist

”Native” XML-databaser lagrer XML-dokumenter i sin helhet rett inn i databasen, uten fragmentering [10]. Dette i motsetning til en vanlig relasjonsdatabase som lagrer data i tabeller. XML-databaser lagrer dokumentene med innhold, elementer og attributter i sin originale form, og utnytter den strukturen som allerede ligger i XML-dokumentene. Forskjellige indekseringsteknikker gjør at søking hvor som helst i dokumentene kan gjøres effektivt.

Generelt sett er fordelen med å bruke en XML-database at innhøstede metadatabeskrivelser foreligger på XML-format. Dette gjør at man slipper å konvertere til et annet dataformat. En ulempe med denne løsningen er relativt lav ytelse ved oppdatering av innholdet, fordi indeksen må opprettes på nytt ved hver oppdatering. Siden ytelsen ikke er av stor betydning for denne prototypen vil det ikke bli tatt hensyn til dette potensielle ytelsesproblemet.

eXist er en native XML-database basert på åpen kildekode. Den er utviklet i Java og kan lett installeres på forskjellige systemer som kan kjøre Java-programmer. Noen av de viktigste egenskapene til eXist er indeksbasert XQuery-prosessering, automatisk indeksering og utvidelser for fulltekstsøk.

2.3.1 XQuery

eXist har sin egen XQuery-motor (engine) som implementerer XQuery working draft fra november 2003, med unntak av noen deler som har med XML Schema å gjøre. XQuery-motoren kan behandle både XPath- og XQuery-kode og skiller ikke på syntaksen mellom disse internt. eXist implementerer det meste av XQuery-spesifikasjonen, det som ikke støttes er det ikke bruk for i denne oppgaven. XQuery-motoren er indeksbasert og er derfor effektiv. I stedet for topp til bunn eller bunn til topp tretraversering brukes path join-algoritmer for å beregne relasjoner mellom noder. Dataene er lagret i B+-trær.

Det finnes en del ekstra funksjoner og to ekstra operatorer som tillegg til standard XQuery i eXist. Disse utvidelsene kan deles inn i flere kategorier, og det er to kategorier som er interessante for denne oppgaven:

- Funksjoner for å spesifisere dokumentsettet som er inndata til et uttrykk
- Utvidelser for fulltekstsøk

Funksjonene collection() og doc() er standardfunksjoner i XQuery, mens xcollection() og document() er spesifikke for eXist. doc() kan bare ta ett argument til en dokument-URI, mens document() kan ta flere argumenter slik at flere dokument-URI-er kan spesifiseres. collection() spesifiserer samlingen som inneholder de dokumentene som skal være med i spørringen. Dokumenter i undersamlinger av den spesifiserte samlingen blir også tatt med. xcollection() derimot tar bare med de dokumentene som finnes i den spesifiserte samlingen og ikke dokumenter i eventuelle undersamlinger.

De utvidelsene som er mest interessante for denne oppgaven er utvidelsene for fulltekstsøk. Det vil si fulltekstsøk innefor bestemte elementer i XML-dokumentene. Det finnes en rekke standard strengmanipuleringsfunksjoner i XPath/XQuery sitt funksjonsbibliotek, men disse er i seg selv ikke tilstrekkelig for søk etter nøkkelord eller fraser i en stor mengde tekst. Derfor finnes det to operatorer som kan brukes til nøkkelordsøk, &= og |=.

Bruk av operator	**Forklaring**
nodesett &= 'streng med nøkkelord'	Dette uttrykket velger kontekstnoder som inneholder alle nøkkelordene i høyre argument i en hvilken som helst rekkefølge. Det brukes en tokenizer for å splitte opp nøkkelordstrengen i enkeltord, samtidig som tegnsetting og mellomrom fjernes. Sammenligningen er case-insensitiv.
nodesett |= 'streng med nøkkelord'	Dette uttrykket velger kontekstnoder som inneholder hvilke som helst av nøkkelordene i høyre argument i en hvilken som helst rekkefølge. Det vil si at ett eller flere nøkkelord fra høyre argument må finnes i en av kontekstnodene for å få treff. Ellers virker den som operatoren ovenfor.

Begge operatorene har mulighet for å bruke wildcards, der ? matcher ingen eller et tegn, mens * matcher ingen eller flere tegn.

Det finnes også en eXist-spesifikk funksjon som heter near(). Den er ganske lik &=, men i tillegg tar den hensyn til rekkefølgen av nøkkelordene og avstanden mellom dem i dokumentet. near() kan derfor brukes til frasesøk. Man kan spesifisere maksimal avstand

mellom nøkkelordene slik at man kan finne ord som står i nærheten av hverandre i dokumentet. Standardverdien til denne maksimalavstanden er 1, det betyr at ordene står inntil hverandre og utgjør en frase.

Funksjonene match-all() og match-any() ligner på &= og |=, men de tolker argumentene som regulære uttrykk.

2.3.2 Indekser

Det finnes tre typer indekser i eXist.

- Strukturell indeks
- Fulltekstindeks
- Områdebegrenset indeks (range index)

Strukturell indeks gjør at strukturen i dokumentene kan navigeres raskere. Enkle XPath- og XQuery-uttrykk bruker denne indeksen. Strukturell indeks styres automatisk av eXist.

Fulltekstindeks brukes av fulltekstutvidelsene til eXist. Dette omfatter operatorene &= og |=, og funksjonene near(), match-all() med flere. Standardoppsettet av eXist fører til at alle noder fulltekstindekseres. Det er også mulig å velge bort eller legge til spesifiserte deler av et dokument som skal/ikke skal indekseres.

Områdebegrenset indeks gjør at eXist kan velge noder direkte ut fra datatypen til verdiene deres. Disse indeksene lages når dokumentet lastes og oppdateres automatisk ved eventuelle senere oppdateringer av dokumentet.

Alle verdier inni elementene i dokumentene som er lagret i databasen er i utgangspunktet tekststrenger. Dersom et dokument for eksempel inneholder et element som heter <pris> og innholdet i dette elementet er et flyttall, kan det defineres en indeks på pris med typen xs:double. Dette fører til at eXist vil prøve å konvertere alle <pris>-verdier til xs:double under indeksering og legge disse verdiene i den områdebegrensede indeksen. Et uttrykk som sammenlikner <pris> med en flyttallsverdi vil bruke denne indeksen og spare spørringsmotoren for konverteringsarbeid.

2.3.3 Kjøremodi for databasen

eXist-databasen kan kjøres på grovt sett tre forskjellige måter:

- stand-alone serverprosess
- innebygd (embedded) i en annen applikasjon
- i en servletkontekst

Stand-alone-modus betyr at eXist kjører i sin egen Java virtual machine. Dette er en fordel fordi det gjør at databasen ikke er avhengig av andre tråder som kan skape problemer, og blir derfor mer pålitelig i følge [11]. Dersom man ikke trenger SOAP, WebDAV eller Cocoon er stand-alone-modus å foretrekke. Klienter må koble seg til databasen gjennom nettverket, ved hjelp av enten XML-RPC-protokollen eller REST-style HTTP API-et. For å gjøre det enklere å aksessere databasen fra Java finnes det et XML:DB API. Dette grensesnittet tilbyr metoder som bruker XML-RPC slik at man kan kommunisere med databasen på et litt høyere abstraksjonsnivå.

Innebygd (embedded) modus vil si at databasen kontrolleres av klientapplikasjonen. eXist kjører i den samme Java virtual machine som klienten. Dermed trengs ingen

nettverksforbindelse mellom klienten og databasen og klienten har full tilgang til databasen.

Dersom eXist kjøres i en servletkontekst brukes databasen som en del av en webapplikasjon. Dette er slik databasen kjøres som standard dersom man starter den med medfølgende konfigurasjon i henhold til hurtigstartguiden til eXist. Servleter som kjører i samme webapplikasjonskontekst vil ha direkte tilgang til databasen. Eksterne klientapplikasjoner kan også koble seg til databasen ved å bruke nettverksgrensesnittene som tilbys i eXist.

2.3.4 Tilkobling til databasen

Som nevnt ovenfor er det flere måter å koble til databasen på. Hvilke muligheter som er tilgjengelig avhenger av modusen databasen kjøres i. Den viktigste databasetilkoblingen i denne oppgaven er den som skjer fra mellomvaren i søketjenesten. Siden det er spørringer vil det være naturlig å sende disse som XQuery-spørringer til databasen. XML-RPC API-et har metoder for å sende XQuery/XPath til databasen. Det finnes bibliotek for å bruke XML-RPC fra webapplikasjonsspråk som CGI-skript, PHP, JSP med flere. En annen mulighet er å bruke et SOAP-grensesnitt som eXist tilbyr. SOAP kan være noe lettere å bruke enn XML-RPC fordi man ikke trenger å skrive metodekallene selv. eXist genererer en WSDL servicebeskrivelse som forteller hvilke metoder som er tilgjengelig og gjør det mulig for klienten å generere koden som trengs for å kalle metodene. Det finnes bibliotek for å kommunisere over SOAP for eksempel i PHP. Det er også laget noen PHP-klasser [14] som bruker et slikt SOAP-bibliotek og som er beregnet spesielt på eXist.

3 Tidligere arbeid

Dette kapittelet ser på relatert arbeid som er gjort tidligere. Ved å knytte denne oppgaven til tidligere utført arbeid ser man lettere hva som er bakgrunnen for å gjøre denne oppgaven. Kapittelet består av to deler:

- Omtale av hovedfagsoppgave: ”Enhetlig tilgang til heterogene metadatabaser”
- Fulltekst kontra strukturbasert tekst

3.1 Enhetlig tilgang til heterogene metadatabaser

Denne oppgaven tar utgangspunkt i tidligere arbeid gjort av Eskil Høyen Solvang i hans hovedfagsoppgave: ”Enhetlig tilgang til heterogene metadatabaser - Interoperabilitet v.h.a. OAI-PMH” [1]. Hovedmålet for den oppgaven var å gjøre informasjon fra forskjellige samlinger lettere tilgjengelig for brukerne. En viktig del av dette var å bygge opp et enhetlig søkegrensesnitt som gir brukerne tilgang til ressursbeskrivelser fra alle metadatabasene de ellers måtte forholdt seg direkte til.

Oppgavens konklusjon slår fast at den opprinnelige målsetningen ble oppfylt. Videre ble det vist at det kan utvikles en arkitektur som muliggjør søking i metadata fra heterogene metadatabaser ved hjelp av OAI-PMH. Det ble utviklet et søkesystem basert på metadata innhøstet ved hjelp av metadatahøsting. Et interessant funn var at metadataformatet MODS kan brukes som felles metadataformat. Det ble også konkludert med at Fast Data Search kan brukes som søke- og indekseringsløsning.

Som nevnt i introduksjonen (se kapittel 1) og ovenfor bruker oppgaven ”Enhetlig tilgang til heterogene metadatabaser” Fast Data Search og som en følge av det formatet FastXML til indeksering. For å illustrere det problemet som ble omtalt i introduksjonen kan vi sammenligne en metadatabeskrivelse på MODS-format med tilsvarende metadatabeskrivelse omformet til FastXML. Nedenfor vises først et utdrag av en metadatabeskrivelse i MODS-format hentet fra Digitalt radioarkiv (se kapittel 4.1.1). Utdraget inneholder en tittel og to navn knyttet til radioopptaket. Den ene personen var produsent og den andre var programleder for programmet/innslaget.

```
<mods>
   <titleInfo>
      <title>Ukens gjest: Næringsminister Finn Kristensen Ekko</title>
   </titleInfo>
   <name>
      <namePart>Havgar, Kristin (prod.)</namePart>
      <role>
         <text>produsent</text>
      </role>
   </name>
   <name>
      <namePart>Solli, Jens</namePart>
      <role>
         <text>programleder</text>
      </role>
   </name>
</mods>
```

Nedenfor følger et utdrag av den tilsvarende metadatabeskrivelsen etter omforming til formatet FastXML. Vi ser at dette er en flat struktur som ikke eksplisitt kobler sammen de elementene som er hierarkisk ordnet i utdraget ovenfor. I eksempelet nedenfor ser vi at navn og rolle ikke er koblet sammen på noen annen måte enn ved at rollen som hører til et

navn kommer rett etter. Det betyr at man ikke uten videre kan søke etter en person som har en bestemt rolle.

```
<document>
   <element name="mods.title">
      <value>Ukens gjest: Næringsminister Finn Kristensen Ekko</value>
   </element>
   <element name="mods.namePart">
      <value>Havgar, Kristin (prod.)</value>
   </element>
   <element name="mods.role">
      <value>produsent</value>
   </element>
   <element name="mods.namePart">
      <value>Solli, Jens</value>
   </element>
   <element name="mods.role">
      <value>programleder</value>
   </element>
</document>
```

Dette er ikke nødvendigvis et problem dersom man ikke er opptatt av å kunne finne disse sammenhengene, men det gjør det vanskeligere å bruke Fast Data Search til å søke strukturbasert. Strukturbasert søk diskuteres i kapittel 3.2.

3.2 Fulltekst kontra strukturbasert søk

INEX (Initiative for the Evaluation of XML Retrieval) [12] er et initiativ der deltakere evaluerer informasjonsgjenfinningsmetoder ved å bruke en testsamling og felles kriterier for resultatmåling. Det har blitt arrangert årlige prosjekter siden 2002 der forskjellige deltakende institusjoner har sammenliknet resultatene sine, og vært med på å bidra til oppbyggingen av en stor testsamling av XML-dokumenter.

Det er to typer spørringer som blir brukt i evalueringen av gjenfinning i samlingen, content-only (CO) og content-and-structure (CAS). CO-spørringer uttrykkes i fritekst og systemet skal finne relevante XML-elementer uavhengig av hvordan dokumentstrukturen er. CAS-spørringer inneholder spesifikke strukturelle begrensninger som sier hvor det skal søkes, for eksempel innenfor en bestemt seksjon i dokumentet.

Under INEX 2004 ble det brukt fire ”tracks” som definerer oppsett av testsamling, spørringer og analyse av resultatene. Disse fire er tilbakemelding på relevans (relevance feedback), naturlig språk, heterogen samling og interaktiv track.

Evaluering av effektiviteten til gjenfinningsmotorene brukt av deltakerne er basert på INEX-testsamlingen og enhetlige vurderingsteknikker. Dette inkluderer recall/precision der man tar hensyn til XML-dokumentenes struktur, og overlapping mellom svar.

For å relatere dette til oppgaven omtalt i kapittel 3.1 kan man si at den bruker CO-spørringer. Fast Data Search er først og fremst beregnet på spørringer der man kun er interessert i innholdet, altså content-only. Noe av hensikten med min oppgave er å finne ut om man kan utnytte strukturen i metadatabeskrivelsene til å søke mer spesifikt etter informasjon. Et eksempel på dette er hvis man ønsker å finne metadata som beskriver noe en bestemt person har hatt en bestemt rolle i. For eksempel vil man finne et lydopptak av et radioprogram som hadde en bestemt person som programleder. Da spesifiserer man det i et brukergrensesnitt og en spørring blir formulert på grunnlag av dette. Spørringen vil på

grunn av strukturen i metadataene kunne kalles en CAS-spørring. Det kan være nødvendig å kombinere data fra forskjellige elementer i metadatabeskrivelsene for å finne svaret på spørringen.

4 Systemspesifikasjon

Systemet som er utviklet er en prototyp på et søkesystem basert på metadata som er samlet inn ved hjelp av OAI-PMH. Med utgangspunkt i tidligere arbeid, som beskrevet i kapittel 3, beskrives valg som er gjort og spesifikasjonene for prototypen i dette kapittelet.

4.1 Vurderinger og valg

Det må gjøres noen valg for de forskjellige delene av systemet. Dette legger grunnlaget for kravene som fastsettes i kapittel 4.2.

4.1.1 Samlinger

Forutsetningene for hvilke samlinger som skal brukes er gitt ved at det skal brukes innhøstede metadata som foreligger i XML-filer med oppføringer på MODS-format. Det er snakk om tre filer som inneholder data fra hver sin samling hentet fra Nasjonalbiblioteket. Prototypen er en del av et system som er en OAI-PMH-basert søketjeneste. Siden metadataene allerede foreligger i innhøstet form og selve konseptet med innhøsting er vist å fungere tidligere, er grensen i denne oppgaven satt ved allerede innhøstede data.

Det kunne vært interessant og testet søketjenesten med et annet og kanskje større utvalg av samlinger. Dette hadde gitt et større datagrunnlag. Kombinert med rikere (mer omfangsrike) metadatabeskrivelser kunne det vært mulig å teste flere problemstillinger knyttet til felles metadataformat. Samtidig er det en hensikt ved å bruke de samme samlingene og de samme metadatabeskrivelsene som tidligere, nemlig at man kan sammenligne resultatene av testingen direkte ved å bruke de samme søketermene.

De tre samlingene som skal benyttes i dette prosjektet inneholder forskjellige typer informasjonsobjekter, noe som er viktig for å kunne se på hvilke problemer som kan dukke opp ved å bruke et felles metadataformat. Samlingene er Galleri NOR, Digitalt radioarkiv (DRA) og Mavis. Samlingene er de samme som ble benyttet av Eskil Solvang i hans prototyp. Nedenfor følger en beskrivelse av disse samlingene hentet fra [1] for å gi en komplett fremstilling.

Galleri NOR

Galleri NOR er Nasjonalbibliotekets fotodatabase og den inneholder mer enn 71000 bilder. Fotodatabasen er tilgjengelig på internett og har vært det siden 1996. Galleri NOR inneholder både metadatabeskrivelser av bildene og selve bildene, men det er bare metadatabeskrivelsene som blir tilgjengelig gjennom søkegrensesnittet i denne oppgaven. De innhøstede metadataene inneholder et utvalg på 1000 metadatabeskrivelser fra Galleri NOR.

Digitalt Radioarkiv (DRA)

Digitalt Radioarkiv (DRA) er et samarbeidsprosjekt mellom Nasjonalbiblioteket og NRK. Formålet er digitalisering og metadatabeskriving av historiske lydbånd og metadatabeskriving av nyere digitale opptak. Digitalt Radioarkiv inneholder metadatabeskrivelser for lydopptakene og lenker til lydfilene. Lydfilene er ikke tilgjengelig gjennom internett på grunn av uavklarte rettighetsspørsmål. Det finnes ca. 24000 timer lydopptak i arkivet. De innhøstede metadataene inneholder er utvalg på 281 metadatabeskrivelser fra Digitalt Radioarkiv.

Mavis

Mavis står for Merged Audio Visual Information System. Det er et integrert system for lagring, administrasjon og gjenfinning av multimedieressurser og tilhørende metadata. Nasjonalbiblioteket bruker Mavis til å administrere metadata for lydmateriale, filmmateriale og pliktavlevert kringkastingsmateriale. De innhøstede metadataene inneholder er utvalg på 122 metadatabeskrivelser fra Mavis, som alle gjelder videofiler.

4.1.2 Metadataomforming og normalisering

Forutsetningene for oppgaven (se kapittel 1.2) sier at innhøstede metadata er lagret i XML-filer med metadatabeskrivelser på MODS-format. Det er én fil for hver av de tre samlingene, Galleri NOR, Digitalt radioarkiv (DRA) og Mavis. Rotelementet i filene er harvest og OAI-PMH-elementer er underelementer til rotelementet, se struktur listet opp nedenfor. OAI-PMH-elementene har underelementer som tilsvarer OAI-verbene (se 2.2.2). Høsterapplikasjonen som er brukt for å høste inn metadataene tidligere har brukt flere av OAI-verbene. Dermed inneholder filene en del informasjon utenom selve metadatabeskrivelsene. Dette er informasjon som en høsterapplikasjon kan ha bruk for, blant annet for å få listet opp metadataformatene som støttes. Denne informasjonen kan tas bort før filene legges inn i databasen, den trengs ikke for å søke i metadataene.

Selve metadatapostene ligger under ListRecords-elementet. Strukturen er slik:

```
<harvest>
   <OAI-PMH>
      <responseDate>(...)</responseDate>
      <request verb="Identify">(...)</request>
      <Identify>
         (informasjon om datalageret)
      </Identify>
   </OAI-PMH>
   <OAI-PMH>
      <responseDate>(...)</responseDate>
      <request verb="ListMetadataFormats">(...)</request>
      <ListMetadataFormats>
         (her listes metadataformatene opp)
      </ListMetadataFormats>
   </OAI-PMH>
   <OAI-PMH>
      <responseDate>(...)</responseDate>
      <request verb="ListSets">(...)</request>
      <error code="noSetHierarchy">The repository does not support sets.</error>
   </OAI-PMH>
   <OAI-PMH>
      <responseDate>(...)</responseDate>
      <request metadataPrefix="mods" verb="ListRecords">(...)</request>
      <ListRecords>
         <record>
            <header>
               <identifier>(...)</identifier>
               <datestamp>(...)</datestamp>
            </header>
            <metadata>
               <mods>
                  (her er de forskjellige mods-elementene)
               </mods>
            </metadata>
         </record>
            .
            .
            .
      </ListRecords>
```

```
   </OAI-PMH>
</harvest>
```

Der det står (…) betyr det at det skal være en verdi. Legg også merke til at de tre prikkene under hverandre nesten nederst betyr at det kan være, og som regel er, flere metadataposter innenfor ListRecords-elementet.

Den unike identifikatoren til posten (under identifier-elementet) er prinsipielt viktig å ha med fordi den gjør det mulig å sjekke for duplikater. Slik duplikatsjekking vil ikke gjøres i denne prototypen, men i et system med store mengder metadatabeskrivelser og gjentatte oppdateringer av metadatabeskrivelser vil det være viktig. Identifikatoren skal vises i søkeresultatene for å gjøre det lett å skille mellom forskjellige poster. Datostempel er informasjon som kan brukes ved oppdateringer av metadataposter. Man kan sjekke om en tidligere høstet post har eldre datostempel enn en post med samme identifikator som finnes i datalageret. Har den det bør den nye posten høstes. Siden oppdateringer ved hjelp av metadatahøsting ikke er aktuelt i denne prototypen vil datostempel ikke bli brukt til noe. Datostempel blir likevel tatt med siden det ikke er noen god grunn til å ikke gjøre det. Dermed blir strukturen i de omformede filene slik:

```
<records>
   <record>
      <header>
         <identifier>(...)</identifier>
         <datestamp>(...)</datestamp>
      </header>
      <metadata>
         <mods>
            (her er de forskjellige mods-elementene)
         </mods>
      </metadata>
   </record>
      .
      .
      .
</records>
```

Legg merke til at rotelementet er records i stedet for ListRecords for å vise at dette ikke er fullstendige OAI-resultater.

I [1] ble datoene normalisert for å vise et eksempel på at det kan være nødvendig å normalisere felter selv om det brukes et felles metadataformat. Datoene har forskjellig format og nøyaktighet alt etter hvordan de opprinnelig er registrert. For å gjøre det lettere å søke med dato som en parameter ble datoene konvertert til et felles format på formen åååammdd, for eksempel 19801025. Datoer vil bli normalisert i denne oppgaven ved å bruke den samme normaliseringsrutinen som Eskil Solvang brukte i [1].

Det vil bli brukt XSLT-stilark til omformingen og normaliseringen, som vil skje i ett trinn ved hjelp av ett stilark. Stilark er en grei måte å gjøre det på, samtidig som det går an å bruke det samme programmet, et som allerede er utviklet.

Dataene legges inn i databasen som XML-filer på MODS-format. Det vil si at ListRecords-delen av filene plukkes ut ved hjelp av stilarket og beholdes uendret med unntak av datokonverteringen.

4.1.3 Lagring og indeksering i databasen

En av forutsetningene i oppgaven er at det skal brukes en XML-database som lokal indeks. Basert på gjennomgangen av egenskapene til eXist i kapittel 2.3 og at databasen er brukt før til prosjekter i IF-gruppen gjør at eXist blir valgt.

eXist var allerede satt opp på en server da denne oppgaven ble påbegynt. Derfor ble det sett på muligheten for å bruke den oppsatte databasen. Det ble også gjort noe testing av funksjoner mot denne databasen. Fordelen med dette er at man kunne ta i bruk en database som allerede var satt opp og klar til bruk. Imidlertid ble det klart at kontrollen over denne databasen var noe begrenset siden den ikke var utelukkende beregnet på denne oppgavens formål. Et annet alternativ var å sette opp en egen database på en egen datamaskin. Fordelen med egen database på en egen datamaskin er at man har full kontroll over maskinen og databasen. Ulempen er at man selv må installere og klargjøre databasen til bruk. Dette var likevel ikke en så stor jobb at det var noe problem, tvert i mot økte det kunnskapen om eXist, noe som var nyttig. Valget falt derfor på å sette opp en egen database på en datamaskin reservert for hovedoppgaven.

Grensesnitt som kan brukes for å aksessere databasen i stand-alone-modus er enten XML-RPC eller et HTTP API, i følge [11]. Imidlertid kan man bruke XML:DB API-et for å jobbe med eXist fra Java-applikasjoner. API-et kan ses på som et lag på et høyere abstraksjonsnivå enn XML-RPC da det kobler seg til en server ved hjelp av XML-RPC. Dette vil være mest aktuelt for lagring og indeksering siden det er her Java skal brukes som programmeringsspråk. Man kan enten bruke XML:DB direkte eller utføre spørringer ved hjelp av XQuery gjennom XML:DB.

Det er også en mulighet å kjøre databasen i en servletkontekst. Fordelen med dette er at det ikke krever noe ekstra oppsett av databasen etter at installasjonsprogrammet er kjørt. En annen fordel er at det følger med et service-wrapper som gjør det mulig å kjøre eXist som en Windows service. Det betyr at den kjører i bakgrunnen uten at man ser programmet når man er logget inn på maskinen. Installasjonen av eXist som service gjøres ved å kjøre en batch-fil som installerer wrapperet som en service.

Den siste muligheten, å kjøre eXist innebygd i en klientapplikasjon, er ikke aktuelt i denne oppgaven. Grunnen til det er at databasen skal aksesseres fra forskjellige typer applikasjoner.

Valget sto mellom å kjøre databasen i stand-alone-modus eller i servletkontekst. Begge deler har blitt testet. Fordi mellomvaren kobler seg til eXist ved hjelp av SOAP (se 4.1.5) var det nødvendig å kjøre databasen i servletkontekst, da stand-alone-modus ikke støtter SOAP.

Indeksering foregår som nevnt (se 2.3.2) automatisk. Både fulltekstindeks og strukturell indeks lages uten at man behøver å gjøre noe. Områdebegrenset indeks kan settes opp for å unngå konverteringsarbeid hver gang for eksempel strenger må konverteres til tall for sammenligning med en annen tallverdi. Imidlertid er ikke den type sammenligninger viktige i denne oppgaven. Derfor settes det ikke opp ekstra indekser utover de eXist styrer automatisk.

4.1.4 Søkegrensesnitt

Søkegrensesnittet skal baseres på søkegrensesnittet som Eskil Solvang utviklet. Dette valget ble tatt fordi det er hensiktsmessig å gjenbruke det som er laget tidligere dersom man kan spare arbeid på det. Det vil gjøre det mulig å konsentrere utviklingsarbeidet på mellomvaren og ikke bruke så mye tid på selve brukergrensesnittet. Dessuten er søkegrensesnittet godt laget og har et enkelt og oversiktlig utseende. Søkegrensesnittet er laget i PHP, HTML og CSS. Disse språkene er kjente og det finnes mye dokumentasjon tilgjengelig på internett dersom man støter på problemer.

For å kunne utføre strukturbaserte søk må det gjøres en utvidelse av søkegrensesnittet. En slik utvidelse må være i stand til å utnytte den hierarkiske strukturen som finnes i metadatabeskrivelsene. Et eksempel er å lage felter i søkegrensesnittet for å angi navn og tilhørende rolle. I tillegg til en utvidelse av søkegrensesnittet må tilsvarende spørringer kunne uttrykkes i mellomvaren. Dette diskuteres i kapittel 4.1.5.

4.1.5 Mellomvare

Mellomvaren skal implementere søking i databasen. Sannsynligvis vil det lønne seg å ta utgangspunkt i mellomvaren Eskil Solvang utviklet. Siden den er utviklet i PHP vil det nok være enklest å bruke det også i denne oppgaven. Dessuten skal søkegrensesnittet og mellomvaren kommunisere, de er ganske tett integrert i søketjenesten til Eskil Solvang. Det gjør at det er enklest å utvikle mellomvaren i PHP.

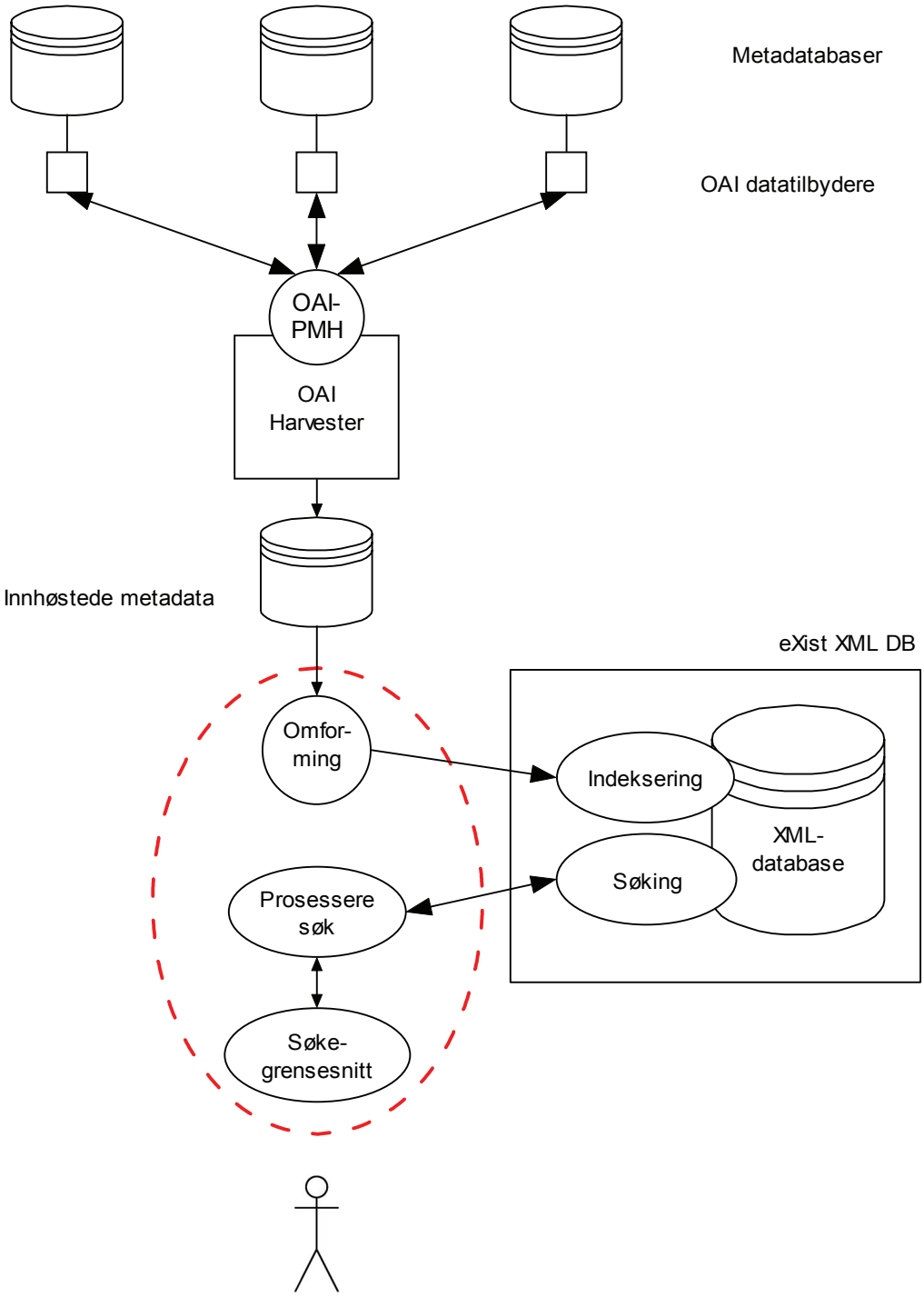
Oppgaven til mellomvaren er å motta spørringer fra søkegrensesnittet og omforme dem slik at de kan sendes til databasen. Mellomvaren får svar fra databasen og omformer dette slik at det kan vises i resultatlisten i søkegrensesnittet. XQuery er språket man bruker for å sende spørringer til eXist. Det er flere måter å sende XQuery-spørringer til eXist fra PHP på.

En mulighet er å bruke XML-RPC ved hjelp av et overliggende bibliotek beregnet for eXist. Et slikt bibliotek kan for eksempel være exist_phpapi [13]. Dette biblioteket implementerer mesteparten av eXist sitt XML-RPC API i PHP. exist_phpapi er foreløpig ikke gitt ut i en endelig versjon og er i følge dokumentasjonen overfladisk testet og inneholder noen kjente bugs. Derfor er det nok lurt å se på andre muligheter først.

Som beskrevet i kapittel 2.3 kan man koble til eXist ved hjelp av et SOAP-grensesnitt. Det er et bra alternativ fordi man sparer en del arbeid ved at man unngår å implementere metodekall slik man må ved direkte bruk av XML-RPC. PHP-klassene [14] utviklet av Òscar Celma er tilgjengelig på Sourceforge. Disse er laget spesielt for eXist og benytter seg av NuSOAP [15], som er et bibliotek for bruk av SOAP i PHP.

Etter denne vurderingen velges det å bruke PHP-klassene utviklet av Òscar Celma for sending av XQuery-spørringer til databasen ved hjelp av SOAP.

Spørringer som muliggjør strukturbasert søk kan formuleres i XQuery. XQuery er laget for å uttrykke spørringer mot XML-data. XML er strukturelt oppbygd og gir mulighet for å uttrykke hierarkiske strukturer. Derfor kan CAS-spørringer (content-and-structure) formuleres i XQuery og sendes til databasen.



Figur 4-1 - Systemets kontekst

4.2 Funksjonelle krav

De forskjellige delene av systemet har oppgaver de skal utføre. Funksjonelle krav beskriver hva delene i systemet skal gjøre. Basert på vurderinger og valg som er gjort beskrives de funksjonelle kravene til systemet.

4.2.1 Metadataomforming og normalisering

Metadatabeskrivelsene (”innhøstede metadata” i Figur 4-1) skal omformes og normaliseres. Kravene til metadataomformingen er:

- Malene for omforming av metadata skal defineres i et XSLT-stilark.
- Det skal gjøres en enkel normalisering av alle innhøstede metadata, slik at forskjellige datotyper konverteres til et felles format.

4.2.2 Indeksering

Det skal ikke settes opp indekser utover de som konfigureres automatisk av eXist.

4.2.3 Prosessering av søk

Prosessering av søk skjer i mellomvaren. Mellomvaren kobler sammen søkegrensesnittet med databasen. Mer detaljert skal mellomvaren:

- Motta søketermer fra søkegrensesnittet
- Lage XQuery-spørringer fra søketermene
- Sende spørringene til eXist ved hjelp av SOAP
- Motta søkeresultater fra eXist
- Omforme søkeresultatene slik at de kan presenteres i søkegrensesnittet

4.2.4 Søkegrensesnitt – grunnleggende krav

Siden søkegrensesnittet skal baseres på søkegrensesnittet utviklet i [1] blir de grunnleggende kravene til søkegrensesnittet i stor grad de samme.

Generelle krav:

- Søkegrensesnittet skal være webbasert
- Søkegrensesnittet skal ha både enkelt og avansert søk

Krav til enkelt søk:

- Skal inneholde ett søkefelt
- Det skal søkes i følgende metadatafelt i metadatabeskrivelsene
    - Tittel
    - Undertittel
    - Navn
    - Utgiver
    - Emneord
    - Abstract
- Det kan søkes med flere søkeord, da skal alle søkeordene finnes i de søkbare feltene
- Det kan søkes med fraser ved å bruke anførselstegn rundt frasen

Krav til avansert søk:

- Det skal kunne søkes med ett eller flere søkeord eller fraser i ett eller flere av søkefeltene:
    - Tittel
    - Navn
    - Utgiver
    - Emneord
    - Abstract
- Ressurstype skal kunne velges (en eller flere):
    - Tekst
    - Lyd
    - Bilde

    - Video
- Det skal kunne velges hvilken datakilde metadatabeskrivelsene opprinnelig kommer fra
- For ressurstype og datakilde skal det bare kunne velges forhåndsdefinerte verdier

Krav til presentasjon av søkeresultater:

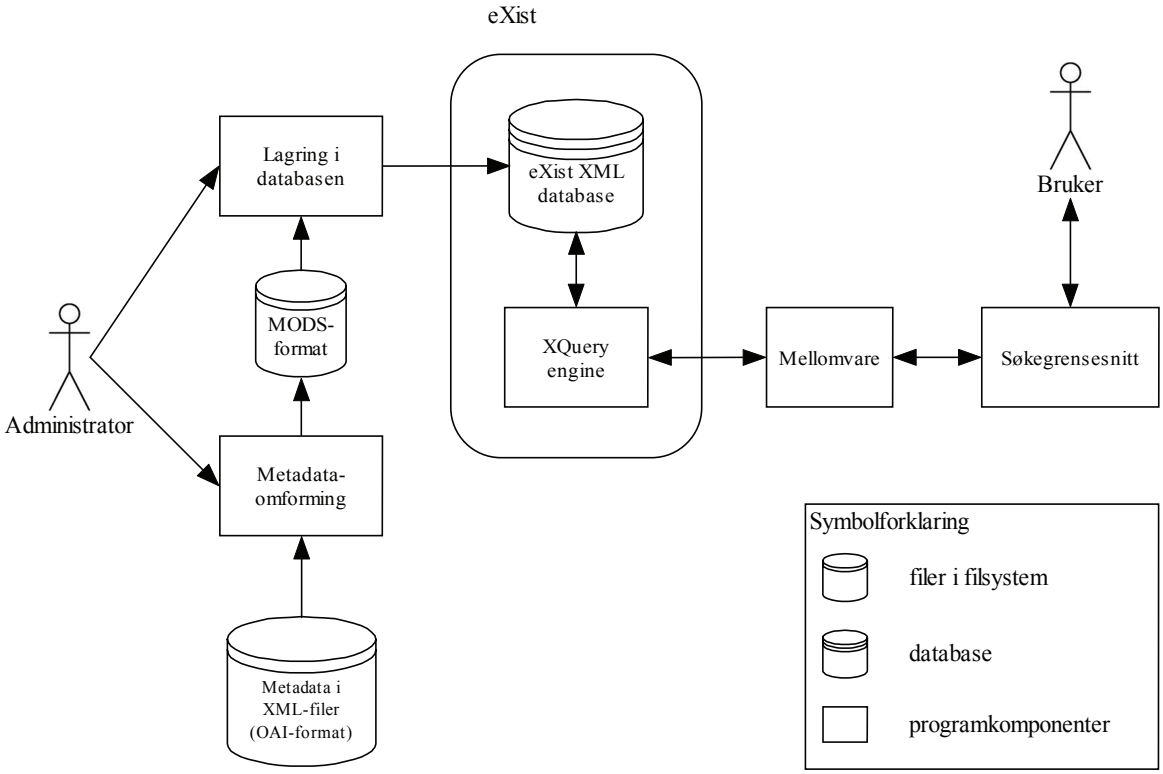
- Antall søketreff skal listes opp
- Søkeresultatene skal inneholde en beskrivelse av hvilken datakilde metadatabeskrivelsen opprinnelig kommer fra
- For enkelt søk skal søkeuttrykket vises i et søkefelt sammen med resultatene

4.2.5 Søkegrensesnitt – krav til utvidelse

Det utvidede søkegrensesnittet skal ha søkefelter for å spesifisere navn med tilhørende rolle. Feltet for rolle skal ikke være begrenset av forhåndsdefinerte verdier, det skal kunne skrives inn rollenavn etter brukerens eget valg.

5 Prototyp

Prototypen har flere deler. Omforming, lagring og indeksering av metadata er oppgaver som foregår uavhengig av brukeren av søkesystemet. Dette er administrative oppgaver som krever en del manuelt arbeid, men til gjengjeld blir de ikke utført så ofte som søking ved hjelp av søkegrensesnittet. Brukeren forholder seg til søkegrensesnittet, som er webbasert for å gjøre tilgjengeligheten lettest mulig.



Figur 5-1 - Protyp

5.1 Metadataomforming

Den utvidede metadatahøsteren som ble utviklet av Eskil Solvang i [1] inneholder funksjonalitet for å omforme XML-filer i henhold til XSLT-stilark. Selve omformingen gjøres av klasser som følger med J2SE i pakken javax.xml.transform. Dette er altså standard funksjonalitet som kun krever litt programmering for å bruke Java sine innebygde klasser.

Klassen TransformXML fra Eskil Solvang sin utvidede OAIHarvester2 [21] skilles ut som et eget program ved å lage en main-metode i klassen slik at den kan kjøres som et selvstendig program. Dermed kan metadataene som foreligger i XML-filer omformes og normaliseres ved å bruke TransformXML og et XSLT-stilark, for til slutt å lagres i nye XML-filer.

Kildekoden til TransformXML finnes som appendiks B-1. XSLT-stilarket finnes som appendiks B-2.

5.1.1 Ekstrahering av metadataposter fra OAI-PMH

Stilark i XSLT bruker en eksisterende XML-struktur og konstruerer en ny struktur basert på maler (templates) og funksjoner. Disse strukturene kalles henholdsvis kildetre (source tree) og resultattre (result tree). Et mønster som matcher elementer i kildetreet fører til at en mal anvendes for å lage en del av resultattreet. Slik kan resultattreet få en helt annen struktur enn kildetreet, eller det kan få den samme strukturen ved å kopiere elementene fra kildetreet til resultattreet. Gjennom denne prosessen kan elementer fra kildetreet filtreres og reorganiseres, og nye strukturer kan legges til.

I ekstraheringen som utføres ved hjelp av stilarket er kildetreet resultatet av metadatahøsting, det vil si XML på OAI-format. Resultattreet består av metadataposter på MODS-formatet med noe OAI-informasjon. En overordnet mal matcher rotelementet i XML-dokumentet. Ut fra dette identifiseres den strukturen som inneholder alle metadatabeskrivelsene i dokumentet. Elementet ListRecords er den øverste noden i denne strukturen. For hver av metadatapostene anvendes det en mal som i sin tur anvender maler for å plukke ut header og metadata i posten. Malen for headere kopierer innholdet i headeren. Malen for metadata fører igjen til at det anvendes maler på selve MODS-elementene. Det er én mal som anvendes på datoelementene, som utfører datokonverteringen. Resten av MODS-elementene kopieres rett over til resultattreet uten å endres.

5.1.2 Normalisering av datoer

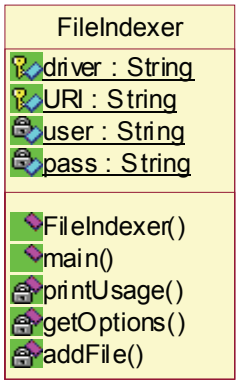
Stilark kan også benyttes til å konvertere dataene i en XML-struktur. Det finnes en rekke funksjoner som kan brukes, først og fremst strengfunksjoner som er en del av XPath. Koden som konverterer datoene til felles datoformat på formen ååååmmdd er hentet fra [1]. Den konverterte datoen lagres i en variabel og det lages et nytt element der verdien av variabelen settes inn. Strukturen i datoelementet er akkurat som før, men verdien er endret.

5.2 Lagring i databasen

Lagring av metadata hentet fra en eller flere datakilder gjøres ved hjelp av Java. Java er et kjent språk som det er forholdsvis enkelt å integrere med eXist. Siden brukeren av søkesystemet ikke er befattet med denne delen av systemet er et enkelt grensesnitt (uten GUI) tilstrekkelig for en prototyp. I en produksjonsversjon av systemet vil et mer avansert lagrings- og indekseringsprogram være å foretrekke. En viktig funksjon i en produksjonsversjon vil være automatisering av oppgavene siden dette er noe som må skje når innhold i metadatakildene endres. For formålet til en prototyp er det derimot viktigst å kunne lagre og indeksere metadata i databasen. Ytelse vil heller ikke være av stor betydning i prototypen.

Etter at metadatene er innhøstet og omformet til riktig format skal de legges inn i databasen. Det er det et Java-program som tar seg av, FileIndexer. Dette er et enkelt kommandolinjebasert program der en katalog og samlingen filene i denne katalogen skal legges inn i oppgis som argumenter. URI til databasen kan også oppgis, dersom den skal peke til noe annet enn en lokal database.

Klassediagrammet i Figur 5-2 viser FileIndexer med egenskaper og metoder.



Figur 5-2 - FileIndexer

5.3 Søketjenesten

Søketjenesten baserer seg på søketjenesten utviklet av Eskil H. Solvang i [1]. Derfor er den grunnleggende oppbygningen den samme. Se figuren i A-1 for UML-modellen av tidligere utviklet søketjeneste. Modellen som viser søketjenesten bruker Web Application Extension (WAE)for UML [16]. Denne utvidelsen av UML kan uttrykke både statiske og dynamiske websider i UML.

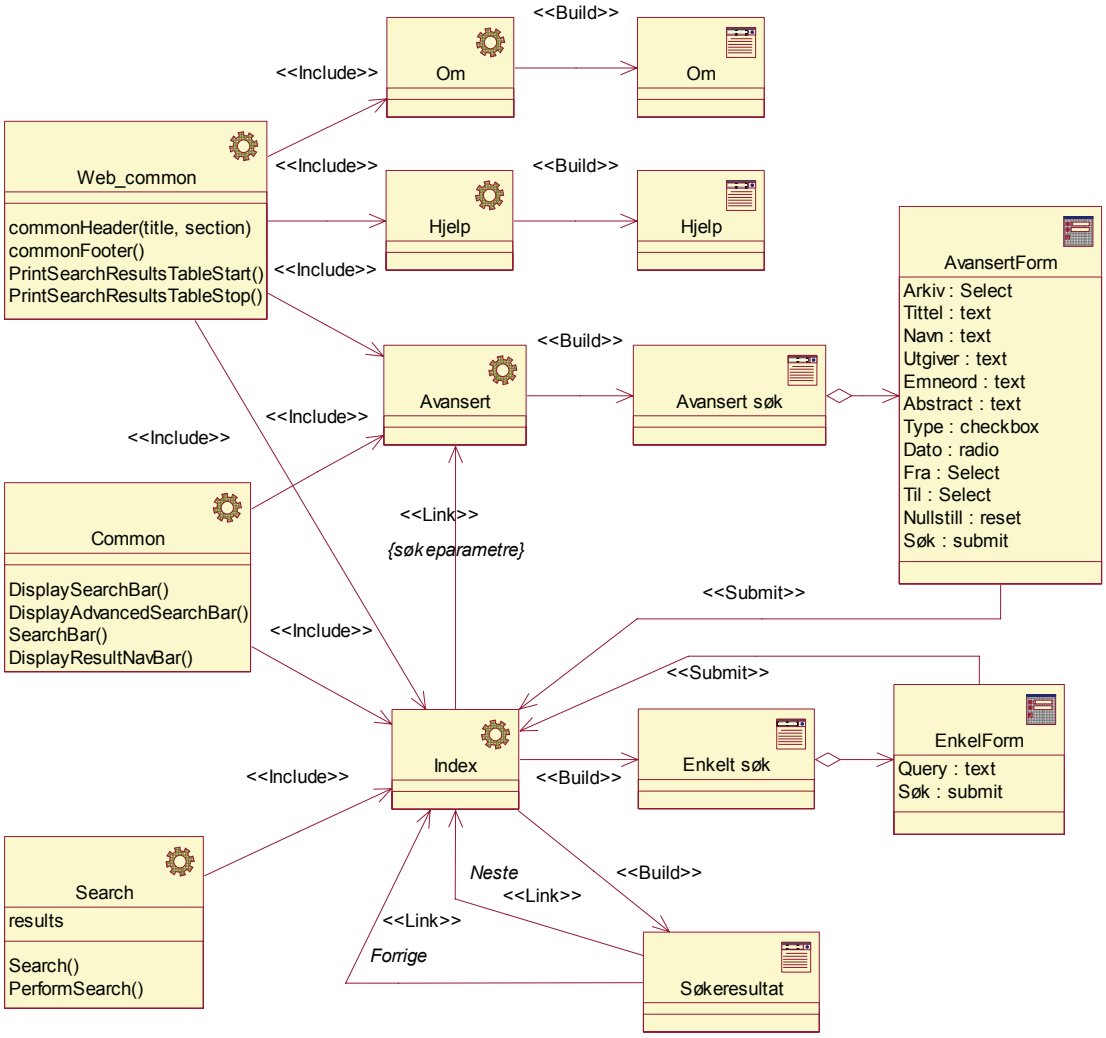
De to hoveddelene i søketjenesten i denne oppgaven er søkegrensesnitt og mellomvare. Mellomvaren tar seg av kommunikasjon mellom søkegrensesnittet og API-et til eXist. Det er denne delen av søketjenesten som skiller seg fra den tidligere versjonen. Notasjonen som brukes i figuren i A-1 skiller seg litt fra notasjonen som brukes i denne oppgaven.

Programmet som brukes i denne oppgaven er Rational Rose, som inneholder noe som kalles Web Modeler. Web Modeler bruker de samme begrepene som WAE og er en modul i Rational Rose som gjør det mulig å modellere og generere kode for webapplikasjoner.

Tabell 5-1 - Modellelementer i Rational Rose Web Modeler

Begrep	Symbol	Forklaring
Serverside		Dynamisk webside med skriptkode
Klientside		HTML-side
HTML-skjema		Skjema i klientside

Modellen i Figur 5-3 viser den samme modellen som figuren i appendiks A-1.



Figur 5-3 - UML-modell av søketjenesten

Søkegrensesnittet består av følgende klientsider:

- Om
- Hjelp
- Avansert søk, inneholder avansert søkeskjema
- Enkelt søk, inneholder enkelt søkeskjema
- Søkeresultat, vises når et søk er utført

Websidene i søkegrensesnittet genereres dynamisk fra de tilhørende serversidene i mellomvarelaget, dette er markert med <<Build>> i modellen.

Serversidene Web_common, Common og Search inneholder metoder som brukes av de serversidene som genererer søkegrensesnittet. Disse metodene, som er skilt ut, kan brukes i flere serversider ved å inkludere serversidene slik modellen viser (<<Include>>).

5.3.1 Mellomvare

Kommunikasjon mellom søkegrensesnittet og databasen er implementert i PHP. Serversiden Search, som inkluderes i serversiden Index, inneholder metoden Search() som

utfører søket. Selve søket i databasen gjøres ved hjelp av XQuery-spørringer. XQuery sendes til databasen ved å bruke noen PHP-klasser som er utviklet for dette formålet [14]. Disse klassene kobler seg til eXist gjennom et SOAP-grensesnitt [17] ved hjelp av en WSDL-definisjon [18]. PHP-klassene benytter seg av NuSOAP [15], som er en samling PHP-klasser som brukes til å lage og ”konsumere” SOAP-baserte webtjenester. Fordelen med NuSOAP er den ikke krever noen utvidelse til PHP for å kjøre. Det holder å ha PHP på webserveren.

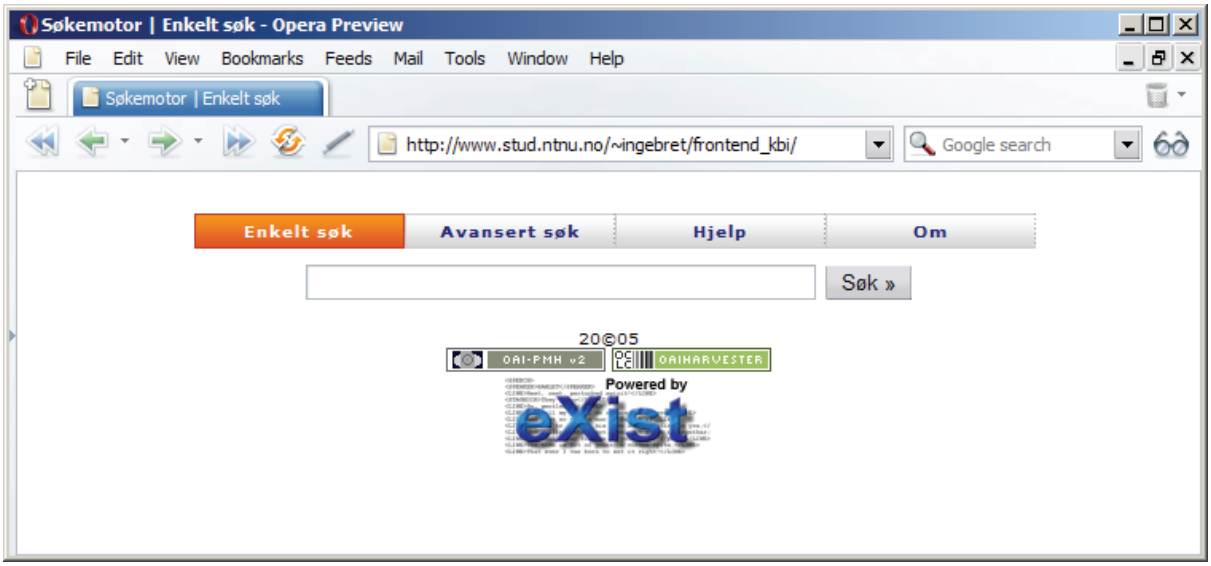
5.3.2 Søkegrensesnitt

Søkegrensesnittet er en tilpasning av tidligere utviklet webbasert søkegrensesnitt [1]. Det er derfor utviklet i PHP, HTML og CSS.

Opprinnelig var det meningen å lage en utvidelse av søkegrensesnittet for å vise hvordan en kan utnytte den hierarkiske strukturen i metadatabeskrivelsene. På grunn av at det tok mer tid enn antatt å ta i bruk det underliggende systemet ble det ikke tid til å lage denne utvidelsen.

Enkelt søk

Enkelt søk gir brukeren mulighet til å skrive inn en eller flere søketermer, eller en frase. Fraser angis ved å sette anførselstegn rundt. Som nevnt tidligere søkes det i feltene tittel, navn, utgiver, emneord og abstract. Spørringen konstrueres i mellomvaren og sendes til eXist.



Figur 5-4 - Enkelt søkegrensesnitt

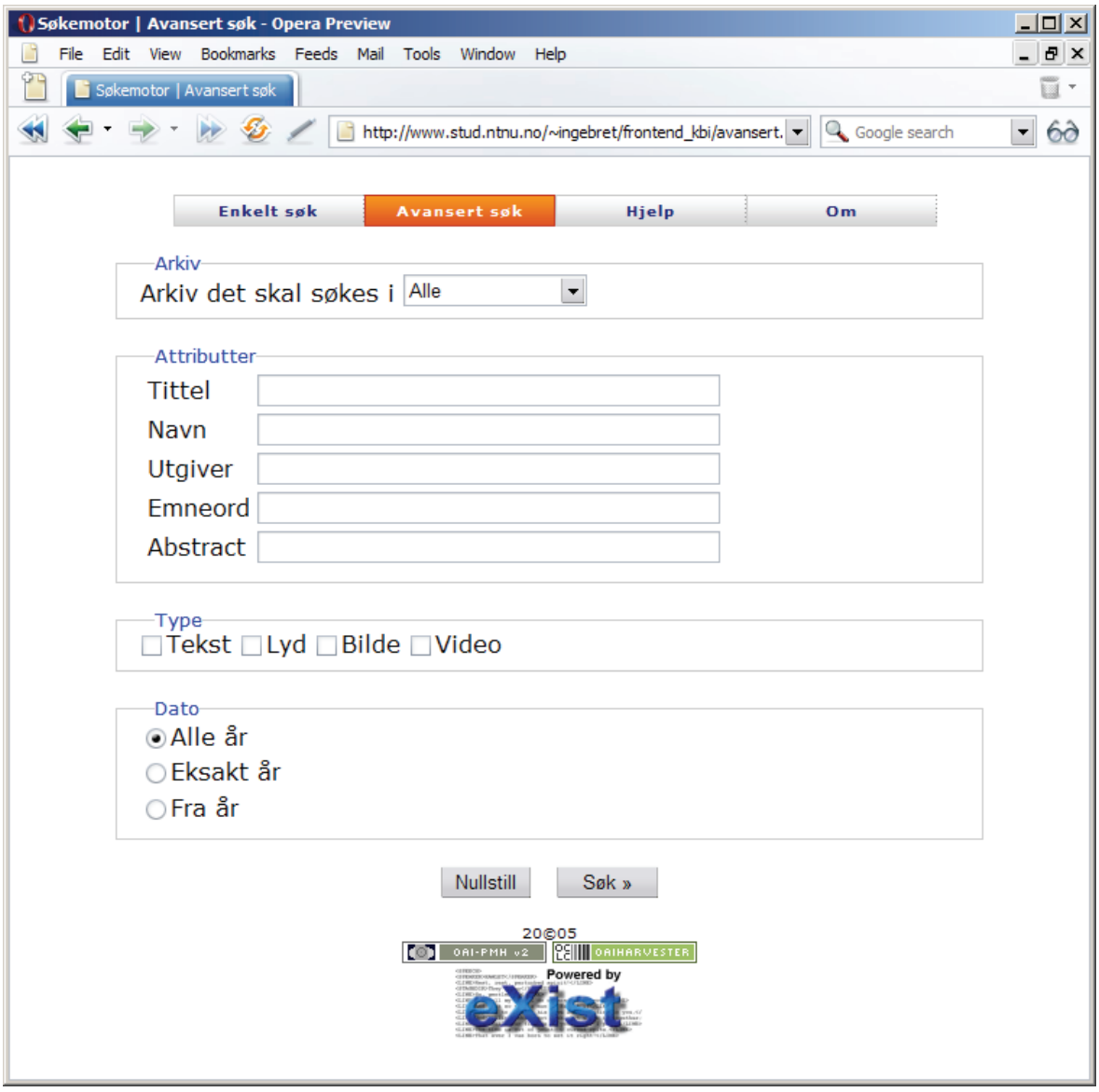
Avansert søk

Avansert søk er en tilpasning av avansert søk fra [1]. I praksis vil det si at et valg i forbindelse med resultatpresentasjon er fjernet, slik at alle søkeresultatene vises på en side. Som nevnt ovenfor ble det ikke utviklet noen utvidelser av søkegrensesnittet.

Avansert søk gir brukeren mulighet til å spesifisere søket mer nøyaktig. Det kan søkes på tittel, navn, utgiver, emneord og abstract, i tillegg til at typen ressurs kan velges.

Ressurstyper er tekst, lyd, bilde og video. Der er også mulig å velge hvilket arkiv det skal søkes i.

Når brukeren trykker på søk sendes søkeparametrene til mellomvaren som formulerer én XQuery-spørring som tar hensyn til alle søkeparametrene. Arkivet velges ved at hvert arkiv er lagret i en egen samling i databasen.



Figur 5-5 - Avansert søkegrensesnitt

6 Testing av søketjenesten

De samme samlingene har blitt brukt i denne oppgaven som i [1]. Som nevnt tidligere (kapittel 4.1.1) var dette for å kunne sammenligne resultatene direkte ved å bruke de samme søketermene. Derfor er eksemplene i dette kapittelet hentet fra eksemplene og sammenlignet med resultatene av eksemplene i [1].

6.1 Enkelt søk

6.1.1 Eksempel 1 (én søketerm)

Søketerm: Stedsnavnet *Eidsvold*.

Dette søket gir 17 treff, som kommer fra disse kildene:

- DRA: 4 treff
- Galleri NOR: 13 treff
- Mavis: 0 treff

Tallene her er de samme som ble funnet i eksempel 1 i [1]. Det er som forventet og viser at søketjenesten gir riktige svar. Søkeresultatene presenteres i en liste som vist i Figur 6-1. Her ser vi også at søketiden var 0,032 sekunder.

6.1.2 Eksempel 2 (én søketerm)

Søketerm: Emneordet *olje*.

Dette søket gir tre treff, som kommer fra disse kildene:

- DRA: 2 treff
- Galleri NOR: 0 treff
- Mavis: 1 treff

Tallene her er de samme som ble funnet i eksempel 2 i [1]. Søkeresultatene presenteres i en liste som vist i Figur 6-2. Søketermen olje ble funnet i tittelfeltet i metadatabeskrivelsen fra Mavis, mens den ble funnet i abstract-feltet i metadatabeskrivelsene fra DRA. Søketiden her var 0,047 sekunder.

6.1.3 Eksempel 3 (sammensatte søketermer)

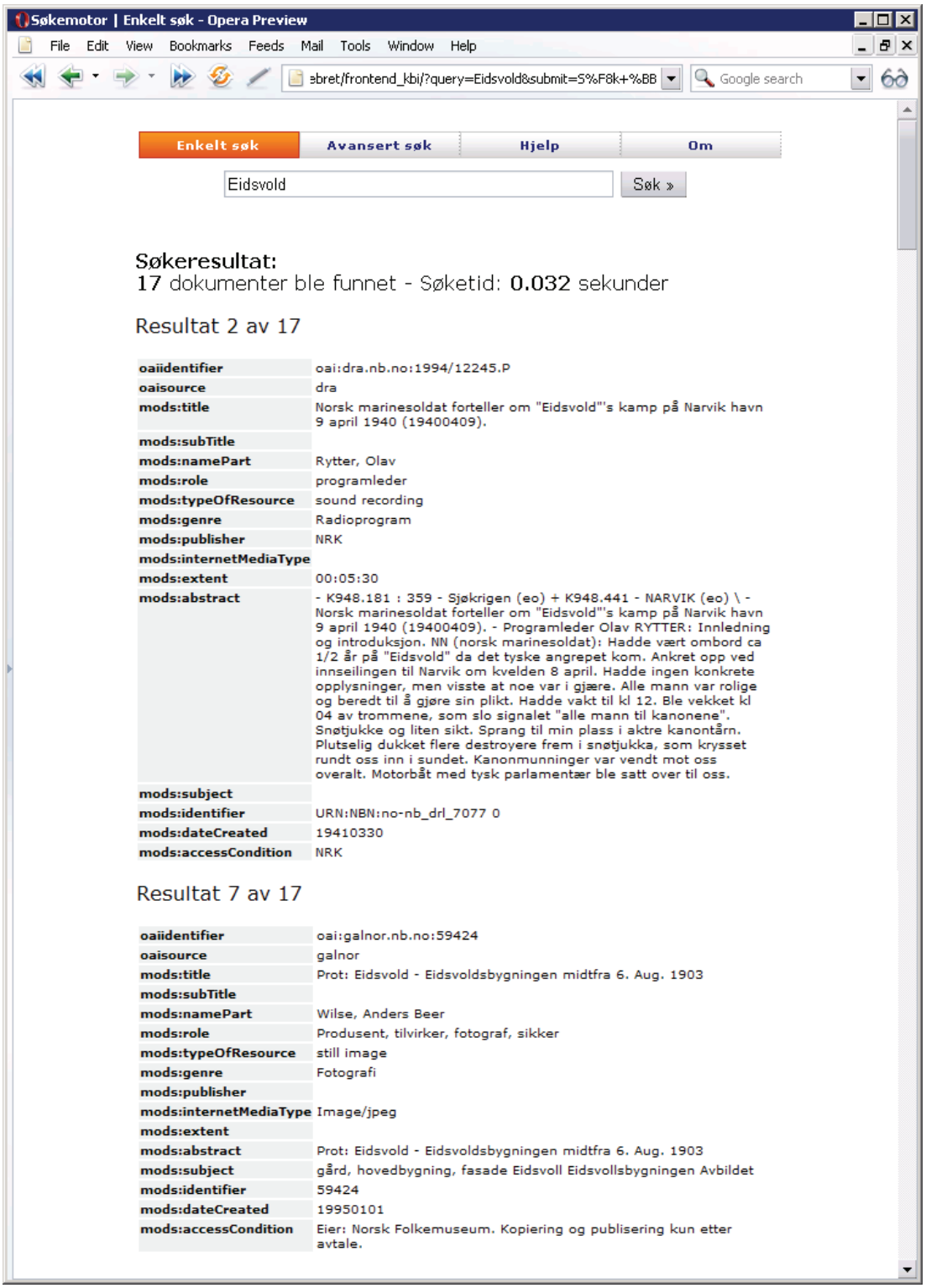
Søkeuttrykk: *Eidsvold jernbane**

Her er trunkeringstegnet * brukt for å finne alle ord som begynner med jernbane. Trunkeringsstøtte får man ved å bruke operatoren &= som er en utvidelse til XQuery spesielt implementert i eXist.

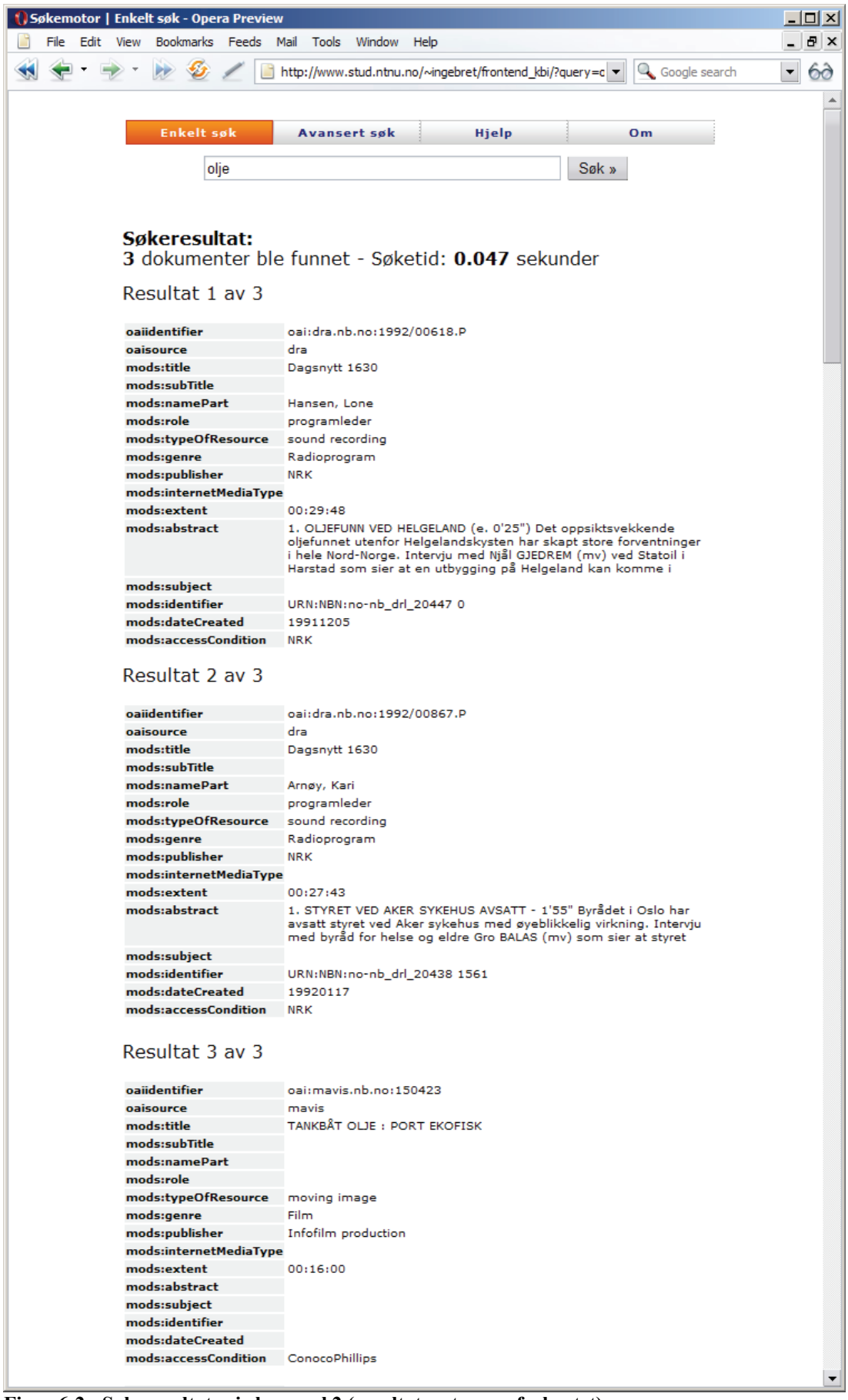
Dette søket gir fire treff, som kommer fra disse kildene:

- DRA: 1 treff
- Galleri NOR: 3 treff
- Mavis: 0 treff

Tallene her er de samme som ble funnet i eksempel 3 i [1]. Resultatene av dette søket er en delmengde av resultatene fra eksempel 1. De inneholder både søketermen Eidsvold og ord som begynner på jernbane.



Figur 6-1 - Utdrag av søkeresultater i eksempel 1 (resultatpost 2 er forkortet)



Figur 6-2 - Søkeresultater i eksempel 2 (resultatpostene er forkortet)

6.1.4 Eksempel 4 (frasesøk)

Frasesøk er mulig ved å skrive anførselstegn rundt frasen. I eksempel 4 på enkelt søk i [1] blir det sagt at det er vanskelig å lage et eksempel på frasesøk som gir søketreff i metadata som stammer fra forskjellige kilder. Grunnen til det er at det er et begrenset datagrunnlag å søke i.

Her er likevel et søkeuttrykk som viser frasesøk: *"Henrik Ibsen*"*

Dette søket gir to treff, som kommer fra disse kildene:

- DRA: 1 treff
- Galleri NOR: 0 treff
- Mavis: 1 treff

Her er trunkeringstegnet * brukt for å finne både Ibsen og Ibsens. Søketreffet fra Digitalt radioarkiv inneholder Ibsens, mens søketreffet fra Mavis inneholder Ibsen. Imidlertid viser det seg at søket gir akkurat de samme treffene ved å utelate anførselstegnene og behandle det som et søkeuttrykk bestående av to søketermer.

6.2 Avansert søk

Avansert søk gir muligheter for å spesifisere søket mer nøyaktig. Brukeren kan velge hvilket arkiv det skal søkes i, hvilke metadatafelt det skal søkes i og typen metadata det skal søkes etter.

På samme måte som enkelt søk er eksemplene på avansert søk hentet fra [1]. Disse eksemplene tar utgangspunkt i søketermen *Eidsvold* for å vise forskjeller og sammenhenger mellom enkelt og avansert søk.

6.2.1 Eksempel 1 (enkeltfelt)

Dette eksempelet tester avansert søk ved å bruke søketermen *Eidsvold* i søkefeltet Tittel.

Dette søket gir 15 treff, som kommer fra disse kildene:

- DRA: 2 treff
- Galleri NOR: 13 treff
- Mavis: 0 treff

Resultatene er de samme som i eksempel 1 på avansert søk i [1]. Med unntak av to færre treff fra DRA er resultatene de samme som ved å søke på *Eidsvold* i enkelt søk. Grunnen til at de to treffene fra DRA ikke kommer med her er at de kun har *Eidsvold* i abstract-feltet og ikke i tittelfeltet.

6.2.2 Eksempel 2 (kombinasjoner av flere søkeparametere)

Søkeresultatene kan avgrenses ved å bruke flere avanserte søkeparametere.

Dersom man søker etter *Eidsvold* i feltet Abstract med DRA som arkiv får man fire treff. Dette er de fire treffene som kom fra DRA ved å søke på *Eidsvold* i enkelt søk.

Det skulle også vært mulig å avgrense søket ved å bruke dato, men dette er ikke implementert i mellomvaren.

6.2.3 Eksempel 3 (avgrensning v.h.a. datakilde og/eller ressurstype)

I det avanserte søkegrensesnittet kan man velge hvilken datakilde metadatabeskrivelsene skal komme fra. Man kan enten velge å søke i alle eller velge en av datakildene Digitalt radioarkiv, Galleri NOR eller Mavis.

Dersom man søker på *Eidsvold* i søkefeltet Abstract og samtidig velger Galleri NOR som datakilde får man 13 søketreff. Hadde man valgt Digitalt radioarkiv som kilde, ville man fått fire søketreff. Til sammen finner man altså alle de 17 treffene fra enkelt søk.

Man kan også avgrense resultatene ved å velge type ressurs. Ved å søke på *Eidsvold* i søkefeltet Abstract og samtidig velge lyd som ressurstype får man fire søketreff. Alle disse er fra Digitalt radioarkiv. Grunnen til det er at Digitalt radioarkiv er den eneste kilden til lydopptak i de samlingene som blir brukt i prototypen.

Disse resultatene er helt i samsvar med resultatene fra [1], noe som viser at avansert søk også fungerer. Unntaket er datoavgrensning som ikke er blitt implementert.

6.3 Oppsummering

Søkegrensesnittet er testet med de samme eksemplene som i [1]. Med unntak av at datoavgrenset søk ikke er ferdig implementert gav søkeeksemplene de samme resultatene. Dette bekrefter at søketjenesten virker og at den gir riktige svar.

For to av eksemplene på enkelt søk er det oppgitt søketider. Disse vises i Figur 6-1 og Figur 6-2, og var henholdsvis 0,032 og 0,047 sekunder. Tilsvarende tider fra eksemplene i [1] var henholdsvis 0,0050 og 0,0030 sekunder. Dette er en forskjell, men i praksis er tidene så lave at det ikke skulle ha noe å si for brukeren av søkesystemet.

En feil som ble oppdaget under testingen fører til at søketermer med æ, ø eller å ikke kan blir funnet i dokumentene, selv om termene finnes. Se kapittel 7.1.2 for en diskusjon av dette.

Det har ikke vært mulig å teste utvidet søk, som tar hensyn til hierarkiske strukturer i metadatabeskrivelsene, fordi det ikke ble implementert.

7 Evaluering

Denne oppgaven har beskrevet en prototyp på et søkesystem basert på interoperabilitet mellom heterogene metadatabaser. Dette kapittelet inneholder en evaluering av prototypen. Evalueringen består av en teknisk del og en bruksevaluering. Den tekniske delen ser på de underliggende funksjonene, det vil si de funksjonene som ikke er synlig for brukeren av søketjenesten. Bruksevalueringen ser på søketjenesten, det vil si funksjonene sett fra brukerens ståsted. Til slutt i kapittelet ses det på mulig videre arbeid.

7.1 Teknisk evaluering

Hovedmålet med oppgaven var å foreslå en alternativ måte å implementere søk på, der metadataenes hierarkiske struktur utnyttes. Det har blitt foreslått en implementasjon som bruker XML-databasen eXist, noe som betyr at metadata kan indekseres på et felles hierarkisk metadataformat. Imidlertid ble det ikke tid til å implementere funksjonaliteten i søketjenesten som kunne utnyttet den hierarkiske strukturen.

Til tross for at det ikke ble tid til å implementere utvidelsen av søketjenesten er det foreslått mulige måter å gjøre dette på. Det er dessuten vist at det er fullt mulig å bruke en XML-database som lokal indeks i et søkesystem for hierarkiske metadata. Løsningen forutsetter ikke bruk av proprietære formater eller løsninger, kun åpent tilgjengelige standarder. Dette er en stor fordel siden man ikke er bundet til begrensninger som kan ligge i bruken av proprietære løsninger. Med dette menes også kommersielle produkter man må betale for å bruke.

7.1.1 XML-database sammenlignet med Fast Data Search

Det å bruke eXist i stedet for Fast Data Search som lokal indeks fungerer bra. De testsamlingene som er brukt inneholder en begrenset mengde metadata, derfor er det vanskelig å si noe om hvordan systemet ville taklet store mengder metadata. Det har ikke blitt testet hvordan eXist skalerer til store mengder data. Dersom datamengdene blir store kan det være at ytelsen går ned. Styrken til Fast Data Search er nettopp muligheten for å takle store datamengder. Størrelsen på metadatabeskrivelser kan variere, men normalt er de ikke veldig store. Det skal derfor mye til at datamengdene som indekseres i databasen blir virkelig store.

Dersom man skal bruke eXist som lokal indeks i et søkesystem der mange og store samlinger skal inngå bør skalerbarheten undersøkes nærmere.

Tidsforbruket til spørringene var høyere ved bruk av eXist enn ved bruk av Fast Data Search. Dette kan indikere at søking i eXist ikke er så raskt som søking med Fast Data Search. Imidlertid er det vanskelig å sammenligne disse tallene fordi de kommer fra to forskjellige systemer og man vet ikke hvordan de er beregnet. Det er også vanskelig å si hvor nøyaktig tallene er siden de varierte en del ved gjentatte søk med identiske søketermer. Dersom man gjentar søket fortløpende flere ganger ved å trykke oppdater i nettleseren ser man at søketiden går opp og ned. Dette indikerer at tallet ikke er så nøyaktig, og ikke kan brukes til å sammenligne ytelsen. Det er likevel interessant å se at tallene konsekvent er høyere for eXist enn for Fast Data Search. Likevel må det påpekes at ytelsen ikke var et viktig kriterium i denne prototypen, derfor legges det ikke så stor vekt på dette. Dessuten er ikke søketiden så lang at det faktisk er et problem, brukeren må ikke vente på resultatet.

En viktig fordel man får ved bruk av Fast Data Search er en ferdig utviklet indekserings- og søkefunksjonalitet. Bruker man en XML-database til å indeksere og søke i data, må man utvikle mye mer av søkefunksjonaliteten selv. Riktignok gir eXist svar på spørringer ved å bruke XQuery, og har utvidelser til XQuery som kan gi kraftige spørringer, men funksjoner slik som relevanssortering, avgrensing av antall treff, og optimalisert søkehastighet får man "gratis" ved å bruke Fast Data Search. Relevanssortering kunne vært implementert ved bruk av eXist, ved å legge sortering av resultatene inn i XQuery-spørringene.

7.1.2 Søketjenesten

Søketjenesten har blitt implementert med tilnærmet samme funksjonalitet som i [1]. Forskjellen er at datoavgrenset søk ikke er ferdig. Det har blitt oppdaget at mye av arbeidet med å utvikle mellomvaren i søketjenesten ligger i å formulere XQuery-spørringer. Dette er et stort område for videre arbeid, og det finnes mange muligheter for å lage avanserte spørringer. En annen utfordring vil ligge i å lage spørringene så effektive som mulig, slik at avanserte spørringer ikke bruker uakseptabelt lang tid.

Et problem som ble oppdaget under testingen av søketjenesten er at søketermer som inneholder æ, ø eller å ikke returnerer noe svar, selv om søketermene finnes i samlingene. Dette kan skyldes "PHP & Perl classes to query eXist XML:DB" eller NuSOAP som denne bruker. Det ble funnet en melding[1] på en nyhetsgruppe som kan tyde på at det er problemer med tegnkodingen i disse klassene. På grunn av at dette ble oppdaget forholdsvis sent i utviklingsprosessen ble det ikke tid til å se nærmere på det. Imidlertid utgjorde det ikke noe problem i forhold til testingen av prototypen.

Under den testingen som foregikk parallelt med utviklingen oppsto det problemer med å få XQuery til å sammenligne flere noder på en gang. Dette er et problem som har ført til at ikke alle søkekombinasjoner fungerer helt som de skal.

7.2 Bruksevaluering

Det ble utført testing av søketjenesten (kapittel 6). Testene viste at søketjenesten ga de samme resultatene som søketjenesten utviklet av Eskil Solvang [1]. Det viser at søketjenesten fungerer og gir riktige svar på spørringer.

Fra et brukerperspektiv skiller ikke søketjenesten utviklet i denne oppgaven seg mye fra søketjenesten til Eskil Solvang. Den gir de samme svarene. Den eneste synlige forskjellen er tidsforbruket til spørringene, som var noe høyere for søketjenesten i denne oppgaven. Fordi søketidene uansett er såpass lave har det ikke praktisk betydning i de eksemplene som er sammenlignet. For større datamengder kunne det gitt utslag slik at brukeren kunne ha merket forskjellen i tidsforbruk.

Det skulle utvikles en utvidelse til søkegrensesnittet, men på grunn av for stort tidsforbruk på andre deler av oppgaven ble det ikke tid til det.

7.3 Videre arbeid

Prototypen fungerer for enkelt og avansert søk, likevel er det flere muligheter for å utvikle den videre. Det kan være interessant å arbeide videre med:

[1] http://article.gmane.org/gmane.text.xml.exist/5282

- Utvikle funksjonaliteten for utvidet søk i avansert søkegrensesnitt.
- Det bør vurderes å inkludere større og flere samlinger i systemet for å få et større datagrunnlag å teste mot.
- Det må undersøkes om eXist kan brukes til store mengder metadatabeskrivelser.
- For å gjøre det lettere for brukeren å finne de mest relevante søkeresultatene kan man legge til relevanssortering av søkeresultatene. Dette forutsetter avanserte XQuery-spørringer.

8 Konklusjon

I denne oppgaven er det vist at en XML-database, nærmere bestemt eXist, kan brukes til lokal indeksering av metadata på et hierarkisk format. Sammenlignet med bruk av Fast Data Search ser det ut til at eXist egner seg godt for dette formålet. Det er imidlertid funnet resultater som kan indikere at eXist kan være noe dårligere på ytelse enn Fast Data Search, men det er ikke gjort nok undersøkelser til å kunne fastslå dette.

Videre er det brukt et tidligere utviklet søkegrensesnitt i kombinasjon med ny mellomvare for å kunne søke i metadatabeskrivelsene i den lokale indeksen. Tilpasningen av søkegrensesnittet til mellomvaren gikk greit, mens mellomvaren krevde en del arbeid for å få til koblingen mellom søkegrensesnittet og databasen. Det har også vist seg at mye av arbeidet med å utvikle mellomvaren til søketjenesten er å lage XQuery-spørringer.

Det skulle også utvikles en utvidelse til søkegrensesnittet. Dette ble ikke gjort, men det er grunn til å tro at en slik utvidelse hadde vært mulig å utvikle.

Noe av funksjonaliteten i søketjenesten er redusert i forhold til søketjenesten som brukte Fast Data Search. Det er hovedsakelig på grunn av at flere søkefunksjoner må utvikles på egen hånd når man bruker en XML-database som eXist. Fast Data Search har en del innebygde funksjoner som gir denne funksjonaliteten.

9 Referanser

[1] Eskil Høyen Solvang. Enhetlig tilgang til heterogene metadatabaser - Interoperabilitet v.h.a. OAI-PMH, Hovedfagsoppgave i informasjonsforvaltning. Institutt for datateknikk og informasjonsvitenskap, NTNU, desember 2004.

[2] Open Archives Initiative - Protocol for Metadata Harvesting - v.2.0, Open Archives Initiative, 2002.
http://www.openarchives.org/OAI/openarchivesprotocol.html
Sist sjekket: 10.6.2005.

[3] Fast Data Search.
http://www.fastsearch.com/no/products/fast_data_search
Sist sjekket: 13.6.2005.

[4] Fast Search & Transfer.
http://www.fastsearch.com
Sist sjekket: 13.6.2005.

[5] XQuery 1.0: An XML Query Language.
http://www.w3.org/TR/xquery
Sist sjekket: 13.6.2005.

[6] XML Path Language (XPath).
http://www.w3.org/TR/xpath
Sist sjekket: 13.6.2005.

[7] eXist. Open Source Native XML Database.
http://www.exist-db.org
Sist sjekket: 13.6.2005.

[8] Guenther, Rebecca S. MODS: The Metadata Object Description Schema.
portal: Libraries and the Academy, Vol. 3, No. 1 (2003), s. 137-150.

[9] Metadata Object Description Schema. Outline of Elements and Attributes in MODS Version 3.0
http://www.loc.gov/standards/mods/v3/mods-3-0-outline.html
Sist sjekket: 11.6.2005.

[10] Shalaku Natu og John Mendonca. Digital Asset Management – Using A Native XML Database Implementation. Proceeding of the 4th conference on Information technology curriculum, ACM Press, 2003, s. 237-241.

[11] Dokumentasjon til eXist.
http://www.exist-db.org/documentation.html
Sist sjekket: 13.6.2005.

[12] Initiative for the Evaluation of XML Retrieval.
http://inex.is.informatik.uni-duisburg.de
Sist sjekket: 13.6.2005.

[13] exist_phpapi
http://cvs.sourceforge.net/viewcvs.py/exist/exist_phpapi
Sist sjekket: 10.5.2005.

[14] PHP & Perl classes to query eXist XML:DB.
http://query-exist.sourceforge.net
Sist sjekket: 13.6.2005.

[15] NuSOAP.
http://dietrich.ganx4.com/nusoap
Sist sjekket: 13.6.2005.

[16] Conallen, Jim. Building Web Applications With UML 2nd Ed. Addison-Wesley, Boston, Massachusetts, 2002.

[17] SOAP Version 1.2 Part 1: Messaging Framework.
http://www.w3.org/TR/soap12-part1
Sist sjekket: 15.5.2005.

[18] Web Services Description Language (WSDL) 1.1.
http://www.w3.org/TR/wsdl
Sist sjekket: 10.5.2005.

[19] Metadata Object Description Schema.
http://www.loc.gov/standards/mods
Sist sjekket: 11.6.2005.

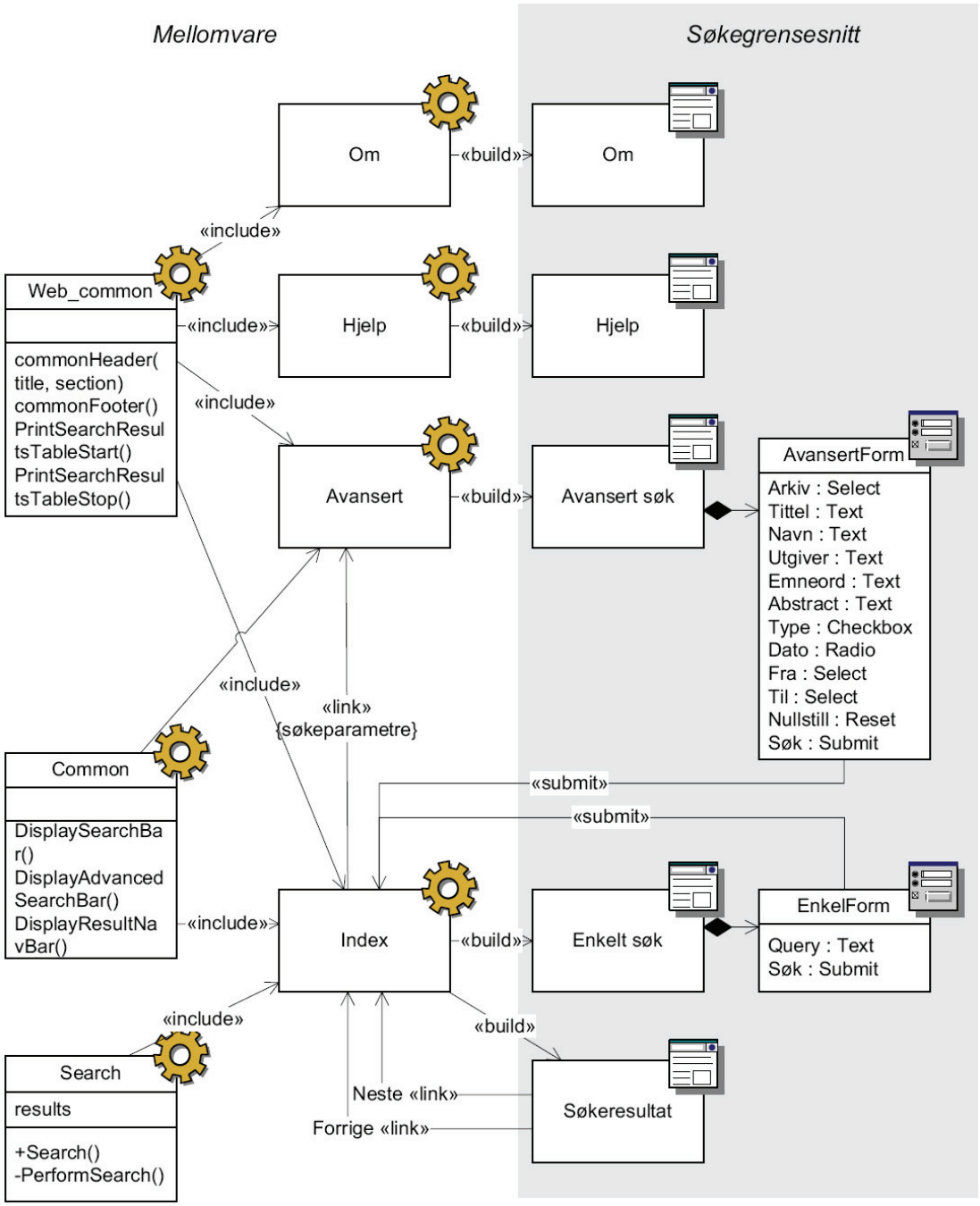
[20] Guenther, Rebecca S. og McCallum, Sally. New Metadata Standards for Digital Resources: MODS and METS. Bulletin of the American Society for Information Science and Technology, 2003, 29:2, s 12-15.

[21] OAIHarvester2. OCLC Research, 2003.
http://www.oclc.org/research/software/oai/harvester2.htm
Sist sjekket: 7.6.2005.

Appendiks A - Søketjeneste

A-1 Grunnlaget for søketjenesten

Søketjenesten bygger på søketjenesten som ble utviklet av Eskil H. Solvang. Figuren nedenfor viser UML-modellen av søkegrensesnittet og mellomvarelaget til søketjenesten.



Figur A-1 - Grunnlaget for søketjenesten

Appendiks B - Metadataomforming

TransformXML er en Java-klasse som omformer XML-filer med metadataposter ved hjelp av XSLT-stilark. Denne klassen er hentet fra prosjektet til Eskil Solvang. Det er lagt til en main-metode slik at den kan startes som et eget program. Programmet tar to argumenter. Det er navnet på XML-filen som skal omformes og xsl-filen som inneholder stilarket som beskriver omformingen.

For å omforme de tre XML-filene med metadata kan følgende kommandoer brukes:

```
java harvester2.app.TransformXML dra.xml TransformXML.xsl

java harvester2.app.TransformXML galnor.xml TransformXML.xsl

java harvester2.app.TransformXML mavis.xml TransformXML.xsl
```

Dette produserer tre filer som output, der filene navngis ved å kombinere navnet på XML input-filene og xsl-filene. For eksempel blir navnet på den første filen dra_TransformXML.xml.

B-1 Java-kode

```
/********************
 * TransformXML.java *
 ********************/
package harvester2.app;

import java.io.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
/**
 * This class transforms xml input to some other format in accordance with an
 * xslt stylesheet.
 *
 * @author Eskil H. Solvang, NTNU-IDI.
 * @author Knut B. Ingebretsen
 *
 * Knut: Lagt til main-metode slik at transformeringen
 * kan kjøres som et eget program.
 */
public class TransformXML
{
    /**
     * Method for transforming xml harvest file
     *
     * @param xmlSource the xml file containing input data.
     * @param xsltSource the xslt stylesheet file.
     * @exception IOException an I/O error occurred.
     * @exception TransformerException a transformation error occured.
     */
    public static void transformHarvest(String xmlFileName, String xsltFileName)
    throws IOException, TransformerException
    {
        File xmlFile = new File(xmlFileName);
        File xsltFile = new File(xsltFileName);
        //transformation sources
        Source xmlSource = new StreamSource(xmlFile);
        Source xsltSource = new StreamSource(xsltFile);
        //strip file endings from input file names, generate output file name
        String newName = "default";
        if ((xmlFileName.endsWith(".xml")) && (xsltFileName.endsWith(".xsl")))
        {
            String xmlStrip =
```

```
                xmlFileName.substring(0,xmlFileName.indexOf(".xml"));
            String xsltStrip =
                xsltFileName.substring(0,xsltFileName.indexOf(".xsl"));
            newName = xmlStrip + "_" + xsltStrip;
        }
        //send the result to a file (and display it on screen)
        File resultFile = new File(newName + ".xml");
        Result result_file = new StreamResult(resultFile);
        //get a transformer factory
        TransformerFactory transFact = TransformerFactory.newInstance();
        //get a transformer for this particular stylesheet
        Transformer trans = transFact.newTransformer(xsltSource);
        //do the transformation
        trans.transform(xmlSource, result_file);
    }

    public static void main(final String args[])
    {
        try
        {
            transformHarvest(args[0], args [1]);
        }
        catch (Exception e)
        {
            e.printStackTrace();
            System.exit(-1);
        }
    }
}
```

B-2 Stilark

```
<?xml version="1.0" encoding="UTF-8"?>
<!--==============================
Document: TransformXML.xsl
Created on: 24. april 2003
Last edited:
Author: Knut Bjørke Ingebretsen
Description:
Konverterer datoer fra diverse formater til et felles format ååååmmdd
Datokonverteringen er laget av Eskil Høyen Solvang
==============================-->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:oai="http://www.openarchives.org/OAI/2.0/"
  xmlns:mods="http://www.loc.gov/mods/">
  <!-- Angir outputtype, samt om denne skal formatteres med innrykk.-->
  <xsl:output method="xml" version="1.0" encoding="iso-8859-1" indent="yes"/>
  <!--========= Template:main ======== -->
  <xsl:template match="/">
    <records>
      <!-- Elementet ListRecords fører til anvendelse av en template -->
      <xsl:apply-templates select="harvest/oai:OAI-PMH/oai:ListRecords"/>
    </records>
  </xsl:template>
  <!--========= Template:record =========-->
  <xsl:template match="oai:record">
    <record>
      <xsl:apply-templates select="oai:header"/>
      <metadata>
        <xsl:apply-templates select="oai:metadata"/>
      </metadata>
    </record>
  </xsl:template>
  <!--========= Template:header =========-->
  <xsl:template match="oai:header">
    <!-- Kopierer oai-headeren -->
    <xsl:copy-of select="."/>
```

```
</xsl:template>
<!--======== Template:mods ========-->
<xsl:template match="mods | mods:mods">
  <!-- Kopierer mods-elementet -->
  <xsl:copy>
    <!-- Går gjennom alle mods-elementene -->
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
<!--======== Template:originInfo ========-->
<xsl:template match="mods:mods/mods:originInfo | mods/originInfo">
  <!-- Kopierer originInfo-elementet -->
  <xsl:copy>
    <!-- Går gjennom alle elementene under originInfo -->
    <xsl:for-each select="child::node()">
      <!-- Finner ut om elementet er av en type som inneholder en dato -->
      <xsl:choose>
        <!-- dato-element, tester for 'ate' fordi elementene kan
        begynne på 'date' eller slutte på 'Date' -->
        <xsl:when test="contains(local-name(), 'ate')">
          <!--====== Datokonvertering/-normalisering =======-->
          <!-- Kontekstnode . = date* | *Date -->
          <!-- Variabel som inneholder antall tegn i date* | *Date -->
          <xsl:variable name="v_datelength" select="string-length(.)"/>
          <!-- Variabel som skal inneholde en 8-tegns heltallsdato -->
          <xsl:variable name="v_dateint">
            <xsl:choose>
              <!-- 4 tegn -->
              <xsl:when test="$v_datelength = 4">
                <xsl:choose>
                  <!-- 1980 -->
                  <xsl:when test="substring(.,1,1) &gt;=0">
                    <xsl:value-of select="concat(.,'0000')"/>
                  </xsl:when>
                  <!-- Ca. -->
                  <xsl:when test="contains(.,'Ca.')">00000000</xsl:when>
                  <!-- Ukjent datotype -->
                  <xsl:otherwise>00000000</xsl:otherwise>
                </xsl:choose>
              </xsl:when>
              <!-- 6 tegn -->
              <xsl:when test="$v_datelength = 6">
                <xsl:choose>
                  <!-- 1980-? -->
                  <xsl:when test="contains(substring(.,6,1),'?')">
                    <xsl:value-of select="concat(substring(.,1,4),'0000')"/>
                  </xsl:when>
                  <!-- 194006, usikker på om dette er juni 1940-->
                  <xsl:when test="substring(.,6,1) &gt;= 0">
                    <!-- p.g.a. usikkerhet fjernes mer nøyaktig info enn år -->
                    <xsl:value-of select="concat(substring(.,1,4),'0000')"/>
                  </xsl:when>
                  <!-- Ukjent datotype -->
                  <xsl:otherwise>00000000</xsl:otherwise>
                </xsl:choose>
              </xsl:when>
              <!-- 7 tegn -->
              <xsl:when test="$v_datelength = 7">
                <xsl:choose>
                  <!-- 1098911 -->
                  <xsl:when test="substring(.,1,1) &gt;= 0">
                    <!-- p.g.a. usikkerhet fjernes mer nøyaktig info enn år -->
                    <xsl:value-of select="concat(substring(.,1,4),'0000')"/>
                  </xsl:when>
                  <!-- Ukjent datotype -->
                  <xsl:otherwise>00000000</xsl:otherwise>
                </xsl:choose>
              </xsl:when>
```

```
                <!-- 8 tegn -->
                <xsl:when test="$v_datelength = 8">
                  <xsl:choose>
                    <!-- Ca. 1990 -->
                    <xsl:when test="contains(.,'Ca.')">
                      <xsl:value-of select="concat(substring-after(.,'Ca.
                        '),'0000')"/>
                    </xsl:when>
                    <!-- Ukjent datotype -->
                    <xsl:otherwise>00000000</xsl:otherwise>
                  </xsl:choose>
                </xsl:when>
                <!-- 10 tegn -->
                <xsl:when test="$v_datelength = 10">
                  <xsl:choose>
                    <!-- 1980-01-01 -->
                    <xsl:when test="contains(substring(.,5,1),'-')">
                      <xsl:value-of select="concat(substring(.,1,4),substring(.,
                        6,2),substring(.,9,2))"/>
                    </xsl:when>
                    <!-- Ca. 1980-? -->
                    <xsl:when test="contains(substring(.,10,1),'?')">
                      <xsl:value-of select="concat(substring(.,5,4),'0000')"/>
                    </xsl:when>
                    <!-- Ukjent datotype -->
                    <xsl:otherwise>00000000</xsl:otherwise>
                  </xsl:choose>
                </xsl:when>
                <!-- Andre datolengder settes til 00000000 -->
                <xsl:otherwise>00000000</xsl:otherwise>
              </xsl:choose>
            </xsl:variable>
            <!-- Nytt element som inneholder en enhetlig dato -->
            <xsl:copy>
              <xsl:copy-of select="@*"/>
              <xsl:value-of select="$v_dateint"/>
            </xsl:copy>
            <!--====== SLUTT: Datokonvertering ======-->
          </xsl:when>
          <!-- ikke dato-element -->
          <xsl:otherwise>
            <xsl:copy-of select="."/>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:for-each>
    </xsl:copy>
  </xsl:template>
  <!--========= Template:not(originInfo) =========-->
  <xsl:template match="//mods:mods/node()[not(self::mods:originInfo)] |
    //mods/node()[not(self::originInfo)]">
    <xsl:copy-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```

Appendiks C - FileIndexer

FileIndexer er en Java-klasse som har som oppgave å legge XML-filer inn i databasen. Følgende er et eksempel på bruk av FileIndexer:

```
java -classpath "exist.jar;exist-optional.jar;lib\core\antlr.jar;lib\core\commons-
pool-1.1.jar;lib\core\log4j-1.2.8.jar;lib\core\xmldb.jar;lib\core\xmlrpc-1.2-
patched.jar;FileIndexer.jar"
FileIndexer -r . -c /db/nb -uri xmldb:exist://localhost:8080/exist/xmlrpc
```

Jar-filene som er listet opp i argumentet –classpath, utenom FileIndexer.jar, følger med eXist. Disse inneholder klasser som FileIndexer bruker for å legge inn filer i eXist. Det er ikke alle jar-filene som brukes av FileIndexer, men de er tatt med siden de er listet opp som et minimum av filer man bør ha i classpath ifølge dokumentasjonen til eXist.

På den siste linjen i eksempelet ovenfor står programmet og parametrene det tar. –r spesifiserer katalogen der filene som skal lagres i databasen ligger, alle filer med endelsen xml leses. Punktum betyr gjeldende katalog. –c spesifiserer samlingen (collection) i databasen der filene skal lagres. Alle samlinger blir lagret under /db. Dersom den spesifiserte samlingen ikke eksisterer blir den opprettet. –uri er en valgfri parameter som spesifiserer URI-en til databasen. I eksempelet kjører eXist-databasen på samme maskin som FileIndexer kjøres på (localhost).

C-1 Koden til FileIndexer

```
/*
 * FileIndexer.java
 *
 * Created on 20. april 2005, 15:38
 * Updated on 7. juni 2005
 * Hver fil som legges inn havner i en egen samling
 */

import java.io.File;
import java.util.HashMap;
import java.util.ArrayList;

import org.xmldb.api.*;
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;

/**
 * This class adds all the xml-files i a given directory to av given
 * collection in the eXist database.
 *
 * @author Knut Bjørke Ingebretsen
 */
public class FileIndexer
{
    protected static String driver = "org.exist.xmldb.DatabaseImpl";
    protected static String URI = "xmldb:exist://localhost:8080/exist/xmlrpc";
    private static String user ="admin";
    private static String pass ="";

    /** Creates a new instance of FileIndexer and
     * configures the Database Manager
     */
    public FileIndexer()
    {
        try
```

```
        {
            // initialize database drivers
            Class cl = Class.forName(driver);
            Database database = (Database) cl.newInstance();
            DatabaseManager.registerDatabase(database);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        // TODO code application logic here
        FileIndexer fileIndexer = new FileIndexer();
        try
        {
            HashMap options = getOptions(args);

            String inDirectory = (String)options.get("-r");
            String collectionName = (String)options.get("-c");
            String databaseURI = (String)options.get("-uri");

            // if directory is not given, use current directory
            if (inDirectory == null)
            {
                inDirectory = ".";
            }
            // if collection is not given, use root collection
            if (collectionName == null)
            {
                collectionName = "/db";
            }
            // is uri given?
            if (databaseURI != null)
            {
                URI = databaseURI;
            }

            // get a list of the filenames in the given directory
            File directory = new File(inDirectory);

            File[] fileList = directory.listFiles();
            if (fileList == null)
            {
                throw new IllegalArgumentException();
            }
            else
            {
                // add all the xml-files in the given directory to database
                for (int i = 0; i < fileList.length; i++)
                {
                    String fileName = fileList[i].getName();
                    if (fileList[i].isFile() &&
                        fileName.endsWith(".xml"))
                    {
                        // each file has its own collection
                        String fileColName =
                            collectionName + "/" + fileName.substring(
                            0, fileName.length() - 4);
                        fileIndexer.addFile(fileColName, fileList[i]);
                    }
                }
            }
```

```
        }
        catch (IllegalArgumentException e)
        {
            printUsage();
        }
        catch (NullPointerException e)
        {
            printUsage();
        }
        catch (Exception e)
        {
            e.printStackTrace();
            System.exit(-1);
        }
    }

    /**
     * Prints instructions on the use of command line parameters
     */
    private static void printUsage()
    {
        System.err.println("Illegal arguments. Usage:");
        System.err.println("FileIndexer [-r source directory]" +
            "[-c database collection] -uri [database URI] (optional)");
    }

    /**
     * Iterates over command line arguments in a string array
     * and stores them in to a hash map
     *
     * @param args the command line arguments to be hashed
     * @return HashMap of parameters
     */
    private static HashMap getOptions(String[] args)
    {
        HashMap options = new HashMap();
        ArrayList rootArgs = new ArrayList();
        options.put("rootArgs", rootArgs);
        for (int i=0; i<args.length; ++i)
        {
            if (args[i].charAt(0) != '-')
            {
                rootArgs.add(args[i]);
            }
            else if (i+1 < args.length)
            {
                options.put(args[i], args[++i]);
            }
            else
            {
                throw new IllegalArgumentException();
            }
        }
        return options;
    }

    /**
     * Adds a file to a named collection in the database
     *
     * @param collection name of the collection in which the file is to be added
     * @param fileName File-object
     */
    private void addFile(String collection, File fileName) throws Exception
    {
        // try to get collection
        Collection col =
            DatabaseManager.getCollection(URI + collection, user, pass);
        if(col == null)
```

```
        {
            // collection does not exist: get root collection and create
            // for simplicity, we assume that the new collection is a
            // direct child of the root collection, e.g. /db/test.
            Collection root =
                DatabaseManager.getCollection(URI + "/db", user, pass);
            CollectionManagementService mgtService =
                (CollectionManagementService)root.getService(
                "CollectionManagementService", "1.0");
            String newCollection = collection.substring("/db".length() + 1);
            col = mgtService.createCollection(newCollection);
        }

        // create new XMLResource; an id will be assigned to the new resource
        XMLResource document =
            (XMLResource)col.createResource(fileName.getName(), "XMLResource");

        if(!fileName.canRead())
        {
            System.out.println("cannot read file " +
                fileName.getAbsolutePath());
            return;
        }
        document.setContent(fileName);
        System.out.print("storing document " + document.getId() + "...");
        col.storeResource(document);
        System.out.println("ok.");
    }
}
```

Appendiks D - Søketjeneste

Søketjenesten er en sentral del av prototypen. I dette appendikset gis en kort forklaring av hvilke filer som inngår i søketjenesten. Deretter vil koden til mellomvaren presenteres. Siden søkegrensesnittet i all hovedsak er basert på gjenbruk vil mye av koden ikke presenteres her.

D-1 Filer som inngår i søketjenesten

Søkegrensesnittet og mellomvaren inneholder følgende filer:

```
avansert.php
hjelp.php
index.php
om.php
result.xsl

css/websok.css

include/common.inc
include/search.inc
include/sok.js
include/web_common.inc

include/eXist/eXist_soap.php
include/eXist/SOAP/nusoap.php
```

Filene avansert.php, hjelp.php og om.php er ikke forandret. Index.php er tilpasset en del til min mellomvare, derfor legges koden ved. Result.xsl brukes til omforming av søkeresultatene fra XML til HTML.

Stilarket websok.css er ikke endret og legges derfor ikke ved.

Filene common.inc og sok.js er ikke endret, mens web_common.inc kun inneholder en liten endring. Disse filene legges ikke ved. Search.inc utgjør en stor del av mellomvaren og er endret mye i forhold til filen med samme navn i [1]. Denne filen legges derfor ved. Det understrekes at en del av koden er gjenbrukt. Opprinnelig opphavsmann er Eskil Solvang.

Filene eXist_soap.php og nusoap.php er eksterne bibliotek som det ikke er gjort endringer i. Disse legges derfor ikke ved. De er tilgjengelig fra [14] og [15].

D-2 Kode for søketjenesten

D-2-1 index.php

```
<?php
/*
=========================
index.php
=========================
Creator: Eskil H. Solvang
Dato: Januar 2004
Oppdatert: 1.3.2004
Oppdatert: 2005 av Knut Ingebretsen
=========================
Funksjoner
 - Enkelt søkegrensesnitt
 - Behandle enkelt søk
 - Behandle avansert søk
=========================
```

```
*/

//Inkluderer filer med funksjoner
include 'include/common.inc';                  // DisplaySearchBar()
include 'include/search.inc';                  // search(), printsearchresults()
include 'include/web_common.inc';              // commonHeader(), commonFooter()

//Variabler
$param = $_GET;
$collection = "nb";
$existwsdl = "http://itvpc458.idi.ntnu.no:8080/exist/services/Query?wsdl";

//Skill enkelt og avansert søk
if ($_GET['adv'] == 1) {
   $section = 'sectiontwo';
   $title = 'Søkemotor | Avansert søk';
} else {
   $section = 'sectionone';
   $title = 'Søkemotor | Enkelt søk';
}

//Header
commonHeader($title, $section);

/*
================
Enkel søkeform
(vises uansett)
================
*/

print '

<!-- content -->
<div id="sok">
   <form action="'; $PHP_SELF; print '" method="get" name="simpleform">';
   DisplaySearchBar();

   print '
   </form>
</div>
';

//søkeparametere
if(!isset($hits))
   $hits = 10;
if(!isset($charset))
   $charset = "utf-8";

/*
================
Utfør enkelt søk
================
*/
if ($query != "") {// Søkefeltet $query inneholder verdi
   print '

   <!-- results -->
   <div id="resultater">
   ';
   PrintSearchResultTableStart();
   //Utfør søk
   $totalnumhits = SimpleSearch($query, $existwsdl, $collection);
   PrintSearchResultTableStop();

   print '</div>';
}
```

```
/*
====================
Utfør avansert søk
(hvis inputparameteret adv er satt til 1, dvs. input kommer fra avansert
søkeskjema)
====================
*/
else if ($_GET['adv'] == 1) {
   //Bygger opp en streng med alle GET-parametrene
   $urlparam = "";
   foreach ($_GET as $key => $value) {
      if ($key == 'submit') {
         echo '';
      } else {
         if($urlparam != "") {
            $urlparam = $urlparam.'&';
         }
         $urlparam = $urlparam.$key.'='.$value;
      }
   }

   //Lenke tilbake til avansert søkeside
   echo '<a href="avansert.php?'.$urlparam.'">Endre søkeparametre</a>';

   //Nullstiller query- og filtervariablene
   $query = "";
   $filter = "";

   //Behandling av søkeparametre + oppbygging av querystreng
   $kilde = $_GET['arkiv'];
   $tittel = $_GET['tittel'];
   $navn = $_GET['navn'];
   $utgiver = $_GET['utgiver'];
   $subject = $_GET['subject'];
   $abstract = $_GET['abstract'];
   $lang = $_GET['lang'];               // brukes ikke!

   $txt = $_GET['txt'];
   $snd = $_GET['snd'];
   $img = $_GET['img'];
   $vid = $_GET['vid'];

   $dato = $_GET['dato'];
   $fra = $_GET['fra'];
   $til = $_GET['til'];

   $hits = $_GET['hits'];

   /*
   Filter: Ressurstype
   Created: 21.1.2004, Eskil
   Updated: 22.5.2005, Knut (Tilpasset til eXist)
   Filtrerer søket avhengig av verdier i modsresourceType i søkeindeksen
   */
   $resourcefilter = "";
   if ($txt == "on") {$resourcefilter .= "text ";}
   if ($snd == "on") {$resourcefilter .= "sound ";}
   if ($img == "on") {$resourcefilter .= "still ";}
   if ($vid == "on") {$resourcefilter .= "moving ";}
   // Tar vekk whitespace på slutten
   $resourcefilter = trim($resourcefilter);

   /*
   Query: Dato
   Created: 21.1.2004, Eskil
   Updated: 22.5.2005, Knut
   Søker i feltet dateCreated (dato i heltallsform ååååmmdd) i metadatapostene
   -----
```

```
   Alle: Definerer start- og sluttdato her
   Eksakt: Søker etter ressurser med ett bestemt år, men bruker intervallet fraÅ0000
- fraÅ1231 for å få treff på alle datoer i betraktet år
   Mellom: Søker etter ressurser mellom to årstall, og benytter intervallet fraÅ0000
- tilÅ1231 for å få treff på alle datoer mellom starten på 'fra' og slutten på
'til'
   */
   if ($dato == "alle") { //søk i alle datoer
      //setter variabler for tidligste og seneste år
      $startdato = '18000000';
      $sluttdato = '20040000';
      //bygger query
      $datoquery = "[" . $startdato . ";" . $sluttdato. "]";
   } else if ($dato == "eksakt") { //søk etter eksakt dato
      $datoquery = "[" . $fra . "0000;" . $fra . "1231]";
   } else if ($dato == "mellom") { //søk mellom angitte datoer
      $datoquery = "[" . $fra . "0000;" . $til . "1231]";
   } else { //ingen datosøk
      $datoquery = "";
   }

   print "
   <!-- results -->
   <div id=\"resultater\">
   ";

   PrintSearchResultTableStart();
   //Utfør søk
   $totalnumhits = AdvancedSearch($kilde, $tittel, $navn, $utgiver, $subject,
$abstract, $resourcefilter, $datoquery, $existwsdl, $collection);
   PrintSearchResultTableStop();

   print "</div>";
}

//Footer
commonFooter();
?>
```

D-2-2 result.xsl

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
  xmlns:oai="http://www.openarchives.org/OAI/2.0/"
  xmlns:mods="http://www.loc.gov/mods/">
  <xsl:output method="html" encoding="ISO-8859-1" indent="no"
    omit-xml-declaration="yes" media-type="text/html"/>
  <xsl:template match="/">
    <table border="0">
      <xsl:apply-templates select="record"/>
    </table>
  </xsl:template>
  <!---------- Template:identifier ---------->
  <xsl:template match="oai:header">
    <tr>
      <td class="resultatterm">oaiidentifier</td>
      <td class="resultatverdi">
        <xsl:value-of select="oai:identifier"/>
      </td>
    </tr>
    <tr>
      <td class="resultatterm">oaisource</td>
      <td class="resultatverdi">
        <xsl:variable name="v_oaiid" select="oai:identifier"/>
        <xsl:value-of
          select="substring-before(substring-after($v_oaiid,'oai:'),'.nb')"/>
      </td>
    </tr>
```

```
</xsl:template>
<!--======== Template:metadata/mods:mods ========-->
<xsl:template match="metadata/mods:mods">
  <tr>
    <td class="resultatterm">mods:title</td>
    <td class="resultatverdi">
      <xsl:value-of select="mods:titleInfo/mods:title"/>
    </td>
  </tr>
  <tr>
    <td class="resultatterm">mods:subTitle</td>
    <td class="resultatverdi">
      <xsl:value-of select="mods:titleInfo/mods:subTitle"/>
    </td>
  </tr>
  <!-- Setter inn alle name med tilhørende namePart og role -->
  <xsl:choose>
    <xsl:when test="mods:name">
      <xsl:apply-templates select="mods:name"/>
    </xsl:when>
    <xsl:otherwise>
      <tr>
        <td class="resultatterm">mods:namePart</td>
        <td class="resultatverdi">
          <xsl:value-of select="mods:name/mods:namePart"/>
        </td>
      </tr>
      <tr>
        <td class="resultatterm">mods:role</td>
        <td class="resultatverdi">
          <xsl:value-of select="mods:name/mods:role/mods:text"/>
        </td>
      </tr>
    </xsl:otherwise>
  </xsl:choose>
  <tr>
    <td class="resultatterm">mods:typeOfResource</td>
    <td class="resultatverdi">
      <xsl:value-of select="mods:typeOfResource"/>
    </td>
  </tr>
  <tr>
    <td class="resultatterm">mods:genre</td>
    <td class="resultatverdi">
      <xsl:value-of select="mods:genre"/>
    </td>
  </tr>
  <tr>
    <td class="resultatterm">mods:publisher</td>
    <td class="resultatverdi">
      <xsl:value-of select="mods:originInfo/mods:publisher"/>
    </td>
  </tr>
  <tr>
    <td class="resultatterm">mods:internetMediaType</td>
    <td class="resultatverdi">
      <xsl:value-of select="mods:physicalDescription/mods:internetMediaType"/>
    </td>
  </tr>
  <tr>
    <td class="resultatterm">mods:extent</td>
    <td class="resultatverdi">
      <xsl:value-of select="mods:physicalDescription/mods:extent"/>
    </td>
  </tr>
  <tr>
    <td class="resultatterm">mods:abstract</td>
    <td class="resultatverdi">
```

```
        <xsl:value-of select="mods:abstract"/>
      </td>
    </tr>
    <tr>
      <td class="resultatterm">mods:subject</td>
      <td class="resultatverdi">
        <xsl:value-of select="mods:subject"/>
      </td>
    </tr>
    <!-- Setter inn alle mods:identifier -->
    <xsl:choose>
      <xsl:when test="mods:identifier">
        <xsl:apply-templates select="mods:identifier"/>
      </xsl:when>
      <xsl:otherwise>
        <tr>
          <td class="resultatterm">mods:identifier</td>
          <td class="resultatverdi">
            <xsl:value-of select="mods:identifier"/>
          </td>
        </tr>
      </xsl:otherwise>
    </xsl:choose>
    <tr>
      <td class="resultatterm">mods:dateCreated</td>
      <td class="resultatverdi">
        <xsl:value-of select="mods:originInfo/mods:dateCreated"/>
      </td>
    </tr>
    <tr>
      <td class="resultatterm">mods:accessCondition</td>
      <td class="resultatverdi">
        <xsl:value-of select="mods:accessCondition"/>
      </td>
    </tr>
  </xsl:template>
  <xsl:template match="mods:name">
    <tr>
      <td class="resultatterm">mods:namePart</td>
      <td class="resultatverdi">
        <xsl:value-of select="mods:namePart"/>
      </td>
    </tr>
    <tr>
      <td class="resultatterm">mods:role</td>
      <td class="resultatverdi">
        <xsl:value-of select="mods:role/mods:text"/>
      </td>
    </tr>
  </xsl:template>
  <xsl:template match="mods:identifier">
    <tr>
      <td class="resultatterm">mods:identifier</td>
      <td class="resultatverdi">
        <xsl:value-of select="."/>
      </td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```

D-2-3 search.inc

```
<?php

// Inkluderer eXist-SOAP
include ('include/eXist/eXist_soap.php');
```

```
/*
** Function: SimpleSearch
** Funksjonen er laget med utgangspunkt i Search() fra Eskil Solvang
** sitt søkegrensesnitt. Funksjonen er kraftig forkortet og det er kun
** noe av strukturen som gjenstår av den opprinnelige funksjonen.
** Parameters:
** $query: spørring for enkelt søk
** $existwsdl: URL of the WSDL-file description of the SOAP interface
** $collection: restrict to specified collection (in eXist DB)
*/
function SimpleSearch($query, $existwsdl, $collection)
{
   global $self;
   global $SERVER_ADDR;
   global $PHP_SELF;
   global $debug;

   $debug = false;

   if($debug)
   {
      print "query: $query<br>";
   }
   if($query == "")
   {
      $results = array();
      $results["numhits"] = 0;
   }
   else
   {
      if($debug)
      {
         print "$query<br>";
      }
      // Formuler XQuery
      $xquery = "for \$x in collection('$collection')//record ";
      $xquery .= "where ";
      if(strstr($query, '"'))
      {
         // For frasesøk, søker i tittel, subtittel, namepart, publisher, subject og
abstract
         $xquery .= "\$x/metadata/mods:mods/mods:titleInfo/mods:title[near(.,
'$query')] or ";
         $xquery .= "\$x/metadata/mods:mods/mods:titleInfo/mods:subTitle[near(.,
'$query')] or ";
         $xquery .= "\$x/metadata/mods:mods/mods:name/mods:namePart[near(.,
'$query')] or ";
         $xquery .= "\$x/metadata/mods:mods/mods:originInfo/mods:publisher[near(.,
'$query')] or ";
         $xquery .= "\$x/metadata/mods:mods/mods:subject[near(., '$query')] or ";
         $xquery .= "\$x/metadata/mods:mods/mods:abstract[near(., '$query')] ";
      }
      else
      {
         // For nøkkelordsøk (ikke frase)

         // TODO: Må endres slik at det bare søkes i de 5 feltene det skal søkes i
         $xquery .= "(\$x/metadata/mods:mods/mods:titleInfo/mods:title  | ";
         $xquery .= "\$x/metadata/mods:mods/mods:titleInfo/mods:subTitle | ";
         $xquery .= "\$x/metadata/mods:mods/mods:name/mods:namePart | ";
         $xquery .= "\$x/metadata/mods:mods/mods:originInfo/mods:publisher | ";
         $xquery .= "\$x/metadata/mods:mods/mods:subject | ";
         $xquery .= "\$x/metadata/mods:mods/mods:abstract)";
         $xquery .= " &= '$query' ";
      }
      // Legg til return-statement i XQuery-setningen
      $xquery .= "return \$x";
```

```
    if($debug)
    {
       print $xquery;
    }

    // Utfør XQuery mot eXist
    $results = PerformSearch($xquery, $existwsdl);
 }

 if(($results != 0) && !array_key_exists("error", $results))
 {
    $numhits   = $results["HITS"];
    $searchtime    = $results["QUERY_TIME"] / 1000;

    print '<table class=mainborder><tr><td width="100%" valign="top">';
    print "<p> <dl class='margins-removed'>" ;
    if($numhits == 0)
    {
       ?>
          <!-- Ingen søketreff -->
          <dt>Ingen dokumenter ble funnet.</dt>
          <dd>Forslag:
             <ul>
                <li>Forsikre deg om at alle søkeordene er stavet korrekt</li>
                <li>Bruk færre søkeord, eller erstatt noen av søkeordene</li>
             </ul>
          </dd>
       <?php
    }
    else
    {
       // Antall søketreff og søketid
       print "<dt>Søkeresultat:</dt>
       <dd><strong>$numhits</strong> dokumenter ble funnet - ";
       print "Søketid: <strong>$searchtime</strong> sekunder</dd>";

       // XQuery Result
       $xmlresults = $results["XML"];

       if ($debug)
       {
          print "<p><b>Result of the XQuery:</b></p>";
          print "<pre>";
          for($index = 0; $index < $numhits; $index++)
          {
             print (htmlspecialchars($xmlresults[$index]));
             print "\n\n";
          }
          print "</pre>";
       }
    }
    for($index = 0; $index < $numhits; $index++)
    {
       $hitentry = $xmlresults[$index];

       echo '<br>';
       print "<dd>Resultat ";
       print $index + 1;
       print " av $numhits</dd>";

       // Henter resultatet omformet til HTML
       $html = ProcessResult($hitentry);

       print $html;
    }
    echo " </dl></td>";
    print "</table>";
 }
```

```
   else //FEILMELDING
   {
      echo "<p>There is a temporary problem connecting to the search engine. Please
try again.</p>\n" ;
        if($results != 0)
        {
           echo "<p>Error code: " .$results["errorcode"] ."<br>\n";
           echo "Error text: " .$results["errortext"] ."</p>\n";
        }
   }
   flush();
   return($numhits) ;
}

/*
** Function: AdvancedSearch
** Funksjonen er laget med utgangspunkt i Search() fra Eskil Solvang
** sitt søkegrensesnitt. Funksjonen er kraftig forkortet og det er kun
** noe av strukturen som gjenstår av den opprinnelige funksjonen.
** Parameters:
** $kilde:
** $tittel:
** $navn:
** $utgiver:
** $subject:
** $abstract:
** $resourcefilter:
** $datoquery:
** $existwsdl: URL of the WSDL-file description of the SOAP interface
** $collection: restrict to specified collections (in eXist DB)
*/
function AdvancedSearch($kilde, $tittel, $navn, $utgiver, $subject, $abstract,
$resourcefilter, $datoquery, $existwsdl, $collection)
{
   global $self;
   global $SERVER_ADDR;
   global $PHP_SELF;
   global $debug;

   $debug = false;

   if($debug)
   {
      print "Kilde: $kilde<br>";
      print "Type: $resourcefilter<br>";
   }

   // Sjekker om kilde er valgt
   if ($kilde != 'alle')
      $collection .= "/$kilde";

   // Formuler XQuery
   $xquery = "for \$x in collection('$collection')//record ";
   $xquery .= "where ";

   // Sjekker om ressurstypen matcher
   if ($resourcefilter != "")
   {
      $xquery .= "\$x/metadata/mods:mods/mods:typeOfResource[. |= '$resourcefilter']
and ";
   }

   // Sjekker om søkefeltene har innhold og om det i tilfelle er fraser
   if($tittel != "")
   {
      if (strstr($tittel, '"'))
      {
```

```
        $xquery .= "(\$x/metadata/mods:mods/mods:titleInfo/mods:title[near(.,
'$tittel')] or ";
        $xquery .= "\$x/metadata/mods:mods/mods:titleInfo/mods:subTitle[near(.,
'$tittel')]) and";
      }
      else
      {
         // TODO: Skal finne alle ordene i $tittel selv om et
         //       f.eks. er i title og et annet i subtitle
         $xquery .= "(\$x/metadata/mods:mods/mods:titleInfo/mods:title[. &=
'$tittel'] or ";
         $xquery .= "\$x/metadata/mods:mods/mods:titleInfo/mods:subTitle[. &=
'$tittel']) and ";
      }
   }
   if($navn != "")
   {
      if (strstr($navn, '"'))
         $xquery .= "\$x/metadata/mods:mods/mods:name/mods:namePart[near(., '$navn')]
and ";
      else
         $xquery .= "\$x/metadata/mods:mods/mods:name/mods:namePart[. &= '$navn'] and
";
   }
   if($utgiver != "")
   {
      if (strstr($utgiver, '"'))
         $xquery .= "\$x/metadata/mods:mods/mods:originInfo/mods:publisher[near(.,
'$utgiver')] and ";
      else
         $xquery .= "\$x/metadata/mods:mods/mods:originInfo/mods:publisher[. &=
'$utgiver'] and ";
   }
   if($subject != "")
   {
      if (strstr($subject, '"'))
         $xquery .= "\$x/metadata/mods:mods/mods:subject[near(., '$subject')] and ";
      else
         $xquery .= "\$x/metadata/mods:mods/mods:subject[. &= '$subject'] and ";
   }
   if($abstract != "")
   {
      if (strstr($abstract, '"'))
         $xquery .= "\$x/metadata/mods:mods/mods:abstract[near(., '$abstract')] ";
      else
         $xquery .= "\$x/metadata/mods:mods/mods:abstract[. &= '$abstract'] ";
   }

   // Sjekker om $xquery slutter på "and ", tar i tilfelle denne vekk
   if(strrpos($xquery, "and ") == strlen($xquery) - 4)
      $xquery = substr($xquery, 0, strlen($xquery) - 4);

   $xquery .= "return \$x";

   if($debug)
   {
      print $xquery;
   }

   // Utfør XQuery mot eXist
   $results = PerformSearch($xquery, $existwsdl);

   if(($results != 0) && !array_key_exists("error", $results))
   {
      $numhits   = $results["HITS"];
      $searchtime    = $results["QUERY_TIME"] / 1000;

      print '<table class=mainborder><tr><td width="100%" valign="top">';
```

```
      print "<p> <dl class='margins-removed'>" ;
      if($numhits == 0)
      {
         ?>
            <!-- Ingen søketreff -->
            <dt>Ingen dokumenter ble funnet.</dt>
            <dd>Forslag:
               <ul>
                  <li>Forsikre deg om at alle søkeordene er stavet korrekt</li>
                  <li>Bruk færre søkeord, eller erstatt noen av søkeordene</li>
               </ul>
            </dd>
         <?php
      }
      else
      {
         // Antall søketreff og søketid
         print "<dt>Søkeresultat:</dt>
              <dd><strong>$numhits</strong> dokumenter ble funnet - Søketid:
<strong>$searchtime</strong> sekunder</dd>\n <br>\n" ;

         // XQuery Result
         $xmlresults = $results["XML"];
      }
      for($index = 0; $index < $numhits; $index++)
      {
         $hitentry = $xmlresults[$index];

         echo '<br>';
         print "<dd>Resultat ";
         print $index + 1;
         print " av $numhits</dd>";

         // Henter resultatet omformet til HTML
         $html = ProcessResult($hitentry);

         print $html;
      }
      echo " </dl></td>";
      print "</table>";
   }
   else //FEILMELDING
   {
      echo "<p>There is a temporary problem connecting to the search engine. Please
try again.</p>\n" ;
      if($results != 0)
      {
         echo "<p>Error code: " .$results["errorcode"] ."<br>\n";
         echo "Error text: " .$results["errortext"] ."</p>\n";
      }
   }
   flush();
   return($numhits) ;
}

/*
** Function: PerformSearch
** Parameters:
** $query: eXist XQuery string
** $host: host (URL) of the WSDL-file describing SOAP interface
** Returns:
** array containing the result in XML-format
*/
function PerformSearch($xquery, $host)
{
   global $debug;
   $result;
```

```
   $debug = false;

   if ($debug)
   {
      print "Host: $host<br>";
   }

   // eXist-SOAP
   $db = new eXist_SOAP("guest", "guest", $host);

   $isConn = false;

   if (!$db->getError())
   {
      $isConn = $db->connect();
   }
   else
   {
      $result = array(
         "error" => $db->getError()
      );
      return $result;
   }
   if (isConn)
   {
      $result = $db->xquery($xquery);

      if ($debug)
      {
         // XQuery Result
         print "<p><b>Result of the XQuery:</b></p>";
         print "<pre>";
         print (htmlspecialchars(implode("\n\n", $result["XML"])));
         print "</pre>";
      }

      // kobler fra databasen
      $db->disconnect();
   }
   else
   {
      $result = array(
         "error" => $db->getError()
      );
   }
   return $result;
}

/*
** Function: ProcessResult
** Parameters:
** $xml: result in XML-format
** Returns:
** string containing result in HTML-format
*/
function ProcessResult($xml)
{
   // Setter på XML-deklarering med encoding ISO-8859-1
   // slik at Sablotron XSLT takler æ, ø og å
   $xml = '<?xml version="1.0" encoding="ISO-8859-1"?>' . $xml;

   $arguments = array(
      '/_xml' => $xml
   );

   // Allocate a new XSLT processor
   $xh = xslt_create();
```

```
   // Process the document, returning the result into the $result variable
   $result = xslt_process($xh, 'arg:/_xml', 'result.xsl', NULL, $arguments);

   return $result;
}
?>
```

Appendiks E - Vedlagt CD-ROM

Koden som er utviklet i denne oppgaven er lagt ved i appendiksene ovenfor. Imidlertid er det en del kode som brukes i søketjenesten som ikke er lagt ved, fordi den ikke er utviklet av undertegnede. Fordi den koden som er gjenbrukt fra [1] brukes i søketjenesten legges både den koden og koden som er utviklet i denne oppgaven med på CD-ROM.

Se readme-fil på CD-ROM for nærmere forklaring av de enkelte filene.