NTNU
Det skapende universitet

Indeksering av heterogene XML dokumenter ved hjelp av datatyper fra XML Schema

Trond Aksel Myklebust

Master i informatikk
Oppgaven levert: Mai 2006
Hovedveileder: Trond Aalberg, IDI

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

"Change the world before someone else does it for you."

- J. Michael Straczynski

Forord

Denne rapporten er resultatet av en masteroppgave utført ved NTNU sin informasjonsforvaltningsgruppe høsten og våren 05/06. Veileder for oppgaven var Trond Aalberg.

Utgangspunktet for oppgaven var et ønske om å studere nærmere hvilke utfordringer informasjonsgjenfinning i semistrukturerte dokumenter stiller et system kontra et ment for ustrukturerte tekstdokumenter.

Siden oppgaven var ganske udefinert i begynnelsen har mye arbeid blitt gjort for å få innsikt i fagområdet og dermed finne en aktuell problemstilling. Dette har hjulpet til med formuleringen av oppgaven og økt forståelsen for faget. Det må likevel nevnes at oppgaven bare tar for seg en liten del av de utfordringer som eksisterer.

Resultatet har blitt denne rapporten som beskriver en spesifikk utfordring, ser på teori og tidligere arbeid, og foreslår og implementerer en løsning på problembeskrivelsen slik at løsningen kan undersøkes oppimot målsetningen.

Takk til Trond Aalberg som veileder for gode råd og tips underveis når noe var uklart rundt videre arbeid. En spesiell takk gis for å hele tiden ha innehatt en positiv innstilling til oppgaven og dens utvikling.

Trond Aksel Myklebust
Fosnavåg / Trondheim våren 2006

Oppsummering

Denne masteroppgaven foreslår og undersøker en metode for hvordan informasjons-gjenfinning i heterogene XML dokumenter kan gjøres ved å differensiere indekserings-prosessen ut i fra datatyper angitt i tilhørende XML Schema. Målet er å tilby bedre søkemuligheter for informasjonssøkere ved å muliggjøre spørringer som er uavhengige av elementnavn i en samling av forskjellig strukturerte dokumenter.

Informasjonssøking foregår i dag primært i ustrukturerte dokumenter der betydningen av innholdet ikke er direkte kjent. Dette krever kompliserte og unøyaktige tolkninger av innholdet for å kunne trekke ut hva som er hva og hvordan dokumentene best mulig kan indekseres. En stadig økende mengde produsert informasjon og metadata gjør dette til en krevende prosess å utføre manuelt. Det trengs derfor nye metoder der innholdet blir beskrevet ved produksjonstidspunktet slik at en datamaskin automatisk kan forstå dokumentenes innhold.

Semistrukturerte dokumentformater som XML inneholder støtte for spesifisering av slik informasjon og muliggjør differensiert indeksering av innholdet basert på annotert informasjon. Dette gjør mer detaljerte spørringer enn tidligere mulig men stiller nye krav til de metoder som brukes for å indeksere dokumentene.

En av de største utfordringene er å lokalisere og tolke den informasjonen som øker kvaliteten på resultatet av et søk uten at noe informasjon forsvinner. Informasjonen eksisterer ikke i en flat tekstfil, men inneholder distinkte datatyper som må behandles individuelt. Dette krever nye metoder som muliggjør indeksering basert på denne informasjonen.

I denne oppgaven presenteres et forslag til et system som indekserer XML dokumenter ved å tolke tilhørende XML Schema inneholdende annotasjoner av datatype og dataformat. Ved å bruke for hvert element denne informasjonen er ønsket at indekseringen gjøres ved å automatisk normalisere elementinnholdet ut i fra angitt format og datatype. Søk kan dermed optimaliseres basert på datatype uavhengig av om originalt format og dokumentstruktur er forskjellig.

Testing av systemet er gjennomført for å finne ut hvordan eksisterende XML dokumenter støtter denne typen indeksering og eventuelle løsninger for hvordan det kan gjøres bedre.

Utkommet fra arbeidet på oppgaven og hovedkonklusjonen er at den foreslåtte metoden fungerer godt som løsning på problemstillingen, gitt at de eksterne data som brukes er strukturert slik at datatyper kan defineres for innholdet.

Summary

This master thesis suggests and investigates a method for supporting information retrieval in heterogeneous XML documents by varying how the content of the documents are indexed based on data types given in a paired XML Schema. The goal is to provide better searching for information seekers by allowing queries independent of element name across semi structured heterogeneous documents.

Information retrieval today focuses primarily on unstructured documents where the context of the information is unknown. This requires complex and unreliable parsing of the content to extract the meaning and to improve the indexing of the documents. A continuously increasing production of information and metadata makes this a tiresome process to do manually. There is therefore a need for new methods that allows annotating the content when created so that a computer automatically can understand it.

Standards as XML for semi-structured documents contains support for describing the needed information and makes differentiated indexing based on the annotated information possible. This makes creating more detailed queries than before possible. At the same time, it creates new demands on the methods used for indexing the documents.

One of the main challenges is to extract and make available in an efficient way that information which can increase the quality of a search result without any loss of information. The information does not exist in a flat file of text, but contains distinct data types requiring individual treatment. This requires new methods for supporting indexation based on this information.

This assignment suggests a system that indexes XML documents by parsing the paired XML Schema containing annotations of data type and data format. By using for each element this information the goal is to improve indexing by automatically normalize the data based on the defined format and data type. Search quality can then be improved using data types even though the original format and document structure may vary.

The system is tested to see how existing XML documents allows this kind of indexation and eventual solutions to how it can be improved.

The outcome of this thesis and the main conclusion is that the idea suggested works as a solution to the problem formulated, given that the structure of the external data supports defining data types for the content.

Innhold

Figurliste

Tabell-liste

1 Innledning

Behovet for å ta vare på informasjon og gjøre den tilgjengelig for senere bruk øker i takt med en stadig økende produksjon av digitale dokumenter. En større mengde informasjon gjør at også tiden som behøves for å lokalisere og fylle informasjonsbehovet øker. Eksisterende systemer møter et stadig økende behov for å slå sammen informasjon presentert på forskjellige måter, enten det er bilder, tekst, lyd eller video.

Det er derfor et behov for nye metoder og systemer som kan gjøre informasjon lettere tilgjengelig og med stor grad av nøyaktighet kunne skille relevant fra urelevant informasjon. Forskjellige informasjonstyper krever i tillegg ulike strategier for hvordan de behandles for å kunne øke søkekvaliteten i store datasamlinger.

Blant de muligheter det er ønskelig at neste generasjons systemer for informasjons-gjenfinning skal kunne tilby er bedre støtte til å formulere i hvilken kontekst en spørring gjelder; Søk etter ”Hamburger” og få tilbake dokumenter sortert etter om det er snakk om en person fra Hamburg eller mat.

Selv om mulighetene er store, er det likevel ikke mye som kan gjøres dersom ikke dokumenter blir publisert med en semantikk i seg som gjør dette mulig. Storparten av de dokumenter som i dag er tilgjengelig på Internett inneholder ikke den informasjonen som trengs for å kunne gjøre dette på en effektiv måte. Nye metoder for hvordan informasjon kan gis en semantikk forståbar av en maskin har derfor blitt foreslått. Likevel eksisterer ingen standard for hvordan søk i dokumenter basert på denne type formater effektivt kan gjøres. Ad hoc løsninger har dermed skapt integrerings- og interoperabilitetsproblemer og viser at det er nødvendig med standardiserte metoder.

De utfordringer som søk i strukturerte dokumenter står ovenfor blir primært forsøkt løst blant annet ved bruk av XML til å gi tekst semantikk og struktur. XML er opphavlig en del av den større SGML [1] standarden og har fått en viktig rolle innenfor informasjonsgjenfinning i semistrukturerte dokumenter. Mye av grunnen er de mulighetene XML gir til å øke en datamaskins forståelse for hvordan et dokument skal forstås. Muligheten en bruker har for spesifisering av egen struktur og innhold i XML er en av utfordringene som er bakgrunnen for denne oppgaven.

Utgangspunktet for oppgaven er at en samling av homogene XML dokumenter består av dokumenter basert på samme struktur. Dette betyr at like elementer i disse dokumentene kan antas å ha lik type innhold. Om mange XML samlinger med forskjellige struktur blir plassert sammen fås en heterogen samling. Dersom da deler av dokumentene har lik struktur inneholder de like elementer, men kan ha et innhold som har forskjellig mening. Dette skaper åpenbare problemer ved indeksering og søking i slike samlinger.

1.1 Problembeskrivelse

Semistrukturerte dokumenter muliggjør å annotere innholdet med informasjon utover hva som er ment for presentasjon. Dette kan brukes til å angi hvordan dokumentene skal behandles ved indeksering av innholdet.

For å kunne utføre gode søk i heterogene semistrukturerte dokumenter er det en fordel å kunne gjøre søk etter innhold med lik mening uten kjennskap til strukturelle forhold. Dette er ikke alltid enkelt siden strukturen varierer og det er derfor nødvendig med metoder som kan identifisere likt innhold til tross for ulik struktur. Basis for oppgaven er derfor noen av utfordringene og behovene som er identifisert og blant annet utdypet av Liu, S. et al.:

> *[..] In addition, traditional IR has only one data type, i.e., plain text, while XML may contain data of various types, such as plain text, numbers, date and time. Thus we need multiple content processing methods and indexing types for the heterogeneous contents in XML documents to support various search predicates. Further, not all tags in an XML document are semantically meaningful.* [2]

For å støtte skisserte behov må det være mulig å identifisere meningen med de forskjellige delene av et dokuments innhold. Det er derfor nødvendig med en måte å kunne formidle informasjonen om innholdets mening ifra de som lager dokumentene til de systemer som gjør innholdet søkbart.

De fleste systemer krever at sammenligningen av heterogene samlinger gjøres manuelt. Administrator må dermed gjennomgå dokumentene og identifisere struktur og innhold. En slik tilnærming er tidkrevende og lite realistisk på annet enn et lite antall dokumentsamlinger. Metoder som kan utføre dette automatisk er en utfordring som ikke er løst på en tilfredsstillende måte.

Ettersom hver enkelt samling av homogene semistrukturerte dokumenter har en felles struktur unikt for hver samling øker dette vanskelighetsgraden for heterogene samlinger siden et element i en samling ikke nødvendigvis har samme meningen i en annen.

Ved søk i et informasjonsgjenfinningssystem er det ikke alltid det er ønskelig at et søk returnerer kun eksakte treff på hva det søkes etter. Ved større forståelse for et dokuments innhold kan det lettere returneres treff i områder rundt hva det faktisk søkes etter.

1.2 Målsetning

I denne oppgaven skal det foreslås en metode for hvordan lik type informasjon i forskjellig strukturerte dokumenter kan søkes etter der søker ikke vet hvordan dokumentene er strukturert.

Det overordna målet er å vise at det er mulig å indeksere heterogene XML dokumenter uavhengig av elementnavn. Istedenfor skal informasjonen som beskriver innholdet gis som skjemainformasjon. Å relatere innhold basert på blant annet datatyper kan være en mulig løsning siden en datatype kan begrense et søke til dokumenter inneholdende en spesifikk type informasjon. Et hovedmål med oppgaven er derfor å se på

hvor godt eksisterende standarder og teknologier, inkludert XML dokumenter slik de er oppbygd i dag, støtter denne metoden med informasjonsgjenfinning som mål. I tillegg er følgende delmål aktuelle å oppnå eller å kunne svare på.

- Studere noen aktuelle XML teknologier og utnytte noen av de muligheter som eksisterer.
- Kan heterogene XML dokumenter homogent indekseres basert på datatyper ved bruk av informasjon fra tilhørende skjema?
- Muliggjør eksisterende XML samlinger slik indeksering?
- Hvordan fungerer det å bruke en søkemotor primært for ustrukturert tekst til å indeksere og søke i XML ut i fra datatyper?

For å kunne svare på de ovennevnte spørsmålene skal en prototyp på et system som nytter skjemainformasjon til å skille innholdet i dokumenter under indeksering utvikles. Systemet skal deretter utprøves på en samling heterogene dokumenter for å ha et grunnlag for evaluering av de brukte metoder.

1.3 Eksempel på behov

Som innledningen og problemformuleringen sier er det ønskelig å kunne søke etter dokumenter med et spesifikt informasjonsinnhold selv om de har en oppbygging som gjør at dette ikke automatisk kan gjøres på tvers av dokumentstruktur. Noen scenarioer sees derfor på nedenfor. Felles for alle scenarioer er behovene for metoder som muliggjør dem uten at søker må angi hvor i hver enkelt samling det er ønskelig å finne treff.

Variering av operasjoner på tekst
I mange tilfeller er det ønskelig å kunne søke etter spesifikke termer som for eksempel navn. De aller fleste søkesystemer foretar forskjellige operasjoner på all tekst, blant annet forskjellige former for stemming. Dette er ikke hensiktsmessig å gjøre på navn siden deler av termen da kan bli fjernet. Et eksempel er søk etter ”Hansen” som vil returnere treff på ”Hans” siden Hansen med bruk av en norsk stemmer kan bli stemmet til Hans. Metoder som muliggjør å hindre dette for spesifikke lokasjoner i strukturerte dokumenter er derfor hensiktsmessig.

Normalisering av data
Datoer oppgis i mange forskjellige formater og det er vanskelig å identifisere hvilket. Det beste eksempelet på dette er kanskje amerikanske og europeiske datoer som bytter om på plasseringen av måned og dag. Det er derfor nødvendig å kunne definere hvordan forskjellige dokumenter angir datoformat slik at det er mulig å kunne søke på datoer uansett originalt format.

Omtrentlige søk
Det er ofte ønskelig å kunne søke etter dokumenter som angir spesifikke årstall for en hendelse. Av og til kan søker være usikker på årstallet og det er ønskelig å kunne

returnere treff med årstall omtrentlig som angitt i spørringen. Dette krever at det er mulig å skille et tall som er et årstall og et som er et vanlig tall.

1.4 Avgrensninger

Løsningen skal være basert på bruk av forskjellige heterogene samlinger av homogene XML dokumenter. Dokumentene skal bestå av både metadataformater og fulltekst XML dokumenter der XML skjema kan brukes til å definere datatyper. Valg av testsamlinger må velges ut i fra hva som er hensiktsmessig for å få utprøvd en prototyp av løsningen som foreslås. Samlingene må gjenspeile dokumenter det er relevant å ha behov for å kunne søke i.

Søk over de indekserte data må kunne gjøres uten forutsetninger om hvordan de forskjellige dokumentene er strukturelt oppbygd. Indekseringen må benytte seg av standardiserte tekstprosesseringsmetoder ut i fra datatype og muliggjøre datatypespesifikke søkemetoder.

Basert på en lokal samling av dokumenter skal dokumentinnholdet indekseres og gjøres søkbart ut i fra datatyper igjennom et enkelt grensesnitt. Søkegrensesnittet må støtte søking i de data som har blitt indeksert og blitt konfigurert som søkbare og støtte vanlige søkeoperatorer. Visning av hele dokumenter og deler av de som angitt i søkeresultatet skal støttes.

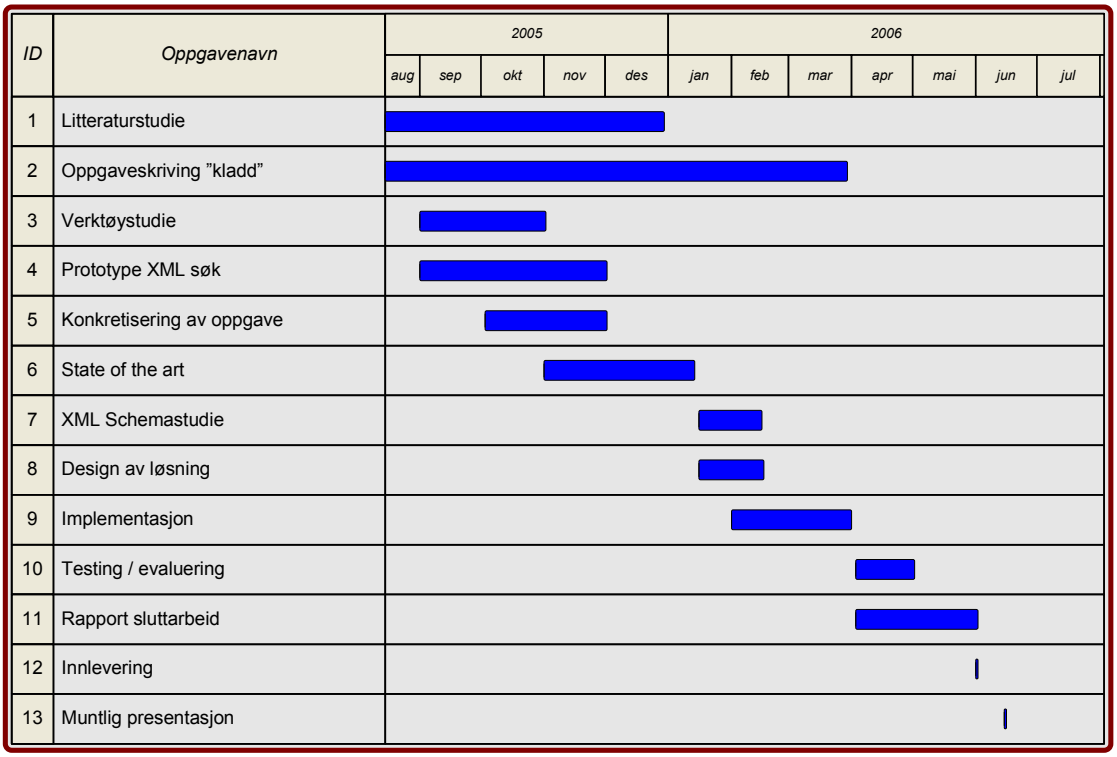
Utviklet prototyp skal så langt praktisk mulig basere seg på eksisterende komponenter og kunne fungere på forskjellige plattformer. Standardiserte XML teknologier skal brukes der hensiktsmessig.

Siden søking i XML basert på bruk av trestruktur ikke er en del av oppgavens mål er dette noe som ikke legges vekt på. Oppsummert må følgende punkt fokuseres på ut i fra problembeskrivelse, målsetning og avgrensningene angitt over:

- Se på noen eksisterende systemer som kan brukes for å utføre søk i XML dokumenter.
- Se på hvilke datatyper som er mulig og naturlig å indeksere, og hvordan det kan gjøres.
- Se på standarder for XML skjema, hva inneholder de av støtte for datatypebasert indeksering og hvilken tilbyr nødvendig funksjonalitet.
- Det skal utvikles en prototyp av et indekserings- og søkesystem for XML basert på studier og analyse av ovennevnte punkter.

1.5 Tilnærming til oppgaven

Hvordan oppgaven er tilnærmet og de forskjellige fasene arbeidet har vært igjennom sammen med en beskrivelse av hver fase er angitt nedenfor.



Tabell 1 - Tilnærmingsmåte

1. Under oppstartsfasen vart litteratur studert for å kunne komme frem til en problemstilling og tilegne seg kunnskap om oppgavens tema.
2. Under hele prosjektets planleggings- og implementasjonsfase vart informasjon og erfaringer fortløpende nedtegnet slik at informasjonen kunne samles opp og struktureres etter type. Fullføring av rapporten vart dermed lettere siden all nødvendig informasjon eksisterte i ett dokument.
3. En studie av hvilke eksisterende verktøy som var aktuelle å bruke under implementasjonen vart gjennomført basert på den foreløpige litteraturstudien og hva som faktisk var tilgjengelig for bruk.
4. En prototyp som muliggjorde søk i INEX samlingen basert på indeksering av alt som tekst men resultat basert på enkeltelement vart utviklet for å få føling med teknologier, teorier og problemer som eksisterte rundt søk i semistrukturerte samlinger.
5. Konkretisering av oppgaven basert på identifiserte utfordringer så langt i tilnærmingen.
6. For å kunne utvikle en effektiv løsning måtte noen eksisterende systemer studeres for å finne ut hva som eksisterte og manglet innenfor indeksering av XML dokumenter og hvordan de som eventuelt eksisterte brukte datatyper.

7. En studie av hvilke muligheter eksisterende XML Schema gav og hva som eventuelt manglet med tanke på problembeskrivelsen.
8. Design av en løsning på oppgaven.
9. Implementasjon av løsning på oppgaven i form av en prototyp.
10. Testing av implementert prototyp og evaluering av løsningen
11. Sluttføring av rapporten basert på arbeidsdokumentet.
12. Innlevering av rapport 1. Juni 2006.
13. Muntlig presentasjon / eksaminasjon etter levering.

1.6 Rapportens oppbygging

I denne rapporten undersøkes følgende tema: Først gjennomgåes det teoretiske rundt informasjonsgjenfinning med fokus på XML før en studie av eksisterende systemer for indeksering og søking i XML gjøres. Dette for å se hva som allerede eksisterer, kunne identifisere problemer som kan oppstå ved egen implementasjon, andre lignende forsøk på løsning av oppgavens problemstilling og mangler ved eksisterende systemer.

Deretter er de krav stilt til det systemet som det skal utvikles en prototyp for analysert. Implementasjonen av prototypen og de forskjellige delkomponentene gjennomgås før testing og en evaluering av systemet gjennomføres. Testingen utføres for å kunne gi en konklusjon på om prototypen overholder de krav og kan brukes for evaluering av foreslått løsning.

Til slutt følger referanseliste og vedlegg med blant annet utviklet kode og datagrunnlag.

2 Informasjonsgjenfinning

Arbeid kan primært utføres på to forskjellige måter; Enten manuelt av en person eller automatisk av en maskin. Tilsvarende er også tilfelle for informasjonsgjenfinning Hva sm er forskjellen på begrepene er derfor nødvendig å klargjøre.

Manuell gjenfinning kan være en bibliotekar på et bibliotek som lager katalogkort for hver bok ut i fra Dewey systemet [3], der kortene deretter brukes til å gjenfinne bøker ved å manuelt lokalisere bøker ut i fra de. Automatisk gjenfinning derimot gjøres av en datamaskin uten menneskelig innblanding utover å initiere prosessen.

Et annet viktig begrep er forskjellen på datagjenfinning kontra informasjons-gjenfinning. Det kan noen ganger være vanskelig å definere ettersom det ikke alltid er like lett å se forskjell på data og informasjon. To beskrivelser på forskjellen er.

> *Information retrieval (IR) is the art and science of searching for information in documents, searching for documents themselves, searching for metadata which describes documents, or searching within databases, whether relational stand alone databases or hypertext networked databases such as the Internet or intranets, for text, sound, images or data. There is a common confusion, however, between data retrieval, document retrieval, information retrieval, and text retrieval, and each of these have their own bodies of literature, theory, praxis and technologies.* [4]

> *'Information retrieval is the term conventionally, though somewhat inaccurately, applied to the type of activity discussed in this volume. An information retrieval system does not inform (i.e. change the knowledge of) the user on the subject of his inquiry. It merely informs on the existence (or non-existence) and whereabouts of documents relating to his request.'* [5]

Ut i fra deres definisjoner kan datagjenfinning sies å finne igjen data der du vet hvor det finnes, mens informasjonsgjenfinning går ut på å finne lokasjonen til informasjon om noe du vet er. Rijsberg angir i sin sammenligning mer detaljert åtte forskjeller mellom data- og informasjonsgjenfinning:

Type	Datagjenfinning	Informasjonsgjenfinning
Treffmodell	Eksakte treff	Delvis, beste treff
Inferens	Deduksjon	Induksjon
Modell	Deterministisk	Sannsynlighetsbasert
Klassifisering	Mono	Flere
Spørrespråk	Kunstig	Naturlig
Spørring	Komplett	Ukomplett
Ønskede treff	Nøyaktige	Relevante
Feilmelding	Sensitiv	Usensitiv

Tabell 2 - Datagjenfinning versus Informasjonsgjenfinning

Informasjonsgjenfinning som fag tar for seg alle de prosesser som er nødvendige for å lokalisere, ta vare på og tilgjengliggjøre informasjon. Selv om denne oppgaven fokuserer på tekst og andre nærliggende datatyper, tar faget som helhet for seg alle dokumenttyper ifra ustrukturert tekst til bilder, lyd og video. Hver dokumenttype krever sine egne metoder men de overordnede prosessene er de samme; dokumentene må lokaliseres, lagres, indekseres, kunne søkes i og hentes frem igjen basert på søk.

I dette kapittelet blir forskjellige områder innenfor informasjonsgjenfinning som på en eller annen måte er aktuelle for oppgaven, gjennomgått for å kunne gi en oversikt over teorigrunnlaget som sammen med state-of-the-art ligger bak valgt løsning. Noen av de gjennomgåtte delene, slik som spørrespråk er ikke direkte en del av oppgaven men er en viktig del av et system for informasjonsgjenfinning. Hovedgrunnen til at det likevel sees på er at for mange systemer er det den informasjonen som er lettest tilgjengelig. Igjennom å studere den kan også informasjon som muliggjør identifisering av hva som er viktig å støtte av indekseringsfunksjonalitet indirekte identifiseres.

2.1 Datatyper og informasjonsgjenfinning

I denne oppgaven er bruk av datatyper sentralt, noe som gjør en definisjon av hva en datatype kan defineres som naturlig å undersøke. Wikipedia [6] definerer en datatype innenfor datavitenskap som et navn på et sett av verdier sammen med de operasjoner som kan gjennomføres på verdiene. Dictionary.com har en lignende beskrivelse som i tillegg definerer at et navn også kan være en datatype.

1. *In programming, a classification identifying one of various types of data, as floating-point, integer, or Boolean, stating the possible values for that type, the operations that can be done on that type, and the way the values of that type are stored.*
2. *In databases or spreadsheets, a classification identifying one of various kinds of data, as a name, date, or dollar amount, found in a specific data field.* [7]

Det definisjonene foreslår gjør at ikke bare vanlige datatyper som tekst, boolske verdier, flyttal og andre enkle datatyper tilgjengelige i en relasjonsdatabase kan ansees å være relevante å undersøke i denne oppgaven. Siden definisjonene sier at et navn kan være en datatype kan dette begrepet også føres videre; det kan antas at en datatype kan være en dato, en liste med ord, en Internettlink, frase eller et årstall. En slik løs definisjon muliggjør også at om ønskelig kan en dokumenttittel kalles en datatype ved behov.

Felles for det som her er nevnt som datatyper er at de har et sett med verdier og definerte operatorer som kan gjennomføres på dem. På et navn kan det for eksempel være ønskelig å ikke utføre stemming siden det kan ødelegge gjenfinningen av navnet, og om enn vanskeligere å definere kan et begrenset verdiområde for dette feltet angis. Verdiområdet er dog ikke i alle tilfeller like interessant, men for tittel kan dette for eksempel defineres som alle kombinasjoner av bokstavene A-Z dersom ønskelig.

2.2 Tradisjonell informasjonsgjenfinning

Automatisert informasjonsgjenfinning av tekst vart utviklet for å kunne håndtere de enorme mengdene digital tekst som har blitt produsert siden de første datamaskinene kom for ca 60 år siden. Formålet er å forvalte tekstdokumenter på en slik måte at de som er relevante for en brukers informasjonsbehov raskt kan bli lokalisert ved behov [8].

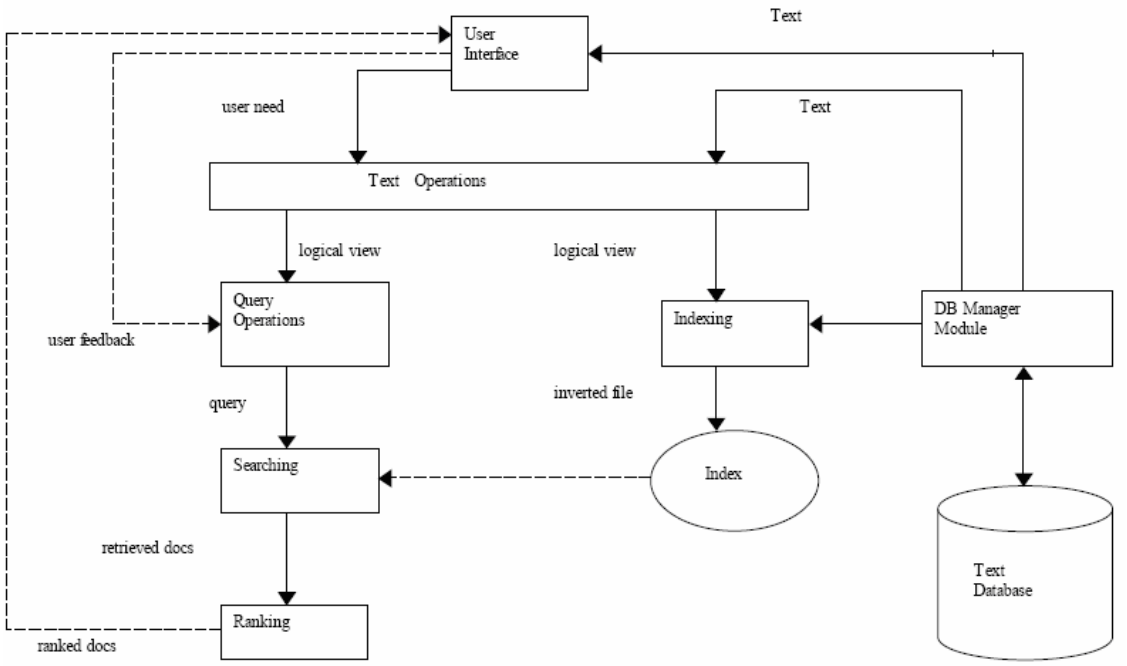
Siden 60-tallet har SMART systemet [9] og videreutviklinger av teknologier dette systemet tok i bruk spilt en viktig rolle, og da spesielt vektormodellen for vekting av relevansen til dokumenter. De senere årene har TREC [10] konferansen vært drivkraften bak mye av forskningen innen videreutvikling av tradisjonell informasjonsgjenfinning.

Tradisjonelt har informasjonsforvaltning tatt for seg gjenfinning av hele dokumenter som ikke har inneholdt noen spesifikk struktur som beskriver hva som er hva. Grunnen til at dette har vært den rådende disiplinen inntil nylig har blant annet vært begrenset datakraft og lagringsplass til å lagre noe ekstra informasjon utover det som er absolutt nødvendig for en brukers behov. I tillegg har også mangelen på standardiserte formater for spesifisering av struktur vært avgjørende.

Siden tradisjonell informasjonsgjenfinning tar for seg ustrukturerte dokumenter er indeksering i stor grad en triviell ting grunnet mangelen på noe hierarki som må tas med i betraktningen. Uten struktur eksisterer det heller ikke noe utover selve innholdet som kan angi meningen med teksten og føre til behov for konfigurerbar og differensiert indeksering. Fokuset har derfor vært på presentasjon av dokumenter, spørresyntaks og hvordan beregne likhet imellom en spørring og et dokument [8]. De fire vanligste modellene for dette er boolsk, vektor, sannsynlighetsbasert (probabilistic) og klusterbasert.

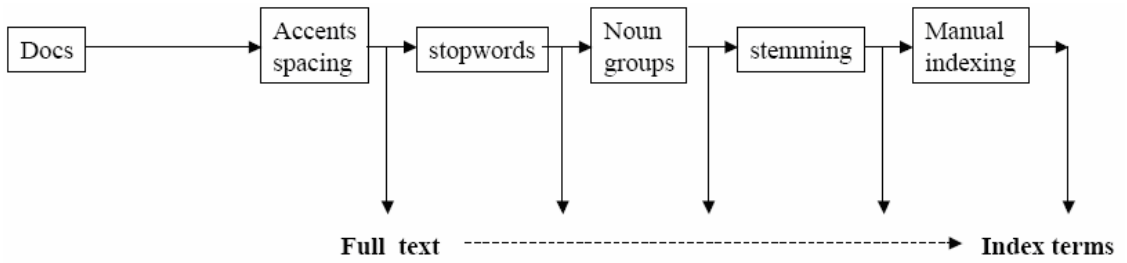
De overordnede prosessene i et system for informasjonsgjenfinning er angitt i illustrasjonen [11] under. I denne illustrasjonen hentes det data ifra en database med dokumenter som deretter gjennomgår tekstoperasjoner som begrenser mengden data som skal indekseres til de mest relevante delene. En Indekser indekserer så hver enkelt term og lagrer de i en invertert fil.

Ved søking formulerer en bruker en spørring igjennom et brukergrensesnitt som deretter gjennomgår de samme tekstoperasjonene som under indeksering. Dette gjøres for at det ikke skal søkes etter stoppord dersom de er fjernet fra indeksen. En spørretolker transformerer deretter spørringen til et format søkemotoren forstår. Søkemotoren mottar spørringen og finner dokumentene som passer til spørringen i den inverterte filen. Treffene går så igjennom en rangeringsalgoritme som returnerer en liste over dokumentene rangert etter algoritmen. Om ønskelig kan deretter bruker reformulere spørringen basert på mottatt sækeresultat eller velge å gjenhente fulltekst av dokumenter i resultatlisten.



Figur 1 - Systemoversikt for tradisjonelt informasjonsgjenfinningssystem

De basiske konseptene som angitt under [11] viser hvordan et dokument blir sett på og behandlet slik som angitt i figuren over for "Text Operation". Alle tekstoperasjoner blir her brukt over hele teksten.



Figur 2 - Prosesseringsmetoder ved indeksering av ustrukturert tekst

Figuren viser at under indeksering av ustrukturerte dokumenter fjernes først de tegn som av systemet tolkes som skilletegn, altså punktum, komma og andre. Deretter fjernes stoppord, substantivgrupper identifiseres, stemming utføres og eventuell manuell indeksering om nødvendig slik at en står igjen med ideelt sett kun ønskede indekstermer.

2.2.1 Datatyper i ustrukturerte dokumenter

Selv om ustrukturerte dokumenter mangler struktur som kan nyttes til å angi datatyper, eksisterer det likevel elementer som kan identifiseres som datatyper. Men grunnet problemer med å identifisere disse ser de fleste systemer på alt som tekst, og eventuelt

tall. Er det ønskelig å støtte mer komplekse datatyper kreves det at analyser av dokumentene gjøres for å tolke konteksten tall og tekst står i.

Ved indeksering av alt innhold som tekst og eventuelt tall brukes de samme indekseringsteknikkene over hele dokumentinnholdet. Indekseringsmetodene varierer altså ikke basert på hvor i dokumentet eller på typen innhold. De forskjellige teknikkene for tekstbehandling som stemming og stoppordfjerning brukes universelt grunnet manglende informasjon som kan fortelle hvor indekseringen skal gjøres annerledes. Dette kan spesielt skape problemer ved søk etter navn siden det kan være en risiko for å få treff på termer som grunnet stemming ikke har noe med hva det søkes etter. Er det ønskelig å søke etter etternavnet Hansen kan resultatlisten inneholde både treff på Hansen og Hans slik tidligere nevnt.

Tall er en annen enkel datatype som det kan være støtte for søk etter. Tilsvarende som med tekst er det lett å identifisere tall. Det vil da kunne være mulig å søke etter treff innenfor intervaller og omtrentlige verdier. Bare søk etter tall i ustrukturerte dokumenter kan likevel ha liten verdi om det ikke er noen mulighet til å vite hva tallet faktisk gjelder. Pris og årstall er to eksempel på hvor det er en fordel å vite dette.

Siden angivelse av kontekst mangler må det tas i bruk lingvistisk analyse av innholdet dersom det er ønskelig å finne ut i hvilken kontekst de forskjellige ordene eksisterer. Om ikke kan søk etter noe annet enn enkeltord eller fraser ikke gjennomføres effektivt. Indekser for tekstdokumenter har derfor i all hovedsak begrenset seg til å støtte raske oppslag basert på ord i inverterte lister.

2.2.2 Indeksoperasjoner og spørrespråk i ustrukturerte dokumenter

Tradisjonelle spørrespråk baserer seg primært på spørringer bestående av enkeltord. Det eksisterer likevel mange operatorer som er mulig å nytte for å forbedre søk. Blant operatorer som i så fall må støttes av indekseringsmetodene som brukes er [12]:

- Ordforekomst - Indikerer hvor ofte et spesifikt søkeord eksisterer inne i ett dokument. Poenget med dette er det antas at et ord som eksisterer oftere enn et annet er en bedre indikator på hva et dokument handler om enn et annet
- Boolsk søk - muliggjør komplekse relasjoner imellom ord/fraseforekomster. For eksempel ”foo AND bar” eller ”Trondheim OR Oslo”. Treff lokaliseres da basert på om de stemmer med ordene gitt i søket i forhold til sammenhengen og eventuelle parenteser.
- Regulære uttrykk - Søk kan gjøres ved bruk av komplekse utrykk som spørringer. Primært brukt for å finne treff på høgt strukturerte data enn for identifisering av konseptuelt innhold.
- Frasesøk - Søking ved bruk av flere ord, del av en setning. Kan også gjøres til dels ved bruk av de andre metodene nevnt.
- Avstandsoperatorer - Muliggjør søk ved å angi spørringer der avstanden mellom deler av spørringen er angitt. Dvs. ting som hvor mange ord det skal være imellom to deler av en spørring.
- Ordstemming – Blir ofte brukt fremfor de aktuelle ordene. Fjerner sluttstavingen av et ord og gjør at søk etter ”look” også får treff på ”looking

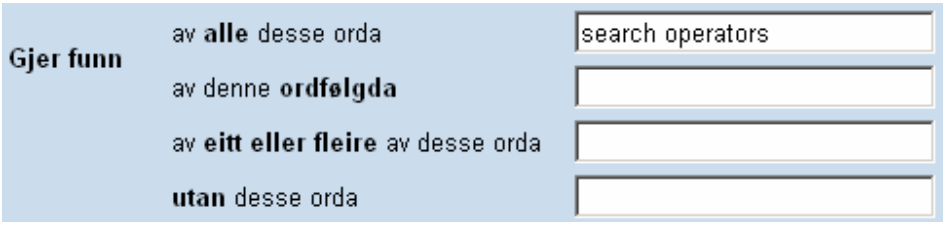
og ”looks”. Spesielt nyttig dersom det er en antatt relevans mellom ordene i de indekserte dokumentene. Ofte bedre enn utføring av ett søk på hvert ord.

- Konseptuelle søk - Finner dokumenter som dekker lignende temaer ved å gjenkjenne ord med tilsvarende mening. Bruker ofte tesauruser for å oppdage sammenhenger imellom termer.
- Soundex søk – Muliggjør at det kan eksistere feil i spørringer i form av stavefeil, spesielt i engelsk. Fremfor å bruke ord slik de er stavet blir de brukt basert på hvordan de uttales, og deretter sammenlignet med ord ifra andre konverterte dokumenter.

2.2.3 Søk i tradisjonelle søkemotorer

Tradisjonelle søkemotorer som Google [13] og andre har begrensede muligheter til avanserte søk basert på ulike datatyper og søker derfor over hele dokumentsamlingen. Nøyaktig hvordan Google gjør dette er ikke helt kjent siden de verner om sine metoder. Spørrespråket gir likevel et visst innsikt i hva som ligger bakenfor grensesnittet.

Vanlige søkemotorer som søker over ustrukturert tekst inneholder normalt ikke muligheter for avanserte spørringer som går på strukturen i dokumentene, noe Google sin ”avanserte søk” side viser under. De muliggjør bare søk basert på ord uten å spesifisere hvor eller i hvilken kontekst ordet ønskes funnet.



Figur 3 - Søkegrensesnitt for tradisjonelle søkemotorer

Søkeresultat fra Google blir presentert som en liste over dokumenter som stemmer med en gitt spørring der noen setninger fra dokumentet sammen med søketermer blir uthevet og presentert for brukeren. Brukeren kan velge dokumenter for å se hele innholdet, men må deretter lete seg frem i dokumentet for å finne hva som gjorde at Google fann dokumentet. Dette er i tråd med de fleste søkemotorer som ikke returnerer faktisk relevant innhold men informasjon om hvor innholdet eksisterer.

2.3 Informasjonsgjenfinning i semistrukturerte dokumenter

Som nevnt tidligere har fokuset innenfor informasjonsgjenfinning de senere årene dreiet ifra søk i ustrukturerte dokumenter til søk i semistrukturerte dokumentformater som XML. Noen av grunnen til dette er at lagringskapasitet og hastigheten på datamaskiner i dag muliggjør å inkludere mer informasjon uten at ytelsen blir svekket, samtidig som den store økningen i mengden informasjon krever nye metoder for å kunne øke

søkepresisjonen. Det er også etter hvert blitt et ønske fra mange at dokumenter skal kunne leses og skrives til uten å være låst til en spesifikk applikasjon.

De første systemene som har benyttet seg av søk i strukturerte dokumenter er digitale biblioteker der heterogene samlinger av XML dokumenter i form av forskjellige metadataformat er vanlig. Vanligvis har disse systemene krevd at bruker først velger hvilken samling det er ønskelig å søke i før et søk kan gjennomføres.

> *Buckland and Plaunt[..] have pointed out, searching for recorded knowledge in a distributed digital library environment involves three types of selection:*
> *1. Selecting which library (repository) to look in;*
> *2. Selecting which document(s) within a library to look at; and*
> *3. Selecting fragments of data (text, numericdata, images) from within a document.[14]*

Med informasjonsgjenfinning i semistrukturerte dokumenter er det ønskelig at et søk skal gjøre de valg som er nødvendige og vise de fragmenter som er relevante for søket. Dette gjelder spesielt for heterogene dokumentsamlinger der det ofte er ønskelig å kunne utføre søk uten å måtte velge hvilket bibliotek og samling det skal utføres i.

De som arbeider med informasjonsgjenfinning i XML har i stor grad kunnet deles opp i to leirer; de som har bakgrunn ifra databaser og de som har bakgrunn innenfor informasjonsarbeid. Som tidligere nevnt fokuserer datagjenfinning på eksakte treff mens informasjonsgjenfinning ser på relevante treff. Overført til XML har databaseteoriene blitt nyttet mot strukturen i dokumentene og ikke for implementasjon av sortering etter relevans. Databaser har også ofte blitt brukt som basis for systemene.

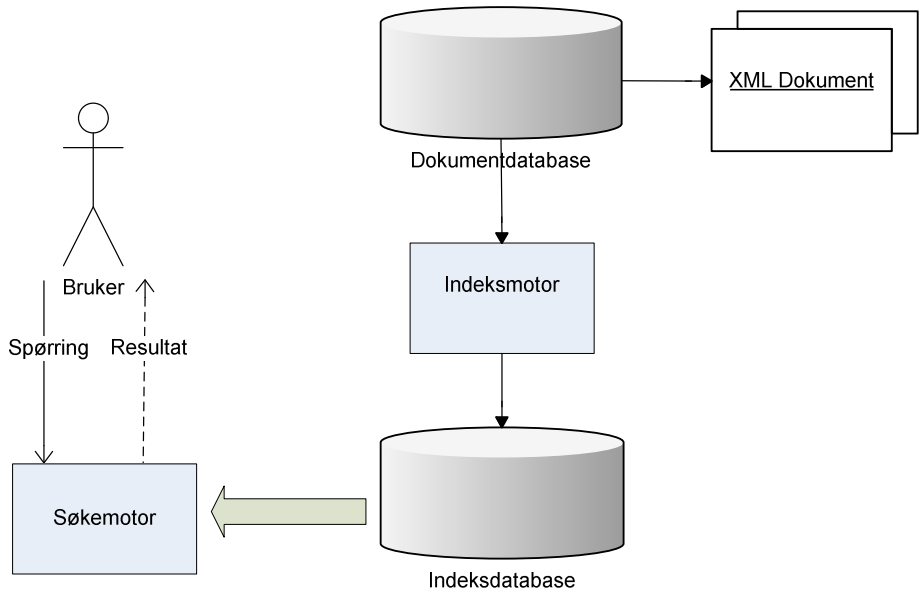
Informasjonsteoriene har blitt brukt til å sortere etter relevans men ikke i så stor grad se på struktur. De senere årene har denne forskjellen forsvunnet etter hvert som ny forskning har foreslått metoder som forener de. Flere systemer er i dag en hybrid av løsninger ifra databaseverdenen og informasjonsverdenen*, "The best of two worlds"* [15].

De løsninger som blir brukt innenfor informasjonsgjenfinning i XML kan deles opp i to kategorier; løsninger som bruker tradisjonelle metoder og nye metoder spesielt rettet mot XML. De tradisjonelle metodene bruker teknikker ifra tradisjonell informasjons-gjenfinning overført til bruk på semistrukturerte dokumenter. Andre forsker på nye metoder spesielt for bruk mot XML og hierarkiske dokumenter generelt ved bruk av XML standarder som XPath og XQuery.

Det eksisterer ikke noen standardisert måte selve søkene i XML utføres på, til tross for at standardiserte dokumentformater brukes.

Forskning på informasjonsgjenfinning i XML dokumenter foregår blant annet innenfor INEX initiativet [16] som stiller med ressursene som er nødvendige for å kunne gjennomføre forsøk i en realistisk setting. Men selv innenfor INEX er det variasjoner i hvordan korrekte resultater blir produsert. De overordnede metodene som brukes er likevel felles for mange systemer.

En generell modell for informasjonsgjenfinningssystemer beregnet på semistrukturerte dokumenter gjennomgåes nedenfor.



Figur 4 - Arkitektur for informasjonsgjenfinning i semistrukturerte dokumenter

XML Dokumenter

Alle data som skal brukes av systemet leses ifra en lokasjon og blir tolket som XML dokumenter av systemet der de valideres og lagres i en dokumentbase sammen med eventuelle skjema. Dokumentene kan komme ifra lokale lager, fra Internett eller bli tilgjengeliggjort igjennom protokoller som OAI-PMH [17]. Tilgang til original-dokumentene er vanligvis ikke nødvendig etter at de er lagret i den lokale databasen.

Dokumentbase

Alle dokumenter blir, dersom de er korrekte XML dokumenter og støtter eventuelle krav ifra skjema, lagret i en base som muliggjør å hente ut elementer på en effektiv måte uten å måtte lese hele XML dokumenter inn i minnet. Dette gjøres ved å lage indekser over eksisterende stier for hvert dokument som deretter kan brukes for raske oppslag. Utsnitt av XML dokumenter kan da hentes ut direkte.

Indeksmotor

Denne leser XML dokumenter ifra dokumentbasen og gjør eventuelle prosesseringer over de data som eksisterer for tilpassning slik at de kan lagres i indeksene. Vanligvis blir det laget en invertert indeks over alle ordforekomster sammen med deres plassering i dokumentene, og en indeks over strukturen slik at denne kan brukes ved søking. Nødvendige utrekninger for å støtte rangering blir også gjort, for eksempel basert på vektormodellen.

Indeksdatabase

Dette er en database i en eller annen form som inneholder de indekser og andre data som er nødvendige for å kunne besvare en brukers søk. Denne inneholder vanligvis ikke original versjon av dokumentene. De kan derfor ikke presenteres på original form ut i fra

innholdet i indeksen. Noen systemer muliggjør å legge fulltekstdokumenter inn i en indeks for senere tilgang, men dette kan øke indeksstørrelsen og gjøre søking tregere.

Spørremotor
Behandler spørringer og omformer de for å gjøre søk effektivt mot de indeksene som eksisterer. Spørringer blir mottatt ifra brukerne, spørringen optimaliseres og utføres mot eksisterende indekser før resultat blir aggregert, sammensatt, satt inn i grensesnittet og returnert til bruker. Fulltekstdata for hvert treff blir hentet ifra dokumentbasen basert på hvert treff sitt opphav. Lagring av ofte brukt spørringer og vanlige resultatsett kan forekomme ved bruk av hurtiglager.

Bruker
Genererer spørringer og evaluerer resultatene som blir mottatt. Resultatene ifra et søk kan bli brukt til å formulere nye spørringer. Igjennom tilbakemelding ifra en bruker på et resultatsett kan et system få informasjon som muliggjør å forbedre og gjenta søk.

2.3.1 eXtensible Markup Language

XML [18] står for 'E**X**tensible **M**arkup **L**anguage' og er et oppmerkingsspråk likt HTML [19] men med til dels store forskjeller. XML er et simpelt og fleksibelt format derivert ifra SGML (ISO 8879) og er primært brukt for å overføre og lagre data på en måte som støtter interoperabilitet.

Med HTML er målet å ta vare på og beskrive hvordan innholdet skal presenteres, mens XML er designet for å beskrive data og har andre standarder som tar seg av presentasjonen. Dette betyr at med XML er ikke elementene definerte slik som i HTML, men kan defineres av bruker ved behov.

XML er i dag den standarden som primært blir brukt for beskrivelse av semistrukturerte data. Semistrukturerte data er ikke like strukturerte som data ifra databaser. Databaser har en klart definert struktur lagret som poster eller tupler, mens XML ligger imellom det og ustrukturerte data. Ustrukturert data kan være et normalt tekstdokument og semistrukturerte data inneholder derfor data etter en forhåndsdefinert struktur men også ustrukturerte komponenter etter et forhåndsdefinert mønster [20].

```
<?XML version="1.0" encoding="ISO-8859-1"?>
<note id="123745">
<to>Trond</to>
<from>Trondheim Kiteklubb</from>
<heading>DET BLÅSER PÅ ØYSAND, 8m/s</heading>
<body>Denne meldingen er sendt ifra Trondheim Kiteklubb til Trond.</body>
```

Tabell 3 - Eksempel på et XML basert semistrukturert dokument

Eksempelet over viser hvordan et XML dokument kan se ut for en melding lagret i XML format. Dette dokumentet sier ikke noe om hvordan innholdet skal tolkes direkte, men gir et format som muliggjør at to eller flere maskiner kan forstå hvordan de data som blir

sendt imellom dem er strukturert. Noe som også gjelder selv om det brukes forskjellige programmer til denne prosessen så lenge standardene støttes.

XML eksempelet over kan deles inn i tre deler. Først er det en deklarasjon som beskriver formatet til det som følger sammen med hvilket karaktersettet som er brukt. <note> er her rotelementet og kan inneholde flere elementer i et hierarki slik det gjør i eksempelet. Innholdet til elementer kan bestå av forskjellige typer data, eller de kan bestå av andre elementer. Elementer kan også være tomme. Hvert element kan også inneholde attributter slik som ”note” sitt id-attributt. Felles for alle data i dette eksempelet er at de er definert som tekst ettersom det ikke er noe informasjon knyttet til eksempelet som angir noen annet.

XML dokumenter kan også inneholde entiteter som kan brukes til å forhåndsdefinere verdier for bruk i dokumentene, tilsvarende bruk av konstanter i programmeringsspråk. En entitet kan defineres og senere refereres til ved å bruke ”&entitetsnavn”. For eksempel kan en entitet ved navn ”eg” og verdien ”Trond” defineres. Under tolkning av teksten ”⪚ er student” vil da teksten ”Trond er student” vises.

Elementer i XML kan tilhøre flere navneområder slik at flere elementer kan ha samme navn men ha forskjellig mening. Dette gjør at generiske elementnavn som ”navn” har forskjellig betydning ut i fra om elementet sitt navneområde er ”by:navn” eller ”båt:navn”.

Presentasjon av XML kan gjøres ved bruk av XSL [21] og CSS [22] for å transformere XML dokumenter til hvilken som helst ønsket presentasjonsform uten å gjøre noen endringer i datagrunnlaget, noe som fremhever XML som format primært for kommunikasjon av data.

For at et XML dokument skal være et korrekt XML dokument må det følge noen regler for syntaksen definert for standarden og dokumentet blir da ”well-formed” ut i fra W3C sin definisjon.

Det eksisterer flere forskjellige måter å definere strukturen i et XML dokument, men primært er det to standarder som er anbefalte av W3C. Et skjema definerer hva som er lovlig innhold for et XML dokument, og kan ta for seg både strukturelle begrensninger og datainnhold. Skjema i XML kan sammenlignes med tabeller i databaser som beskriver hva som er lovlig innhold, men uten å definere så spesifikt som i databaser hva dette kan være. Det eksisterer to forskjellige skjemaspråk anbefalt av W3C, disse to er DTD, ”Document Type Definition”, og XML Schema. Dersom dokumentet stemmer overens med et skjema, enten det er DTD eller XML Schema, er det et ”valid-XML” dokument.

XML Skjema

Document Type Definition

DTD standarden brukes for å definere regler og enheter for ett eller flere dokumenter. Ved å nytte en DTD defineres strukturen som skal brukes i et XML dokument, de dokumenter som ikke følger skjemaet kan risikere å ikke bli akseptert. En DTD kan enten være en del av dokumentet selv, eller den kan være lagret i en egen fil som det refereres til fra dokumentet. Et dokument må ikke ha en DTD, men det kan også ha flere hvis det er ønskelig. Generelt er det lurt å bruke en DTD dersom det skal opprettes mange dokumenter med samme oppsett for å sikre strukturell likhet imellom dokumentene [23].

XML tilbyr begrenset støtte for beskrivelse av datatyper igjennom bruk av DTD. DTD muliggjør bare bruk av #PCDATA datatypen for tekst og #CDATA for andre uspesifiserte datatyper som ikke skal tolkes. I tillegg er det mulig å spesifisere forskjellige attributtverdier, men de er ikke fleksible nok til å kunne støtte validering av innhold mot en datatype for å sikre at feltet faktisk inneholder hva det skal inneholde.

XML Schema

Ettersom XML standarden i seg selv bare kjenner til tekst som datatype i tillegg til karakterdata er det begrenset hvilke datatyper som kan beskrives ved bruk av kun XML. Som svar på dette vart XML Schema utformet.

XML Schema standarden utvider antallet datatyper til å gjelde flere av de brukt i databaser og muliggjør i tillegg definering av egne typer [24]. Støtte for flere datatyper gir XML Schema noen fordeler fremfor DTD og validering av innholdet er en av disse. I tillegg er det:

- Enklere å beskrive lovlig dokumentinnhold
- Enklere å jobbe med data ifra en database
- Enklere å definere restriksjoner på data
- Enklere å definere dataformat
- Enklere å konvertere data mellom forskjellige datatyper

Om et XML dokument tilhører et XML Schema må dette identifiseres i dokumentet slik at en eventuell XML Schema validerer kan finne skjemaet. Dette kan gjøres ved å spesifisere et XML Schema for hvert navneområde eller et XML Schema for hele dokumentet om ønskelig. Bruk av flere XML Schema i ett dokument ut i fra navneområde er også mulig men begrenser til at bare ett skjema per område kan være definert. En deklarasjon av XML Schema informasjon er likevel ikke påkrevd å bli verifisert av prosessoren som leser XML filen. Prosessoren er fri til å bruke de definerte skjema, bruke andre skjema, eller ikke noe skjema i det hele tatt [25].

Et eksempel på et enkelt XML Schema er angitt under, her er elementet ”sizes” av datatypen desimal og hver enhet i listen er et desimaltall:

```
<simpleType name='sizes'>
  <list itemType='decimal'/>
</simpleType>
<cerealSizes xsi:type='sizes'> 8 10.5 12 </cerealSizes>
```

Tabell 4 - XML Schema Simpletype

XML Schema muliggjør definering av applikasjonsspesifikk informasjon for et skjema igjennom bruk av en xs:annotation beskrivelse for hvert element. Dette kan selv inneholde XML og dermed gi informasjon om et XML Schema til en applikasjon. Primært brukes det til å formidle data til en applikasjon igjennom en xs:appinfo deklarasjon mens et ”xs:documentation” element eksisterer for å formidle informasjon til leser og utvikler av skjemaet.

```
<xs:element name="creator" type="SimpleLiteral">
<xs:annotation>
<xs:appinfo XMLns:iso11179-3="urn:ISO-IEC:11179-3:1994">
 <iso11179-3:Name>Creator</iso11179-3:Name>
 <iso11179-3:Definition>An entity primarily responsible for making the content
of the resource.</iso11179-3:Definition>
 </xs:appinfo>
 </xs:annotation>
 </xs:element>
```

Tabell 5 - XML Schema xs:annotation eksempel

Bruken av xs:annotation muliggjør dermed å gi spesifikk informasjon til applikasjoner med behov for informasjon utover hva som kan angis igjennom direkte bruk av klossene standarden tilbyr. En standard som benytter seg av dette er Schematron [26]. Schematron tilbyr regler for definering av ekstra krav igjennom xs:annotation.

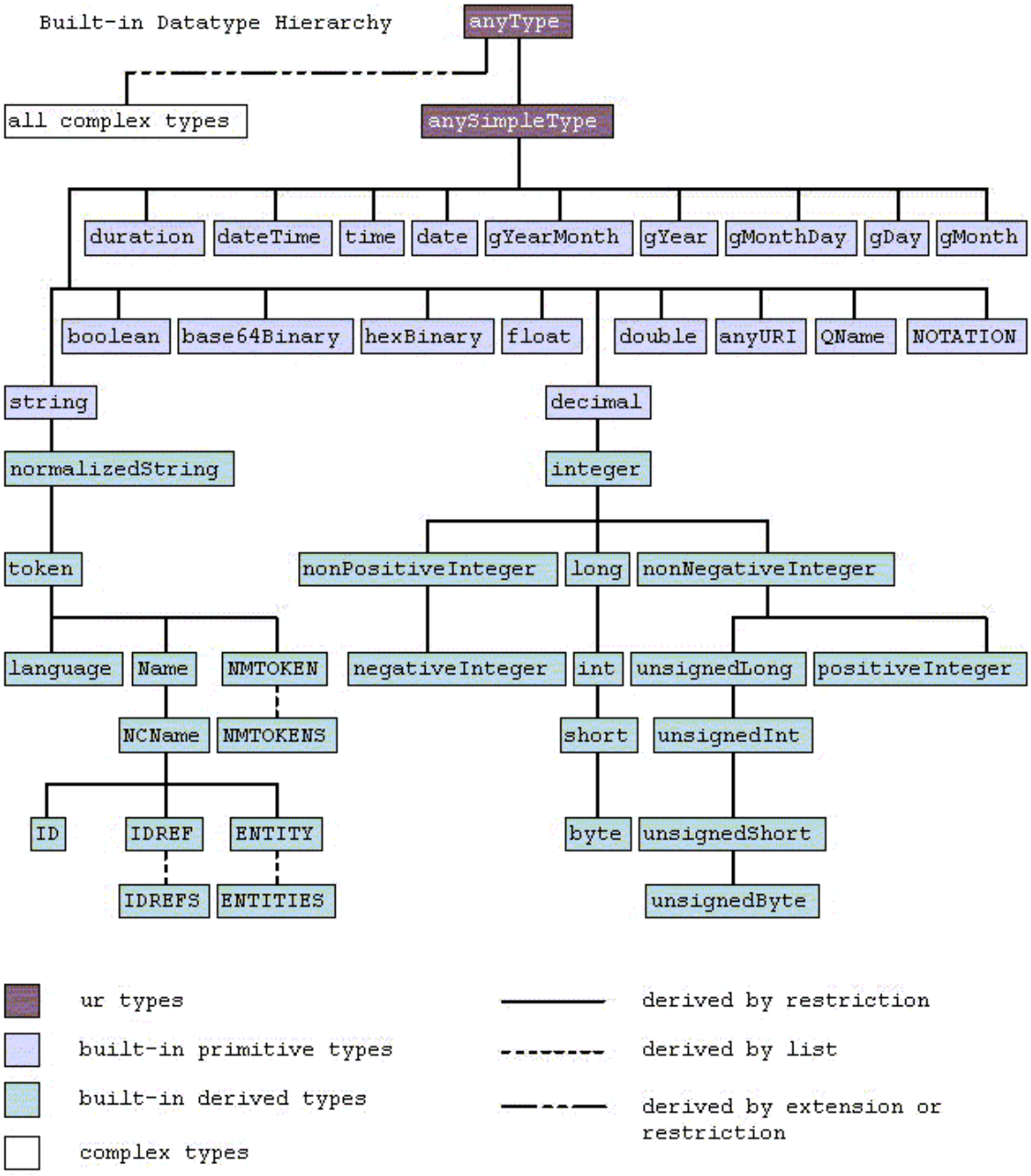
W3 Consortium [27] som har utarbeidet XML Schema standarden har skilt datatypene som er tilgjengelig i XML Schema imellom simple, deriverte og komplekse typer. De simple består av en primitiv datatype, deriverte er basert på andre datatyper mens komplekse typer kan bestå av en miks av flere primitive datatyper. Som hierarkiet nedenfor viser, støtter XML Schema de mest nødvendige primitive datatypene.

> *[Definition:] An XML Schema simple type d is characterised by a value space, V(d), which is a non-empty set, a lexical space, L(d), which is a non-empty set of Unicode strings, and a set of facets, F(d), each of which characterizes a value space along independent axes or dimensions.* [28]

I tillegg til forhåndsdefinerte datatyper tilbyr også XML Schema muligheten til å definere egne datatyper:

> *[..] allow creation of user-defined datatypes, such as datatypes that are derived from existing datatypes and which may constrain certain of its properties (e.g., range, precision, length, format).* [25]

Under vises hierarkiet [29] over datatyper i XML Schema slik de er delt opp ifølge standarden.



Figur 5 - XML Schema datatyper

Metadatadokumenter uttrykt i XML

Metadatadokumenter er en av de dokumenttypene der XML er mest brukt for strukturering av innhold og som det for mange formater eksisterer regler for bruk sammen med XML. Dette viser at XML er spesielt fordelaktig for strukturering av denne type data. Eksempel på andre dokumentformater er fulltekstsamlinger som INEX og ellers alt annet som det er ønskelig å definere et format for grunnet XML sin fleksibilitet.

Forenklet er metadata en beskrivelse av data, populært ”data om data”. Metadata er ikke en erstatning for et dokument, men derimot et supplement. For eksempel kan hva som er tittel og hvem som er forfatter av et dokument struktureres slik at alle dokument angitt i samme metadataformat kan forstås på en felles måte av en datamaskin.

Metadata er altså strukturerte data om et objekt som gjør det lettere for en datamaskin å gjenfinne objektet basert på tilgjengelige data i deklarasjonen. Grunnen er at det ikke er behov for å lete i selve objektet, men den kan finnes på en bestemt plass i et bestemt format. Dette gjør det mye raskere å finne dokumenter basert på visse kriterier enn om hele objekter måtte gjennomsøkes. Det kan dermed med fordel brukes metadata på objekter som ikke inneholder tekst siden objektet dermed kan gjenfinnes uten å måtte søke igjennom store mengder binære data.

Siden de fleste metadataformat på en eller annen måte er strukturerte, er XML et ideelt format for oppbevaring og beskrivelse av metadata. Bruk av XML forenkler også konverteringer mellom de forskjellige formatene. En gjennomgang av noen metadatastandarder som bruker XML gjennomgås under ut i fra hva de har i relevans til oppgaven.

Dubin Core Metadata Initiative – Dublin Core

Dublin Core (DC) [30] ble utviklet for å være et generelt metadataformat og er ment å kunne brukes til beskrive den viktigste informasjonen om et dokument, enten det er et bilde eller et tekstdokument. Målet til DCMI (Dublin Core Metadata Initiative) er å legge til rette for utvikling av metadatastandarder som støtter interoperabilitet og muliggjør mer intelligente informasjonsgjenfinningssystemer. Formatet har sitt utspring fra en konferanse i Dublin, Ohio i 1995, derav navnet. Dublin Core er spesielt mye brukt av OAI, ”Open Access Initiative” [17], som igjennom OAI-PMH bruker Dublin Core til å beskrive metadata for annoterte objekter.

Fokuset under utvikling av standarden var at det skulle være enkelt å fokusere på selve elementene, ikke på syntaks og notasjoner. Videre tar det hensyn til utvidbarhet, dette ble utelatt i starten men har blitt innført senere. Syntaks og notasjoner som brukes er bygd på HTML og XML slik at det tillates metadata i primærdokumentet. I HTML for eksempel brukes meta-elementer til å uttrykke DC. DC består ellers av 16 hovedelementer for å beskrive et objekt og ”DC Qualified” har igjen 28 underelementer som kan brukes til å beskrive et objekt ytterligere. DC definerer ikke eksplisitt hvilke datatyper som må brukes for de forskjellige feltene i standarden, men gir anbefalinger til innholdet.

DC er et generelt metadataformat og kan beskrive hvilken som helst type ressurs enten det er fysisk eller elektronisk. Metadataformatet kan brukes av biblioteker, utdanningsinstitusjoner, statlige institusjoner, forfattere av nettsider, bedrifter med store kunnskapsstyringssystemer og generelt sett der et behov for enkle format eksisterer. Grunnet et lite antall støttede elementer er likevel bruken av DC av begrenset nytte dersom spesielle behov skulle eksistere.

```
<?XML version="1.0"?>
<metadata
  XMLns="http://example.org/myapp/"
  XMLns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://example.org/myapp/
http://example.org/myapp/schema.xsd"
  XMLns:dc="http://purl.org/dc/elements/1.1/">
  <dc:title>UKOLN</dc:title>
  <dc:description>
    UKOLN is a national focus of expertise in digital information management. It
provides policy, research and awareness services to the UK library, information
and cultural heritage communities. UKOLN is based at the University of Bath.
  </dc:description>
  <dc:publisher>
    UKOLN, University of Bath
  </dc:publisher>
  <dc:identifier>
    http://www.ukoln.ac.uk/
  </dc:identifier>
</metadata>
```

Tabell 6 - Dublin Core metadataeksempel

"Metadata Object Description Schema" (MODS)

MODS [31] er en forkortelse for 'Metadata Object Description Schema' og opphaver fra 'Library of Congress Network Development' og 'MARC Standards Office' [32].

MODS er et nytt metadataformat som var ferdig i 2001, men som i stor grad bygger på MARC 21 og er ment brukt spesielt av bibliotek. Det er i likhet med MARC basert på bruken av tagger, og poster i MARC-formatet kan konverteres til MODS men ikke omvendt. Formatet er spesielt konstruert for bruk sammen med XML og XML Schema.

I motsetning til MARC21 bruker MODS språkbaserte tagger fremfor tallbaserte, noe som gjør forståelsen av en MODS metadatapost lettere. MODS legger seg midt mellom Dublin Core, som er kritisert for å ha litt for få kategoriinndelinger, og den større og langt mer kompliserte MARC standarden.

```
 <xs:group name="modsGroup"><xs:choice>
  <xs:element name="titleInfo" type="titleInfoType" />
  <xs:element name="name" type="nameType" />
  <xs:element name="typeOfResource" type="typeOfResourceType" />
  <xs:element name="genre" type="genreType" />
  <xs:element name="originInfo" type="originInfoType" />
  <xs:element name="language" type="languageType" />
  <xs:element name="physicalDescription" type="physicalDescriptionType" />
  <xs:element name="abstract" type="abstractType" />
  </xs:choice></xs:group>
```

Tabell 7 - Utdrag fra MODS XML Schema

Som utdraget over viser er MODS et format som er beregnet for bruk mot metadata og inneholder derfor ikke elementer som støtter angivelse av innholdet i et fulltekst-dokument. I MODS er hovedelementene generelle mens roller og typeinformasjon spesifiserer mer nøyaktig meningen. En datatype i MODS er dermed en kombinasjon av elementnavn, type og rolle.

```
<name type="personal">
 <namePart type="given">Ash</namePart>
 <namePart type="family">Amin</namePart>
<role><roleTerm type="text">author</roleTerm> </role>
 </name>
```

Tabell 8 - MODS rolleeksempel

2.3.2 Heterogene dokumenter

En av de store forskjellene mellom informasjonsgjenfinning i semistrukturerte og ustrukturerte dokumenter er variasjonen i struktur som et strukturert dokument ut i fra XML standarden kan ha.

Homogene dokumenter er dokumenter som har et felles skjema og som det dermed kan antas å ha likt innhold for like elementnavn. Slike samlinger eksisterer primært innenfor organisasjoner som har mye lik informasjon eller likt strukturerte dokumenter, som tilfellet er med samlinger av metadatadokumenter basert på Dublin Core. Ved å kjenne denne strukturen er det også lettere å lage spørringer som er optimalisert for en samling. På heterogene dokumenter kan ikke en slik struktur benyttes siden det må antas at alle dokumenter med forskjellig skjema har forskjellig struktur og oppbygging. Elementer med samme navn trenger heller ikke nødvendigvis bety det samme. Om det er ønskelig å skjule dette for en søker må det manuelle eller automatiske metoder til for å vedlikeholde et skjema felles for alle indekserte dokumenter.

De mest brukte metadataformatene støtter bruk av XML som format for lagring av informasjonsinnholdet. Dette gir mulighet til å kunne søke i store mengder dokumenter som i seg selv er lagret i forskjellige formater men som igjennom metadata likevel muliggjør gjenfinning basert på struktur. Indeksering av XML baserte metadata er mindre komplisert enn hele dokumenter ettersom regler for bruk av de forskjellige feltene eksisterer.

2.3.3 Persistent lagring av XML

Semistrukturerte dokumenter inneholder en større mengde data som må tolkes for å kunne forstå dokumentene korrekt enn vanlige tekstdokumenter. Lesing av slike dokumenter er derfor mer ressurskrevende enn ved lesing av ustrukturerte dokumenter og har ført til nye lagringsløsninger.

Effektiv tilgang til innholdet av XML dokumenter krever at de gjøres tilgjengelig på en mer effektiv måte enn et vanlig filsystem muliggjør. Om en XML fil er lagret på disk må hele XML filen vanligvis leses inn i minnet for å hente data ifra en spesifikk plass i

filen. Informasjonsgjenfinning i XML dokumenter krever at spesifikke elementer i dokumentene til forskjellige tider kan hentes ut; innlesing og behandling av hele filer er da ikke effektivt nok. En annen måte å lagre XML dokumenter på er å bruke en native XML database. En native XML database er en database som [33]:

- Definerer en (logisk) modell for et XML dokument og lagrer og henter dokumenter basert på modellen. Modellen må minimum inkludere støtte for elementer, attributter, #PCDATA og dokumentrekkefølge. Eksempler på slike modeller er XPath datamodellen, XML Infoset, og modellene gitt ut i fra DOM og hendinger i SAX 1.0.
- Har et XML dokument som sin fundamentale enhet for (logisk) lagring, akkurat som en database har en rad i en tabell som sin fundamentale enhet for (logisk) lagring.
- Det er ikke krav om å ha noen spesiell underliggende fysisk lagringsmodell. For eksempel kan en native XML database bli bygd oppå en eksisterende relasjons, hierarkisk eller objektorientert database eller andre proprietære lagringsformat som indekserte, komprimerte filer.

XML databaser tilbyr i motsetning til et vanlig filsystem å gjenfinne deler av et dokument uten å måtte traversere hele dokumentet etter de ønskelige delene. En XML database tilbyr også søkemuligheter basert på tidligere nevnte databaseteorier, og dermed ikke sortering etter relevans eller støtte for forskjellige datatyper. En XML database er ekvivalent i forhold til en relasjonsdatabase i form av at mange av de samme teknikkene og mulighetene eksisterer, men også mange av de samme ankepunktene.

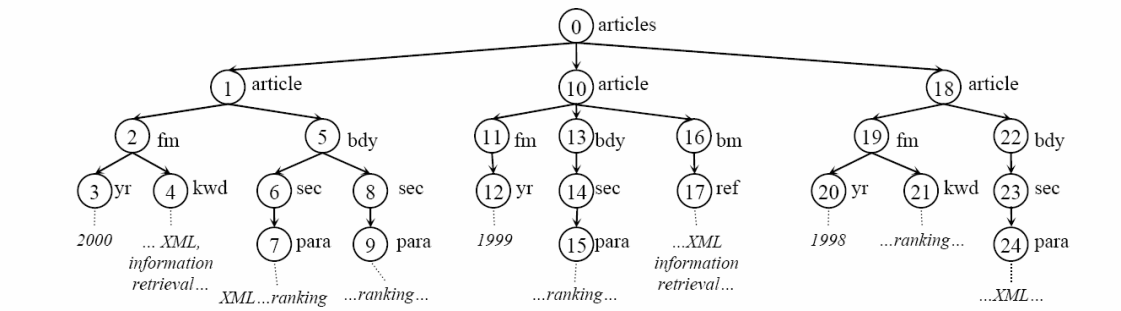
Relasjonsdatabaser har støtte for søk og uthenting av data ifra en struktur i form av tabeller, noe også XML databaser har mulighet for igjennom spørringer mot enkeltelementer eller hele samlinger. Forskjellen er at ikke alle XML databaser krever at et skjema, enten det er DTD eller XML Schema, er knyttet til en samling. XML databaser skiller seg her ifra den faste strukturen en tabell i en database elles har.

En relasjonsdatabase bruker SQL eller lignende spørrespråk for uthenting av data ifra databasen mens en XML database bruker XPath og XQuery til å utføre tilsvarende spørringer. XPath er ikke beregnet på å bli brukt som XMLDB spørrespråk og mangler mange av de mulighetene SQL tilbyr i SQL-databaser (gruppering, sortering, cross document joins, og støtte for datatyper). Den er også begrenset til operasjoner på primitive datatyper som strenger, nummer og boolske verdier. XQuery har grunnet de mange begrensningene som XPath har tatt over i stor grad, men XPath er likevel i bruk som en del av XQuery. De to standardene sees nærmere på senere i rapporten.

2.3.4 Indeksering av semistrukturerte dokumenter

Indeksering av XML stiller nye utfordringer men også nye muligheter for hvordan indekseringen kan gjennomføres enn hva tilfellet var med ustrukturerte tekstfiler. I tillegg til tekstlig innhold er det stier som unikt identifiserer hvor i treet teksten befinner seg og som kan representeres i form av et trehierarki som vist under [2]. XML sammen med XML Schema muliggjør også å unikt beskrive hvilken datatype som et element

inneholder. Dette gjør at data av forskjellige typer kan identifiseres og indekseres på egne måter tilpasset hver enkelt datatyper sammen med eventuelt andre tilgjengelige data som gir semantikk til innholdet.



Figur 6 - Eksempel på strukturhierarki i XML

Indeksering kan deles opp i tre metoder ut i fra hvordan dokumentet tolkes; som en flat fil, semistrukturert, eller strukturert indeksering. Semistrukturert indeksering kan igjen deles opp i feltbasert, segmentbasert og trebasert [34]. De forskjellige metodene definerer primært hvordan strukturen i et XML dokument skal behandles og brukes under indeksering.

- Flat fil: Indeksering uten benyttelse av struktur, tilsvarende invertert indeks for ustrukturerte dokumenter.
- Semistrukturert: Bruker bare deler av den strukturelle informasjonen som er tilgjengelig ved indeksering. Sette kan være trær, segment eller feltbasert struktur.
- Strukturert: Strukturert indeksering brukes på informasjon som inneholder en fast definert struktur definert i et skjema eller i form av en tabell i en database. Kan derfor sees på som å ha mye til felles med databaser.

Et komplett indekseringssystem for XML bør ta i betraktning strukturelle forhold og tilby indekser som kan brukes til å finne de mest relevante elementene, og ikke bare returnere treffene direkte. Eksempel på slike krav er aggregering av felt der det må bestemmes om en undernode skal vises fremfor foreldrenoden, altså hvor spesifikt et treff skal være. Dette krever indekser der det kan navigeres i strukturen på en effektiv måte.

For å kunne indeksere XML er det nødvendig å definere hvilke deler av et XML dokument som er ønskelig å indeksere og hvordan de forskjellige delene skal indekseres. Automatisk konfigurering av dette er å foretrekke og kan gjøres på homogene samlinger der alle dokumenter er basert på samme skjema. Mer problematisk blir det når heterogene samlinger med de samme elementnavnene i forskjellige kontekster skal indekseres. Det må da gjennomføres kompliserte databehandlinger basert på innhold for å kunne lage et felles skjema, hvordan kan et felt med navn ”tittel” i ett skjema automatisk kobles sammen med ”title” i et annet? Dette kompliseres også ved at samlinger som INEX ikke har skjema som spesifiserer datatyper for hvert element og attributt. Det er da nødvendig å gjennomgå dokumentene for å finne karakteristikker for hvert element som kan brukes, og kan føre til feil i datagrunnlaget [2]. Ved manuell konfigurering er det enklere ettersom administrator sammen med støtte fra automatiske systemer bestemmer hvordan

og hvilke data som skal indekseres. Dette er likevel ueffektivt for bruk mot mange heterogene dokumentsamlinger.

Hva som skal indekseres av et dokument baseres på om innholdet av elementene kan være relevante for en bruker å søke i og kan variere med konteksten. Søk i ID nummer på et element er ikke alltid et interessant søkeobjekt, om det da ikke er et personnummer eller lignende. Måten de indekserte data behandles og lagres bestemmer også hvor raskt et søk kan gjennomføres og innvirker på gjenhentingsgrad og søkepresisjon. Dette gjør korrekt indeksering nødvendig for å kunne oppnå best mulig søkeresultat.

De data som skal indekseres kan også være av forskjellige datatyper og stå i kontekster der forskjellige metoder må brukes ved indeksering. Aktuelle metoder ved indeksering er fjerning av stoppord, stemming, støtte for frasesøk, trunkering og datostøtte. Metodene som brukes i hvert enkelt tilfelle bestemmes av innholdet i elementene til et XML tre som er ønskelig å indeksere. For eksempel er det kanskje ikke ønskelig å fjerne store bokstaver eller bruke stemming på navn ettersom det da kan være ønskelig med eksakte treff på et søk. Fjerning av stoppord kan også i noen tilfeller skape problemer dersom fraser som ”to be or not to be” ønskes søkt etter. Ofte er det heller ikke ønskelig at eksempelvis +- tegn blir fjernet siden det vil kunne skape problem for søk etter matematiske formler.

Som eksempel på definering av hvordan innhold skal indekseres som forskjellige typer kan Z39.50 og BIB-1 sine strukturattributt brukes der de forskjellige attributtene inneholder krav til bruk.

Z39.50 BIB-1 Strukturattributt	**Bruk**
Frase	Rekkefølge må tas vare på, ellers fritt
Ord	Eksakt, om ikke direkte spesifisert
Ordliste	Ingen rekkefølge, kan behandles

Tabell 9 - Z39.50 indeksering av datatyper

Indekstyper

De forskjellige indekstypene som er nødvendig for å kunne indeksere XML er primært bestemt ut i fra hvilke typer data det er ønskelig å kunne søke etter. Ikke alle datatyper er interessante å indeksere for seg selv, noe som bestemmes ut i fra hvilke behov systemet har. Siden hver indekstype krever egne strukturer for hvordan data lagres og gjenhentes mest mulig effektivt er det nødvendig å begrense antallet for ikke å få et for komplisert system.

For tekst er den vanligste indekstypen invertert indeks som lagrer hvert ord sammen med lokasjonen til hver forekomst og dermed muliggjør søking etter ordforekomster. Datoer krever andre indekser som effektivt støtter søk etter treff imellom to datoer. XML krever også strukturer som tar for seg hvordan dokumentene er oppbygd dersom systemet skal kunne benytte seg av den strukturelle oppbyggingen av dokumentene for å bedre søkeresultatene. En analogi er å se på hver XML node som ett dokument og dermed bruke vanlige tekstmetoder og deretter se på struktur. Et annet eksempel er tall som må tolkes annerledes enn tekst ved sammenligning mot andre tall og derfor ikke uten videre kan lagres i en tekstindeks.

2.3.5 Standardiserte spørrespråk mot XML

Spørrespråk er en viktig del av informasjonsgjenfinning ettersom det bestemmer hva det er mulig å søke etter og må formes etter hvilke muligheter det underliggende systemet støtter.

Spørrespråk beregnet på flate tekstfiler er kraftig begrenset i forhold til hva som kan gjøres mot XML dokumenter. Flate tekstfiler muliggjør sjelden noe mer avansert enn rene boolske spørringer for å finne dokumenter inneholdende ett eller flere ord ved bruk av AND, OR og NOT. XML derimot muliggjør at du effektivt kan søke ved å oppgi konteksten du ønsker å få treff på, enten det er forfatternavn, titler, byer, lokasjoner eller datoer. Dette gjør også at det krever mer av brukerne for å kunne søke effektivt i XML dokumenter. De to eksemplene under viser at slike søk er langt mer sammensatt og krever koblinger imellom informasjon fra forskjellige plasser i et dokument.

> *Query 1: simple IR-style query*
> *Find document components in articles.XML that are about ‘‘search engine’’. Relevance to ‘‘internet’’ and ‘‘information retrieval’’ is desirable but not necessary.*
>
> *Query 2: structured IR-style query*
> *Find document components in articles.XML that are part of an article written by an author with last name ‘‘Doe’’ and are about ‘‘search engine’’. Relevance to ‘‘internet’’ and ‘‘information retrieval’’ is desirable but not necessary. [35]*

Grunnet strukturen kan mer gjøres for å forbedre søk i strukturerte dokumenter. Om en bok har ti kapitler der navnet på alle kapitlene inneholder ordet ”informasjon” vil vanlige teknikker for informasjonsgjenfinning kunne returnere ved søk etter ”informasjon” ti relevante treff som alle hører samme dokumentet. Dersom strukturen tas i bruk ved å se på teknikker innenfor databaser kan de ti treffene aggregeres sammen ved å returnere de ti treffene som ett treff. Dette kan gjøres ved å returnere elementet over de ti treffene i hierarkiet basert på en algoritme. Men både støtte for aggregering og sammenfletting av resultater krever ressursintensive operasjoner og datatyper i form av trær som muliggjør navigering i hierarkiet over dokumentene.

Spørringer mot XML kan i enkelte tilfeller være mer avanserte enn bare søk etter ord eller fraser. Det kan være ønskelig å returnere alle titler på dokumenter laget av en viss forfatter, eller returnere forfatternavn der etternavn er x og fornavn y. Spørringene må likevel ikke bli for avanserte siden de må være tilpasset kunnskapsnivået til brukerne av systemet.

Det eksisterer i dag to offisielle standarder for søking og gjenfinning av data i XML. Disse er XPath og den nyere XQuery standarden som innehar flest muligheter.

XPath

XPath [36] er ikke et språk basert på XML men muliggjør å identifisere en spesifikk plass i et XML dokument om lag på samme måten som i et vanlig filsystem. Ved bruk av XPath kan det angis at hvilke som helst lokasjon i trehierarkiet til et dokument er interessant, inkludert attributter. Som eksempel peker ”/doc/chapter[5]/section[2]” på det

andre seksjonselementet av det femte kapittelet av dokumentet. I tillegg støtter XPath enkle spørringer som ”chapter[title="Introduction"]”, som returnerer kapitler med en tittel som inneholder ordet ’Introduction’. XPath muliggjør også enkle aritmetiske operasjoner eller manipulering av strenger. Men avanserte spørringer er ikke mulig å gjøre med XPath.

XML består av noder og for XPath eksisterer det sju forskjellige typer som kan nåes ved hjelp av XPath: rotnoder, elementnoder, tekstnoder, attributtnoder, kommentarnoder, instruksjonsnoder og navneområde. Eksempel på en mer avansert XPath spørring er ‘/cd[artist=’anja garbarek’ and (tracks<5 or tracks>10) and contains(title,’trick’)]. Her angis det at det er ønskelig med cd plater fra artisten Anja Garbarek, med imellom 5 og 10 låter og inneholder ordet ”trick” i tittelen på en låt.

XQuery

XQuery 1.0 [37] er et spørrespråk spesielt laget for spørringer rettet mot forskjellige datakilder inkludert semistrukturerte dokumenter. Det er altså ikke bundet mot XML og kan derfor brukes mot lignende strukturerte formater. I bunnen av XQuery ligger XPath og XQuery støtter derfor de samme funksjonene og operatorene som XPath. I XQuery er alt et utrykk med en verdi som svar, noe som gjør at 5 + 7 er et akseptabelt XQuery program, uten at XQuery definerer hvordan 5 + 7 faktisk skal bli behandlet. Hvordan uttrykkene utføres er opptil implementasjonen å bestemme.

Som tidligere nevnt tilsvarer XQuery for XML det SQL er for relasjonsdatabaser. Dette gjelder også faktumet at XQuery fokuserer på datagjenfinning, det eneste XQuery støtter for informasjonsgjenfinning er søking etter enkeltord i tekst, ikke rangering og andre informasjonsgjenfinningsfunksjoner. Men XQuery har et bredt bruksområde som ikke bare spenner seg over å søke i XML dokumenter. Blant de muligheter XQuery gir er [38]:

- Ekstrahering av informasjon for bruk i en Web Service
- Generering av sammendragsrapporter
- Transformere XML til HTML dokumenter
- Søk i webdokumenter for relevant informasjon

XSLT [39] kan gjøre mange av de samme tingene som XQuery og har lignende syntaks men er ikke like effektivt som XQuery.

Ettersom XPath og XQuery mer kan sammenlignes med SQL er de også tilsvarende fokusert på gjenfinning av data og ikke informasjon. På samme måte som at en brukers spørringer i en vanlig søkemotor kan bli konvertert til SQL spørringer gjelder også dette for XPath og XQuery. Dette siden inngående kjennskap om hvordan dokumentene er oppbygd ikke kan forventes å være tilgjengelig for en bruker slik som hvordan en SQL-database sin oppbygging ikke er kjent for andre enn systemadministratorer.

2.3.6 INitiative for the Evaluation of XML Retrieval Retrieval

INEX [16] er sponset av DELOS, ”DELOS Network of Excellence on Digital Libraries” [40]. INEX sees på i sammenheng med denne oppgaven grunnet viktigheten av initiativet innenfor informasjonsgjenfinning i semistrukturerte dokumenter. INEX kan til dels sees på som state-of-the-art innen informasjonsgjenfinning i XML men sees her på som grunnlag for senere studium av forskning gjort som en del av INEX.

INEX vart første gang arrangert i 2002 og har som mål å fremme forsking innenfor informasjonsgjenfinning, og da spesielt innen digitale bibliotek og semistrukturerte dokumenter. For å nå dette målet arbeider INEX med å støtte forskning på feltet ved å tilby de nødvendige fasiliteter som kreves for gjennomføring av forskning som kan sammenlignes med tilsvarende arbeid fra andre. Blant de hjelpemiddel INEX tilbyr er INEX samlingen som består av XML dokumenter, en standard for lik tolkning av søkeresultater og et forum som lar organisasjoner sammenligne resultater.

INEX samlingen (2004) består av artikler og journaler i fulltekst som er tilført ekstra semantikk ved bruk av XML. Denne samlingen består av 12 107 artikler på til sammen rundt 500mb ifra 12 forskjellige journaler fra perioden 1995 - 2002. Ettersom det er samlingen fra 2004 som har vært tilgjengelig for denne oppgaven er det den som er sett på videre i rapporten. I 2005 vart samlingen utvidet med litt over 4000 flere dokumenter mens dokumentsamlingen vart byttet ut for 2006.

I 2006 har INEX gått over til å bruke en samling ifra Wikipedia på 1,9 millioner dokumenter på totalt 100GB. En slik samling er sannsynligvis mer interessant ettersom informasjonsinnholdet er mer variert enn i IEEE samlingen.

INEX samlingens struktur

INEX er i motsetning til metadata tilpasset fulltekstdokumenter strukturert ved hjelp av XML og bruker en DTD for spesifisering av hva som er et godkjent INEX dokument.

En artikkel i INEX er en del av en journal og består av tre hoveddeler, FrontMatter, Body og BackMatter. FrontMatter tilsvarer i stor grad det som kan kalles metadata, mens Body inneholder selve artikkelteksten delt opp i seksjoner og mindre deler. BackMatter inneholder primært referanser / bibliografisk informasjon. Vanlig metadatasøk kan derfor i stor grad sees på som søk i FrontMatter delen av en artikkel sammen med informasjon om selve journalen, søk i fulltekstdokumenter blir i Body og om det er ønskelig å søke etter artikler som referer andre artikler må BackMatter delen tas i betraktning.

Ikke alle strukturdeler i skjemaet som hører til INEX samlingen tilfører innholdet semantikk. Samlingen inkluderer bl.a. også et sett med tagger for tekstrepresentasjon slik som bold ”<b>” og italic ”<it>”, noe som har blitt kritisert siden den blander innhold og presentasjon. Presentasjonsdata i XML strider mot en av hovedideene bak XML; skillet mellom presentasjon og data. Strukturen i INEX samlingen er også til dels komplisert siden den bærer et klart typografisk preg grunnet at innholdet opphavlig var strukturert for presentasjon i tidsskrift for lettere presentasjonen av innholdet. Grunnet dette opphavet eksisterer det tagger som er brukt om hverandre av historiske publiseringsgrunner [41] [42].

Type data	Elementnavn
Paragrafer	ilrj, ip1, ip2, ip3, ip4, ip5, item-none, p, p1, p2, p3
Seksjoner	sec, ss1, ss2, ss3
Lister	dl, l1, l2, l3, l4, l5, l6, l7, l8, l9, la, lb, lc, ld, le, list, numeric-list, numeric-rbrace, bullet-list
Overskrifter	h, h1, h1a, h2, h2a, h3, h4

Tabell 10 - Ekvivalente INEX elementnavn

Grunnet like elementnavn fører dette til at det kan utføres spørringer som er ekvivalente:

//article//sec[about(.//p, Computer)]

og

//article//ss2[about(.//item-none, Computer)]

er identiske.

INEX samlingen inneholder ikke definisjoner for datatyper utover hva det er mulig å anta basert på elementnavnene i skjemaet, men grunnet mye søppel inne i spesifikke tagger er dette vanskelig for mange datatyper som datoer.

Mange elementer i INEX inneholder også like typer data, er overflødige og ikke optimale for strukturell indeksering. Ulike elementer med likt innhold i en homogen samling øker vanskeligheten med å kombinere innholdet med andre heterogene samlinger basert på struktur.

Table 2: Tags ignored during indexing.

ariel	en	item-text	ss
art	entry	label	stanza
b	enum	large	sub
bi	f	li	super
bq	it	line	tbody
bu	item	math	tf
bui	item-bold	proof	tfoot
cen	item-both	rm	tgroup
colspec	item-bullet	rom	thead
couplet	item-diamond	row	theorem
dd	item-letpara	scp	tmath
ddhd	item-mdash	sgmlf	tt
dt	item-numpara	sgmlmath	u
dthd	item-roman	spanspec	ub

Figur 7 - Elementer relaterte til presentasjon i INEX

INEX sin søkesyntaks

Standardspørrespråket innenfor INEX kalles for NEXI og blir brukt til å definere de spørringene deltakerne må levere svar på. De to hovedtypene spørringer som her er interessant, er CO and CAS spørringer. CO står for ”Content Only” og søker over alle dokumenter uten spesifisering av strukturelle krav. CAS, ”Content and Structure”, søker basert på innhold og struktur, noe som gjør at de lokasjoner hvor det er ønskelig å finne svar må spesifiseres sammen med hva det er ønskelig å motta som resultat. CAS utnytter de mulighetene XML strukturen gir, men krever at strukturen er kjent for de som ønsker å lage spørringer ved bruk av NEXI.

Spørringer utformet i NEXI er grundig formulert med forskjellige beskrivelser av hva det er ønskelig å søke etter både i form av stikkord, hierarki og tekstlige beskrivelser som gjør en formulering i dette spørrespråket krevende. Det er derfor ikke et spørrespråk som er naturlig for vanlige brukere å nytte ved formulering av spørringer.

NEXI krever ikke støtte for flere datatyper enn tall, termer og fraser, og spesifiserer ikke hva som blir gjort med henhold til stemming, stoppord og lignende. Eneste er at på tall må aritmetiske operatorer støttes. Det er dermed opp til hver enkelt deltaker å bestemme hvordan de ønsker å utføre indeksering og søking utover de krav INEX stiller for å kunne delta. ”*Positive numbers, negative numbers and sequences of alphanumerics proceeded by an alphabetic character are all valid search words[…]”* [41].

Søkegrensesnitt i INEX

Et brukergrensesnitt for søking i XML skiller seg ut ifra vanlige søkergrensesnitt ved at de bør ta i bruk den strukturen som er i et XML dokument ved presentasjon av resultater. Dette for at en bruker lettere kan finne ut hvor i dokumentet relevant informasjon kan finnes og muliggjøre effektiv navigering over dokumentene. De ekstra søkemuligheter som XML tilbyr bør på en lettfattelig måte kunne tilbys igjennom et grensesnitt.

INEX sin egen søkemotor muliggjør søk der resultat blir returnert med tittelen på treffene som blir funnet sammen med stien til resultatet slik at direkte tilgang til de mest aktuelle områdene i dokumentene er mulig. Søkemotoren viser dermed at den bruker strukturen i dokumentene ved søk og presentasjon av resultat.

Siden den undersøkte INEX samlingen i dette tilfellet er homogen er det lett å bestemme hva som er ønskelig å vise som beskrivende tekst for hvert dokument i et søkeresultat. Med heterogene XML dokumenter blir dette vanskeligere siden det er vanskelig å vite hvilke element som fungerer som tittel, forfatter og lignende uten noen fast måte å definere dette på [43].

2.4 Informasjonsgjenfinning vha Z39.50

Z39.50 [44] er en internasjonal standard for kommunikasjon mellom bibliotek- og andre informasjonsrelaterte systemer. Grunnet Z39.50 sin popularitet er det relevant å se nærmere på denne standarden, men siden den er for omfattende undersøkes kun de delene som er mest aktuelle for denne oppgaven.

... the very fact that Z39.50 has been extensively used for long enough to be criticized as old fashioned is surely a testament both to its robustness and to the lack of any viable alternative.

New technologies such as XML and RDF certainly fulfil aspects of the information discovery and retrieval process better than basic Z39.50, but work is underway to capitalize upon this, and to tie such technologies more closely to Z39.50… [45]

2.4.1 Datatyper i Z39.50

Ettersom Z39.50 har en viktig posisjon og i stor grad blir brukt til å søke i strukturerte metadataformat er det naturlig å se på hvilke datatyper Z39.50 støtter og forskjellene på dem kontra de støttet av XML Schema.

I Z39.50 er datatyper definert som strukturelle attributt og inngår i et sett av flere støttede sett attributter. Det vanligste av disse settene er BIB-1 og den store utbredelsen av BIB-1 gjør det interessant å se på for tilsvarende bruk i sammenheng med indeksering av XML basert på XML Schema. ”*The most common attribute set is bib-1 used primarily for bibliographic searching [..]”* [46].

BIB-1 inneholder 16 datatyper, eller attributter, og de blir primært brukt for å spesifisere hvordan en søkestreng skal behandles og inneholder derfor begrenset semantikk. Formålet med dem er å spesifisere om søkestrengen skal behandles for eksempel som et år, noe som kan angi at det skal søkes etter årstall pluss/minus noen år rundt oppgitte verdi. De 16 attributtene er:

- Phrase: En eller flere grupper med karakterer oppdelt av mellomrom. Søkeverdien er den som er nøyaktig oppgitt med tanke på rekkefølge og tilhørighet. Trunkering kan brukes.
- Word: En gruppe karakterer uten mellomrom. Søkeverdien er eksakt i forhold til søkeverdien. Kan trunkeres dersom oppgitt.
- Key: En nøkkel spesifiserer en sekvens av karakterer ekstrahert ut i fra de bokstaver i et indeksert ord men som ikke nødvendigvis er et helt ord.
- Year: Som søketerm er år numerisk og består av fire tall.
- Date (normalisert): Dagen, måned, året og tiden en transaksjon eller hending tar/tok plass.
- Word List: En liste med ord avskilt med mellomrom. Rekkefølgen har ikke noen betydning. Kan trunkeres om angitt.
- Date (ikke normalisert): Ustrukturert angivelse av dag, måned og år.
- Name (normalisert): Navn som er strukturert i en spesifikk rekkefølge (Fornavn, etternavn)
- Name (ikke normalisert): Ustrukturert angivelse av navn
- Structure: Ordet har en struktur som er angitt av Use attributt eller av mottaker.
- Urx: Dokumentidentifikator
- free-form-text: Tekst gitt av avsender

- Document text: Tekst hentet ifra dokument
- Local number: Nummer som er signifikant for mottager
- String: Hele søkestrengen skal behandles sammenhengende.
- Numeric string: Tall som skal behandles som tekst.

De datatyper Z39.50 støtter for bruk ved søking er som nevnt definert i BIB-1 som ”Structure Attributes” og forteller Z39.50 hvilken datatype inndata er på og hvordan det skal behandles. Blant annet definerer et ”word” attributt eksakt søk uten noen form for trunkering eller stemming dersom ikke annet er definert.

Grunnet de begrensninger attributtene har vedrørende struktur og hvilke operasjoner som kan gjøres på dem, er det ikke relevant å overføre attributtene direkte til XML Schema siden de samme begrensningene ikke kan implementeres ved å spesifisere disse i skjemaet. De kan likevel nyttes om begrensningene utføres igjennom ekstern kode. 'BIB-1 Use Felt’ som ”name (normalized)” kan dermed beskrives ved bruk av komplekse felt i XML Schema. ”*It is thus possible to generate an XML Schema definition of the ASN.1 definition for Z39.50 […]”* [46].

En oversikt over hvordan Z39.50 kan brukes i XML finnes hos ASF [47] inkludert Z39.50 ASN.1 uttrykt ved bruk av XML Schema, men dette tar ikke for seg BIB-1 attributtsettet.

Felt i Z39.50

I tillegg til strukturelle attributt har også Z39.50 flere andre typer attributter. Blant disse er ”Use Attributes” som definerer hvilket ”felt” det kan søkes i, mellom annet tittel og forfatter. Ifølge Miller [48] er de fleste attributt-typer utover ”Use” attributtene forskjellig tolket av forskjellige leverandører og arbeid med å løse dette igjennom profiler har blitt forsøkt. En god oversikt over de forskjellige Use attributter finnes i ”Global Information Locator Service” settet [49]. Bib-1 utvidelser gjort etter 1995 inneholder også felt for Dublin Core.

2.4.2 Spørrespråk i Z39.50

Z39.50 har langt flere muligheter til å spesifisere mer nøyaktige spørringer. Standarden gir denne muligheten i form av forskjellige attributter som kan konfigureres av administrator og deretter nyttes av brukerne ved utforming av spørringer.

Søketype	**Spørring**
Enkeltfelt	title = cat
Boolsk over flere felt	title = XML and author = sanderson
Spørring over et område på felt	year: > 1950

Tabell 11 - Eksempel på spørringer i Z39.50

Som søkene viser, brukes ’use attributter’ for å søke etter dokumenter med visse ord i en tittel mens bruk av strukturattributter definerer hvordan ordene i spørringen skal tolkes av søkemotoren.

De attributter brukere har til rådighet ifølge BIB-1 attributtsettet til Z39.50 er:

Use Attributes: Definerer hva du ønsker å søke i, som forfatter, tittel, tema og lignende. I overført betydning til XML kan dette sees på som søk i spesifikke elementer i XML.

Relation Attributes: Relasjonen mellom søketermer og verdier i indeks. Om du ønsker å finne verdier som er større, mindre enn eller like søketermen, om stemming skal bli brukt og lignende.

Truncation Attributes: Hvilken del i indeksen det er ønskelig at søketermen skal stemme med. For eksempel som at indeksen skal inneholde ord som slutter med, begynner med eller inneholder søketermen.

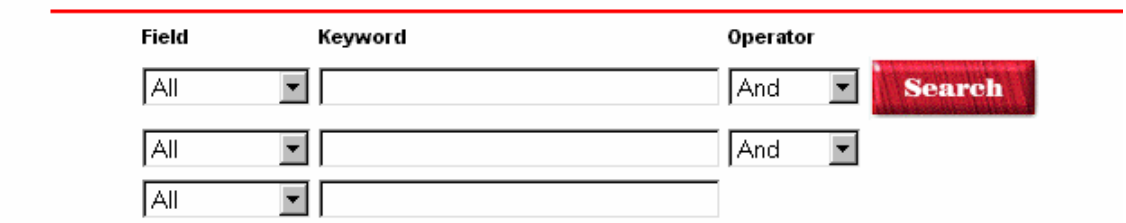
Completeness Attributes: Spesifiserer om andre termer enn de det blir søkt på kan eksistere i indeksen for de felt og underfelt det søkes i. For XML kan underfelt sees på som barneelement.

Position Attributes: Spesifiserer hvor i det indekserte feltet det søkes i en søketerm skal eksistere.

Structure Attributes: Studert tidligere i sammenheng med datatyper.

2.4.3 Søkegrensesnitt for Z39.50

En spesifikk implementert Z39.50 søkemotor [48] bruker de mulighetene tidligere nevnt om Z39.50 sine søkeattributter og muliggjør søk i felt og bruk av operatorer. ”Use attributter” er her merket som ”Field”, og ”Operator” definerer forholdet mellom de forskjellige søkene. Andre søkeoperatorer blir lagt inn i tekstfeltene under ”nøkkelord”. Grensesnittet er stort sett likt som for ustrukturerte dokumenter men muliggjør bruk av felt til å spesifisere søk mer nøyaktig.



Figur 8 - Z39.50 søkegrensesnitt

Basert på de treffene Z39.50 gir, blir en liste over antall treff presentert med ett treff per dokument sammen med kilden og unik identifikator for hvert treff. Fokuset her ligger på å få frem det som identifiserer dokumentet og ikke selve dokumentets innhold eller svar på en brukers informasjonsbehov.

3 State-of-the-art

Dette kapittelet ser på eksisterende systemer og metoder for gjenfinning av innhold i semistrukturerte dokumenter. Spesielt fokuseres det på de teknologier som brukes.

For å kunne lage en best mulig løsning er det foredelaktig å se på erfaringer, arkitekturer og hvilke fallgruver som eksisterer. En gjennomgang av forskjellige systemer som i dag er state-of-the-art er dermed naturlig.

Først undersøkes XML databaser og hvordan de fungerer for informasjonsgjenfinning. De studeres for eventuelle mangler i forhold til å kunne utføre gjenfinning av informasjon versa data, og hvilke funksjoner i de som eventuelt kan brukes. eXist og Berkeley DB XML er de primære systemene det blir sett på for XML databaser.

Videre sees det på eksisterende indekserings- og søkesystemer for semistrukturerte dokumenter og erfaringer de har gjort seg igjennom blant annet INEX. For XML søkemotorer er det interessant å se på i hvilken grad forskjellige datatyper har blitt brukt ved søking, teknikkene de har nyttet og hvilke erfaringer de har gjort seg, og hvorfor/hvorfor ikke dette har blitt gjort. I tillegg er det naturlig å se på om de har gjort seg nytte av XML Schema sammenheng med datatyper. Gjennomgangen gjøres ved å primært se på de tre artiklene fra Liu, S. et al. [2], Larson, R. [50] og Fuhr, N. / Großjohann, K. [51] samt søkemotoren Lucene [52] / PyLucene [53].

Ettersom det ikke er så mange eksisterende tilgjengelige systemer blir det primært en gjennomgang av artikler som tar for seg tilstøtende løsninger, både vedrørende databaser og søkesystemer.

3.1 XML Databaser for informasjonsgjenfinning

Med XML databaser menes det her ”Native XML” databaser som tidligere definert i kapittel 2.3.3. De forskjellige databaseproduktene blir ikke sett på hver for seg siden de i stor grad fungerer likt. Det undersøkes derfor på generelt grunnlag hvilke fordeler og ulemper en native XML database har sammenlignet med behovene til denne oppgaven.

Tidligere i rapporten er det nevnt at hovedankepunktene ved å bruke en XML database for informasjonsgjenfinning tilsvarer i stor grad de om eksisterer for relasjonsdatabasesystemer siden begge er basert på databaser. Nå er ikke dette lengre et reelt problem siden mange databasesystemer har tilnærmet seg informasjonsgjenfinning og tilbyr fulltekstindeksering av tekst og støtte for rangering. Men fortsatt mangler de mange av de ekstra prosesseringsmetodene som stemming, stoppordfjerning, casefjerning og andre muligheter rene systemer for informasjonsgjenfinning støtter. Årsakene til disse manglene er blant annet at systemene har fokusert på å bruke effektive indeksstrukturer for å gjøre systemene raske nok til å takle den ekstra mengden data XML har, i tillegg til behovet for å tilby eksakte treff.

Et eksempel på slike begrensninger i XML databaser er eXist [54] som ved søk basert på en spørring leter etter elementer inneholdende nøyaktige treff på de forskjellige

delene av spørringen; kun elementer med ord som passer eksakt ifra hvert dokument blir returnert. Resultatlisten sorteres først etter den interne dokumentidentifikatoren gitt ut i fra rekkefølgen dokumentet vart lagt inn i databasen, deretter sortert etter hvor i trehierarkiet treffene eksisterer. Noe informasjon om hvilket av elementene som er mest relevant er altså ikke tilgjengelig siden boolske spørringer er benyttet [15].

.

Artikkel	**Svarelement**
ic/1999/w4095	/article[1]/bdy[1]/sec[2]/ip1[1]
ic/1999/w4095	/article[1]/bdy[1]/sec[2]/ss1[1]/ip1[1]
ic/1999/w4095	/article[1]/bdy[1]/sec[4]/p[1]
ic/1999/w4095	/article[1]/bdy[1]/sec[4]/p[2]
ic/1999/w4095	/article[1]/bdy[1]/sec[4]/p[3]

Tabell 12 - eXist søkeresultat eksempel

Hovedårsaken til rekkefølgen i resultatsettet er at eXist støtter seg til XQuery spesifikasjonene som spesifiserer at elementene skal returneres etter rekkefølgen i treet. På grunn av kravet om eksakte treff kan eXist også returnerer de samme treffene flere ganger dersom det er flere deler av en spørring som stemmer med de samme elementene.

Native XML databaser støtter heller ikke inkludering av dokumenter som kan være relevante for et søk men ikke har direkte treff på en spørring. Dette begrenser gjenfinningsgraden og genererer derfor behov for egne systemer for inkludering av slike dokumenter i søkeresultatet [15]. Boolske søk fører til at mange relevante treff ikke kommer med i resultatet og gjør databaser som eXist mest effektive ved søk i homogene produktdatabaser eller bibliografer der mer eksakte treff er ønskelig å motta. En fordel er likevel at en høy søkepresisjon kan antas siden alle treff i utgangspunktet er relevante.

Som relasjonsdatabaser tilbyr mange XML databaser å utnytte tilsvarende krav til struktur ved å bruke XML Schema. Men en del native XML databaser krever ikke skjema for å akseptere XML dokumenter så lenge de er velformet. Dette gjør utviklingen lettere men vanskeliggjør bruk av avanserte metoder for indeksering av XML siden dataintegriteten er usikker [55]. Bruk av XML Schema er derfor av begrenset verdi, spesielt dersom data kommer ifra mange kilder og skjema delvis mangler. Årsaken til dette er at ad hoc skjema sjelden tilfører noe ekstra om de bare gjelder for noen få dokumenter. Dette kan skape problemer og dårlig kvalitet på søkeresultat sett ifra et informasjonsgjenfinningsståsted grunnet manglende konsistent informasjon ifra eventuelle skjema.

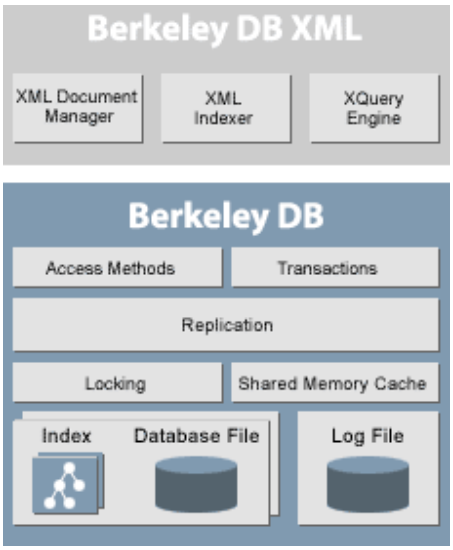
> *Unfortunately, it is also a feature that makes database administrators pull their hair out, due to the risk of low data integrity. The old adage of using the right tool for the job definitely applies here. If you need strong schema structure, then either make sure you use a NXD that supports schemas or find another way to store your XML data.* [56]

De indekser brukt av XML databaser har begrenset funksjonalitet og støtter ikke kravene til hastighet som er ønskelig ved store samlinger grunnet en generell oppbygging. Den generelle oppbyggingen eksisterer primært for å støtte alle nødvendige standarder. Egne indeksstrukturer er derfor nødvendige siden de kan optimaliseres til å støtte spesifikke operatorer ut i fra hvilke krav hvert system setter til funksjonalitet [57].

Digitale bibliotek har i liten grad hatt XML databaser og tilhørende spørrespråk som basis for sine prosesser. Strukturen i XML har blitt utnyttet ved bruk av ad hoc metoder grunnet fokuset på informasjonsgjenfinning og ikke datagjenfinning som støttet av XQuery og XPath. De senere årene har dette endret seg de og de utnytter nå i økende grad XML til å strukturere sine metadata siden de vanligvis er av semistrukturert karakter. Bruk av XML databaser blir dermed mulig ettersom spørringer kan gjøres mot tilgjengelige data på generell basis fremfor bruk av ad hoc løsninger. Likevel kreves det at skjemaer blir fulgt slavisk og spiller rollen som tabelldefinisjon for semistrukturerte samlinger. Digitale biblioteker har dermed en fordel siden de i stor grad innehar et fåtall metadatastandarder basert på felles skjema som i tillegg til en viss grad er konverterbare imellom hverandre [20].

Berkeley DB XML

Berkeley DB XML er en native XML database som støtter datarettet gjenfinning av XML dokumenter. Berkeley bruker en normal database i bunn med et XML lag over som muliggjør behandling av XML dokumenter som XML, både ved innlegg, manipulasjon og uthenting igjennom bruk av XPath, XUpdate og XQuery.



Figur 9 - Berkeley DB XML oppbygging

Siden Berkeley DB XML er bygd over Berkeley DB deler de mye av den samme arkitekturen. I tillegg har DBXML ekstra komponenter for forvaltning av dokumenter og nødvendige indekser for effektiv utføring av XML spørringer igjennom en egen XQuery motor. Grunnet XQuery støtte støtter den ikke rangering av resultater.

Berkeley DB XML har også begrenset støtte for noen datatyper, men kan ikke bruke et XML Schema til å definere hvilken datatype et element inneholder. Den inneholder heller ikke støtte for case usensitivt søk eller tradisjonelle IR teknikker som stemming og stoppordfjerning. Dette gjør at Berkeley DB XML fungerer dårlig for indeksering basert på datatyper.

XML Schema kan brukes av Berkeley DB XML til å muliggjøre validering av enkeltdokumenter basert på innhold, men den krever ikke at alle dokumenter har tilhørende XML Schema. På grunn av dette trenger ikke en samling i DBXML å bestå av homogene dokumenter basert på samme skjema. DBXML skiller seg dermed ifra vanlige relasjonsdatabaser og også ifra andre XML databaser [58]. Den muliggjør heller ikke bruk av denne informasjonen videre etter at databasen er ferdig med validering av dokumentene. Uthenting av spesifikke datatyper ifra Berkeley DB XML er altså ikke støttet.

Andre XML databaser eksisterer også, men de fleste har tilsvarende funksjonalitet og mangler som Berkeley DB XML. Mange systemer er i tillegg ikke ferdigutviklet og mangler full støtte for mange XML standarder. Flere av systemene er tilgjengelig med kildekode slik at andre kan legge til ny funksjonalitet om ønskelig, et eksempel er XIndince [59]. For en oversikt over flere systemer kan Bourret, R. [60] konsulteres.

3.2 Bruk av XML Skjema

Det eksisterer mange forskjellige XML skjema men det er bare DTD og XML Schema som er offisielle standarder fra W3C og det er derfor bare de det sees på for hvordan datatyper kan beskrives og til bruk for informasjonsgjenfinning.

Fuhr, N. [61] sier at en DTD sine fordeler for informasjonsgjenfinning er at de metoder som skal brukes kan angis basert på dokumenttype og for spesifikke elementer som personnavn, dato og klassifikasjonskode ettersom strukturen angir datatype. Han nevner ikke noe om sammenkobling av tilsvarende informasjon i forskjellige DTD skjema.

De fordelene med XML Schema for informasjonsgjenfinningssystemer som blir fremhevet er at datatyper kan angi språk, tesaurustermer / klassifikasjonskode, geografiske posisjoner og andre tilsvarende typer. Et problem eksisterer likevel siden de ikke kan defineres på et syntaktisk nivå og at typesjekking ikke er mulig selv om datatyper relevante for informasjonsforvaltning kan defineres. XML Schema er primært beregnet for spesifisering av datainnhold og ikke informasjon slik Fuhr ser det.

Siden DTD ikke tar for seg innholdet i elementer i form av datatyper men bare ser på elementstruktur muliggjør ikke DTD standarden at datatyper kan angis imellom heterogene dokumenter siden strukturen er forskjellig. Mer detaljerte studier av DTD er derfor ikke like relevant videre i denne oppgaven.

XML Schema

XML Schema er som tidligere nevnt en nyere standard for definering av skjema for XML og er ennå ikke veldig utbredt i brukt, dette til tross for de muligheter den tilbyr kontra eldre standarder som DTD. I Dublin Core sine anbefalinger er dette nevnt som et av de viktigste punktene:

> *Recommendation 1. Implementors should base their XML applications on XML Schemas rather than XML DTDs. Approaches based on XML Schemas are more flexible and are more easily re-used within other XML applications. In some cases it may be sensible to provide both an XML Schema and a DTD for the application. Where XML Schemas are not used, a DTD should be provided instead. The DCMI maintains a list of XML schemas that are in use in projects or products using DCMI metadata.* [62]

I TOX [63] blir bruk av XML Schema spesielt nevnt i forbindelse med de muligheter skjemastandarden gir til å øke kvaliteten på alle prosesser fra lagring, indeksering til prosessering av spørringer i et søkesystem. I sin rapport fokuserer Barbosa, D. et al. på behovet for at forskjellige skjematyper som DTD og XML Schema støttes. Den sistenevnte spesielt ettersom den muliggjør mer detaljert informasjon om de strukturelle egenskapene ved et dokument, noe som kan brukes til å øke kvaliteten på de data de nevnte prosessene returnerer.

Pal, S. et al. [64] ser det på hvordan XML Schema kan brukes ved lagring av XML dokumenter i Microsoft SQL Server 2005, denne har en egen datatype ”XML” for lagring av XML dokumenter. Artikkelen diskuterer hvordan datatypeinformasjonen i XML Schema kan brukes for koblinger mellom de primitive datatypene i XML Schema og tilsvarende datatyper i relasjonsdatabasen og bli brukt som metadata for dokumenter. Bruken av datatyper muliggjør effektiv behandling av XML og domenebaserte indekser basert på datatyper for effektiv gjenfinning.

Fuhr, N. et al. [51] ser på hvilken nytte muligheten til å definere datatyper relevante for informasjonsgjenfinning har og hvordan dette kan gjøres igjennom bruk av XML Schema. Relevante datatyper som blir foreslått er tekst på forskjellige språk, klassifikasjonsskjema, tesauri og personnavn. Igjennom definisjoner av disse i XML Schema kan dermed også vanlige informasjonsgjenfinningsoperatorer brukes på innholdet. Videre bør forskjellige typer defineres i et hierarki ala ”Tekst – Vestlige språk – Engelsk”, noe som ikke er støttet fullt ut slik datatyper kan defineres i XML Schema. Å spesifisere nok struktur igjennom et Schema er heller ikke mulig på en måte som gjør at de kan overholdes. Dette gjør spesifisering av at et felt inneholder engelsk tekst som en restriksjon på feltets innhold vanskelig. Spesifisering av datatyper med struktur kan likevel gjøres igjennom å beskrive de som applikasjonsinformasjon og dermed bli forstått som kommentarer av skjemaprosessoren.

> *Thus, we use XML Schema for describing the schema of a document base, by specifying the document structure and the datatype of elements. Due to the typing problems described before, the schema specification may only refer to the set of data types provided by the actual implementation.* [51]

På mange måter tilsvarer dette det RDF er ment brukt til, men denne standarden er primært ment å kunne beskrive en ressurs eksternt, og ikke inkluderes inne i ressursen i seg selv [65].

Ifølge Thompson, H. og Garshol, L. [66] ligger ansvaret på datapresentasjon ved bruk av XML Schema hos den som implementerer et skjema i et dokument. Den som definerer skjemaet definerer den leksikalske formen mens implementator må forholde seg

til skjemaet dersom spesifisert struktur ønskes overholdt. Siden XML Schema mangler muligheten til å definere mer nøyaktig semantikk på et lavere nivå er dette den eneste muligheten applikasjoner kan være sikre på hvilken form en dato er angitt i et dokument. På bakgrunn av dette påstås det at XML Schema mangler støtte for god interoperabilitet imellom heterogene dokumenter.

Mixed content i XML Schema

Et XML element kan inneholde tekst eller element men det er også lovlig at et element kan inneholde både andre element og tekst. Dette kalles ”mixed content” og muliggjør setninger der enkeltdeler kan merkes:

<sentence>I går var jeg og <aktivitet>kita</aktivitet>.</sentence>

Dette muliggjør å inkludere semantikk og beskrive enkelte deler av en setning der det er relevant. Det er også mulig å beskrive liknende igjennom XML Schema med noen forbehold i relasjon til datatyper:

> *W3C XML Schema specification defines many different built-in datatypes. These datatypes can be used to constrain the values of attributes or elements which contain only simple content. These datatypes are not available for constraining data in mixed content.* [67]

Ettersom alle simple datatyper må være derivert av en eksisterende standard datatype hjelper det altså ikke å definere egne datatyper for dette heller. Ut i fra XML Schema sin definisjon kan følgende identifiseres som grunnlag for hvorfor det ikke er mulig:

> *1 If the {base type definition} is a complex type definition, then all of the following must be true:...*
> *1.4.1 The {content type} of the {base type definition} and the {content type} of the complex type definition itself must be the same simple type definition.*
>
> *If you derive a mixed complex type (with some elements in the content) by extension from a complex type with simple content, then 1.4.1 applies, and the content type of the base type (which is a simple type definition) must be the same as the content type of the derived type (i.e. the same simple type definintion). That isn't the case.* [68]

Definisjon av hva som står imellom underelementene av et ”mixed content” element er altså ikke mulig og begrenser nytten av å ha muligheten til å definere dette for alle andre elementer i et dokument. En omskriving av et XML dokument er derfor nødvendig for å støtte bruk av datatyper i XML Schema dersom ”mixed content” elementer eksisterer.

> *Unlike DTDs, W3C XML Schema mixed content doesn't modify the constraints on the sub-elements, which can be expressed in the same way as simple content models. While this is a significant improvement over XML 1.0 DTDs, note that the values of the character data, and its location relative to the child elements, cannot be constrained.* [69]

Store standarder som XML Schema må begrenses på noen områder og en gjennomgang av XML Schema før standarden vart vedtatt poengterte flere mangler ved XML Schema og datatyper [70] som er problematiske:

- Noen av de primitive datatypene som xs:float og xs:double i XSD overlapper hverandre.
- Støtter ikke validering av andre strukturerte datatyper enn URI og datoer.
- For mange primitive datatyper
- Brukerne bør kunne definere en transformasjon imellom forskjellige datoformat igjennom XML Schema. For eksempel bør en dato utrykt som "24 mai 2002" kunne formateres til ISO format ("2002-05-24") eller til et annet format som er forståelig av en datamaskin.

3.3 Lucene

Lucene er en søkemotor for ustrukturert tekst som i flere tilfeller har blitt brukt til å kunne søke i XML dokumenter. To eksempel på hvordan dette har blitt gjort sees på.

Case Study: Enabling Low-Cost XML-Aware Searching Capable of Complex Querying
Jockman, B. et al. [71] bruker Lucene til å indeksere og søke i XML med hell ut i fra de krav de stiller, med blant annet ønske om å kunne søke ved bruk av struktur. Dette viser at Lucene kan brukes til å søke i XML for mange av de vanligste bruksområder der det måtte være ønskelig å søke i XML. De ser ikke på hvordan heterogene dokumenter kan søkes i der det er ønskelig å finne like data imellom to dokumenter med forskjellig struktur. Noen av de søketypene de har implementert ved bruk av Lucene er støtte for søk i:

- Hele #PCDATA innholdet til et XML dokument
- #PCDATA innhold i definerte elementer
- Prosesseringsinstruksjoner basert på navn og innhold
- Attributter
- Elementer med spesifikke foreldreelement
- Elementer på en spesifikk plass under et foreldreelement
- Elementer med spesifikke slektninger

De påpeker likevel at Lucene krever fra 2 til 10 ganger større indeks enn hva faktisk størrelse på de indekserte dokumentene er, noe som tilsier at store dokumentsamlinger ikke kan effektivt indekseres med Lucene ved bruk av deres metode.

Parsing, indexing, and searching XML with Digester and PyLucene
IBM [72] har en artikkel som viser hvordan PyLucene kan brukes til å indeksere homogene dokumenter basert på datatypen de innehar for bestemte dokumentstier i hierarkiet. Dette muliggjør å indeksere data ifra XML basert på datatyper dersom systemet først har fått regler for hvordan data i de forskjellige stiene skal indekseres. En angivelse av at stien ”address-book/contact/name” tilsvarer et felt ”name” i PyLucene og muliggjør søk på datatyper uten å måtte angi hele stien under søk.

3.4 Utnyttelse av datatyper i ustrukturerte dokumenter

Ustrukturerte dokumenter inneholder ikke noen mulighet for uthenting av datatyper uten at innholdet er såpass godt kjent at det kan gjøres automatisk eller ved bruk av tekstanalyse.

Google muliggjør til en viss grad søk etter informasjon basert på bruk av datatyper ved å spesifisere i hvilken kontekst det er ønskelig å gjøre et søk. Dette gjøres ved å angi konteksten som forstaving til selve søkeordet. Et eksempel er ”movie:Braveheart” der det spesifiseres at Braveheart er tittelen på en film. Dette fører til søk i en begrenset base av utvalgte dokumenter og ikke over hele samlingen, altså er det spesifikt indeksert for å kunne tilby dette. Det gir deg heller ikke muligheten til å definere at i tilfelle det er flere filmer med samme navn ønskes den med Kevin Costner. De operatorene Google tilbyr er derfor begrenset til å kun kunne brukes effektivt direkte overført til dokumentsamlinger med struktur og metadata. Men selve ideen med ”movie:” attributt kan brukes til å definere hvilken kontekst hvert ord skal ha i en dokumentsamling.

Søketype	**Spørring**
Boolsk:	vacation london OR paris
Ikke fjern stoppord:	Star Wars Episode +I
Rangequery:	DVD player $250..350
Filmsøk:	movie:Braveheart

Tabell 13 - Eksempel på Google søk

Google støtter også delvis søk basert på datatyper i form av omtrentlige spørringer. For søk på verdier mellom 250 og 300 kan ”250..300” gis som spørring. Et forsøk på søk etter ”Apple..Banana” gir derimot ikke noe annet svar enn ”Apple” og ”Banana”, noe som viser at de begrenser ut i fra datatype som her er tallverdier. Grunner til denne begrensningen er at antallet ord mellom ”Apple..Banana” er mye større enn tilsvarende for tall, noe som gjør det ineffektivt å tilby denne muligheten. Google gir ellers ikke uttrykk for at de støtter forskjellige datatyper utover tekst og tall som nevnt ovenfor. De prøver likevel igjennom oppslag i forskjellige databaser å finne ut om en spørring har en mening ut i fra innholdet i de forskjellige basene og foreslår eventuelle treff i grensesnittet.

3.5 Indeksering og søking i XML

> *If you ask a database for a record with date > 1999 and it reports a record with date = 1999, that's an error. If you ask an information retrieval system for documents with yr > 1999 and it returns one with yr = 1999, that's not an error, it's just somewhat less relevant than one that matches the clue precisely [..]* [73]

Flere systemer bruker i dag en kombinasjon av XML databaser og systemer for informasjonsgjenfinning i vanlige ustrukturerte dokumenter med ekstra tilpassing som gjør at XML kan indekseres og søkes i. Blant disse er "RMIT INEX" [57] systemet som kombinerer eXist og Lucy. eXist brukes som XML database, mens Lucy er en søkemotor som indekserer hele dokumenter uten å støtte gjenfinning av enkeltkomponenter.

Gjenfinning av enkeltkomponenter gjøres av eXist, jamført med hvordan XML databaser fungerer. Kombinasjonen av Lucy og eXist begrenser siden Lucy ikke støtter XML og dermed ikke kan benytte seg av de muligheter XML innehar. Returnering av XML dokumenter der hvert dokument er ett resultat fører til at forskjellen mellom dette og tradisjonell informasjonsgjenfinning i ustrukturerte dokumenter er liten. Å basere seg på resultatene ifra en database er lite hensiktsmessig for informasjonsgjenfinning.

Problemer relatert til strukturelle forskjeller mellom to heterogene XML samlinger er ikke sett på grunnet liten relevans for oppgaven. Det sees heller ikke på hvordan lingvistikk kan brukes for å identifisere meningen til innholder utover ren tekst, noe som er et annet fagfelt.

3.5.1 Datatypebasert indeksering

Med datatypebasert indeksering menes det indeksering av et dokument der deler av innholdet kan identifiseres som en datatype. Datatypen bestemmer hvordan data normaliseres, indekseres og kan søkes etter.

Innenfor INEX er datatyper lite brukt, delvis på grunn av at det bare eksisterer DTD skjema for INEX samlingen. Mange påpeker også at vanlige datatyper som tall, dato og tekst er av liten interesse å kunne indeksere forskjellig siden de er for udetaljerte. Det påstås også at indeksering av slike datatyper er mer interessant for andre semantisk rikere samlinger enn hva tilfellet er med INEX samlingen. Blant typer samlinger som nevnes [74] som mer aktuelle er dokumenter inneholdende informasjon for kjemiske prosesser, finansiell informasjon og geografiske lokasjoner. Felles for alle er at de har mer strukturert innhold enn tilfellet er for artikler.

Liu, S. et al [2] nevnt tidligere i rapporten er en av de få som har sett på bruk av datatyper i sitt forslag til XML informasjonsgjenfinningssystem. Spesielt blir det bemerket hvor viktig det er med korrekt indeksering av dokumenter og det angis som en nøkkelfaktor sammen med korrekt rangering av søkeresultatene. De sier også at effektiv informasjonsgjenfinning i XML krever forskjellige indekseringsmetoder basert på innholdet. Feil indeksering av elementer uten semantisk verdi og bruk av feil datatype kan føre til dårligere søkeresultat. Indekseringsteknikker må derfor brukes for å få datoer tolket rett og fjernet annet som er lite ønskelig slikt som tagger primært for presentasjon av innholdet.

I samme artikkel avslutter de med å bemerke at lite arbeid har blitt gjort på alternative indekseringsmetoder for XML. Det har primært blitt fokusert på å inkludere strukturen i XML, og ikke datatyper eller annen semantikk ved at de tradisjonelle IR metodene i stor grad direkte har blitt brukt på XML med små endringer. Mye av grunnen til dette er som tidligere nevnt og påpekt av flere at INEX samlingen ikke inneholder nok semantikk til å kunne indekseres basert på datatyper. Bruk av datatyper krever da også tilhørende XML Schema med rik annotering. Liu, S. et al. påpeker at systemet deres fikk den høyeste gjennomsnittlige søkepresisjonen sammenlignet med andre INEX 03 resultater. Det kan derfor antas at indekseringen de gjorde likevel har en effekt, selv om de ikke sier noe spesifikt om hvilken del av systemet som førte til høyere presisjon enn andre systemer.

Fuhr, N. et al. [51] sammenligner en datatype med et metadatafelt i Dublin Core [30]; dc.title, dc.author og kobler det opp mot bruk av egne søkeoperatorer for hver datatype. Det foreslåes bruk av egne søkeoperatorer på elementer som inneholder for eksempel personnavn og at likhetssøk bør kunne tilbys på slikt innhold, mens tekniske dokumenter inneholdende verdier bør støtte sammenligningsoperatorer som større/mindre enn på flyttall. Dette viderefører de til å foreslå muligheten for elementer med forskjellige datatyper der hver datatype har sine egne spesifikke søkeoperatorer. For å støtte informasjonsgjenfinning best mulig anbefaler de å gjøre dette med støtte for omtrentlige verdier og muliggjøring av søk som ”målinger tatt rundt 20 grader celsius”. Hovedutfordringen er at de eksisterende XML teknologiene støtter dårlig de krav informasjonsgjenfinning har til å definere passende datatyper. Dette siden de sjelden kan defineres på et syntaktisk nivå tilsvarende datatyper for datagjenfinning. Omveier eksisterer dog ved å bruke xs:appinfo:

> *On the other hand, XML Schema does not deal with (vague) predicates of datatypes; they can be listed as application info only and are treated like comments by the schema processor. Thus, we use XML Schema for describing the schema of a document base, by specifying the document structure and the datatype of elements. Due to the typing problems described before, the schema specification may only refer to the set of data types provided by the actual implementation.* [51]

Søk i heterogene dokumentsamlinger der bruker ikke kjenner strukturen krever støtte for søk etter datatyper uten at strukturelle begrensninger må gis. Spesielt gjelder dette Internett; det kan ikke antas at en bruker kjenner strukturen til et dokument eller vet hvor i et dokument treff finnes.

> *As another example, a user may wish to search for a value of a specific datatype in a document (e.g. a person name), without bothering about the element names. Thus, appropriate generalizations should be included in the query language.* [51]

> *[..] Such a situation arises frequently on the Web; a user visits an (XML) Web site, but does not know (and does not want to know) how the data is stored at that Web site.* [75]

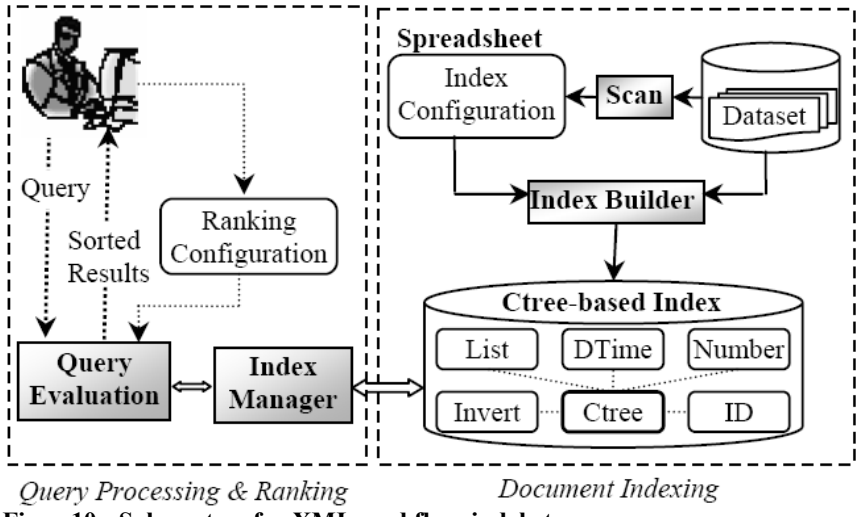
Ved å bruke datatyper kan søk lettere gjøres i heterogene dokumenter uten at søkeresultatene inneholder feilaktige data.

Bruk av datatyper [51] skaper spesielt problemer for dårlig markert tekst tilsvarende INEX samlingen. I denne samlingen består mange elementer som skal inneholde tall også av andre alfanumeriske karakterer som ikke skal være der og som skaper problemer for tidligere XPath versjoner og begrenser muligheten til bruk av datatyper. ”*Not only do we need rules for converting text to numbers that are different from the rules in XPath, we need to interpret comparisons fuzzily.”* [73].

3.5.2 Konfigurasjon av indeksering

For å støtte opp om konfigurasjon av indekseringen kan det brukes en kombinasjon av automatisk innsamling av statistikk og manuell konfigurering [2]. Statistikken genereres basert på innholdet i en samling og kan deretter nyttes til å bestemme hvordan hvert enkelt element skal indekseres ut i fra parametrer for datatyper og prosesserings-operatorer. Statistikken muliggjør at personer som ikke er erfarne i hvordan en XML samling er bygd opp kan få informasjon om hvordan konfigurasjonen bør gjøres. Grunnen til at det brukes en slik innsamling av statistikk er at mange samlinger ikke har skjema knyttet til seg som effektivt muliggjør å ekstrahere nødvendig informasjonen direkte.

I figuren under består databasen av flere indekser som hver inneholder data ifra forskjellige elementer av samme datatype. De forskjellige indeksene kan deretter brukes til å generere spørringer optimalisert basert på datatype [2]. Det fokuseres i motsetning til i andre løsninger også på at heterogene XML dokumenter inneholder ikke bare forskjellige elementer, men kan også ha forskjellige datatyper for like elementer dersom XML Schema ikke er spesifikt eller rikt nok.



Figur 10 - Søkesystem for XML med flere indekstyper

Feil indeksering kan føre til feil resultat og gjør konfigurering essensielt. Dette krever at det heterogene innholdet i et XML dokument går igjennom varierte

prosesseringsoperasjoner før indeksering og også forskjellige indekstyper for effektiv gjenfinning av de forskjellige typene.

De forskjellige parametrer angitt som konfigurerbare er interessante med tanke på å se om de samme kan defineres i XML Schema for tilsvarende samlinger. Artikkelen spesifiserer også at det er hentet inspirasjon ifra XML Schema ved valg av hvilke datatyper det var behov for å kunne indeksere.

I artikkelen foreslår de fem forskjellige indekstyper; Invertert, liste, nummer, dato og id. De forskjellige indekstypene er definert for å støtte vanlige XML datatyper angitt av XML Schema som xs:string, xs:decimal og noen andre spesielle dataverdier som IDREF attributt. Konfigurering av hvordan data skal indekseres i de forskjellige indeksene defineres basert på operatorer som er manuelt angitt ved hjelp av statistikk og kunnskap om dokumentene. En del valg for hver datatype / element som skal indekseres må derfor tas. Operatorene de tilbyr for dette er:

- Element-indekstype: Indeks eller ikke indeks. Om elementet skal indekseres eller ikke.
- Prosesseringsoperatorer: 1. Tokentype: Om tall, ord, miks eller alle deler av en tagg sitt innhold skal indekseres. 2. Stoppord: Om stoppord skal fjernes. 3. Stemming: Om endinger på ord skal fjernes. 4. Om store bokstaver skal fjernes eller tas vare på.
- Type indeks: 1. Ingen indeks, 2. Invertert indeks, 3. Nummer, 4. Dato, 5. Liste og 6. ID. Ingen indeks mener at ingen noder i gruppen skal indekseres.

I CoXML [76] brukes det ikke konfigurering av hvordan innhold i XML elementer skal indekseres, dette skaper problemer siden alt innhold blir brukt stemming på. Derfor ser de på konfigurering av indekseringsprosessen som en viktig del av deres fremtidig arbeid. Konfigureringen bør ta for seg forskjellige prosesseringsmetoder og indekstyper basert på en verdi sine karakteristikker.

> *For example, for a content-only (CO) query "web, internet", the document fragment "<author><snm>webb</snm></author>" will be returned as an answer since "webb" and "web" share the same stem: "web".* [76]

Fuhr, N. et al. [51] foreslår lignende indeksering som Liu men fokuserer mer på bruk av XQuery/XPath som spørrespråk i sin artikkel og går ikke dypere inn på hvordan indekseringen gjøres utover at XML Schema brukes sammen med XML dokumentet.

Cheshire-systemet [50] er bygd for å støtte informasjonsgjenfinning i forskjellige typer dokumenter, ikke bare XML. Systemet er derfor laget som et konfigurerbart system med muligheter til å bestemme hvordan indeksering skal gjøres ved å endre en SGML-basert fil. Systemet er i bunnen basert på Z39.50 og bruker konfigurasjonsfiler til å oversette mellom Z39.50 og andre filformater. Flere forskjellige indekstyper er støttet og hvilke operasjoner som skal gjøres på data før de blir indeksert kan bestemmes for hvert element. Et slikt system krever egne konfigureringer av systemoperator for hvert eneste skjema og er derfor problematisk å bruke i heterogene samlinger med mange forskjellige dokumentstrukturer. En av hovedgrunnene til at dette er en utfordring er at de som legger

til nye samlinger må ha gode kunnskaper om hver enkelt XML samling sin struktur, noe som er tidkrevende å tilegne seg.

> *This configuration file is subsequently used in search processing to control the mapping of search command index names (or Z39.50 numeric attributes representing particular types of bibliographic data) to the physical index files used and also to associated component indexes with particular components and documents. This configuration file can be treated as if they were a single database.* [50]

Fordelene med å ha slik konfigurering er at den kan mappe forskjellige strukturer i XML samlinger til felles felt tilsvarende "use attributter" i Z39.50 slik at blant annet forfattere kan søkes etter i heterogene samlinger uten at elementnavnene trenger å være like. Søk i virtuelle felt sammenlignes med en form for distribuert søk tilsvarende Z39.50 protokollen. Det påpekes da også at en av fordelene med en slik konfigurering fremfor å bruke elementnavn direkte er at mange elementer inneholder samme informasjon selv om de har forskjellige navn. Søk kan dermed få langt bedre gjenfinningsgrad siden flere relevante felt kan søkes i på en gang uten å måtte kjenne til de fysiske feltnavnene. Figuren under ifra INEX viser eksempel på slike tilfeller der mange tagger med forskjellige navn har likt innhold.

Name	Description	Contents
docno	Digital Object ID	//doi
pauthor	Author Names	//fm/au/snm //fm/au/fnm
title	Article Title	//fm/tig/atl

topic	Content Words	//fm/tig/atl
		//abs
		//bdy
		//bibl/bb/atl
		//app

topicshort	Content Words 2	//fm/tig/atl
		//abs
		//kwd
		//st

date	Date of Publication	//hdr2/yr
journal	Journal Title	//hdr1/ti
kwd	Article Keywords	//kwd
abstract	Article Abstract	//abs
author_seq	Author Seq.	//fm/au @sequence
bib_author_fnm	Bib Author Forename	//bb/au/fnm

Figur 11 - Ekvivalente tagger i INEX

Automatisk konfigurering av indeksering

Med automatisk indeksering er det ønskelig å kunne trekke ut data ifra heterogene dokumenter/samlinger som direkte kan brukes til å koble sammen ulike elementer med likt innhold imellom dokumentene. De metoder som Liu, S. et al. bruker for manuell konfigurering kan videreføres til automatisk konfigurering ved å lage regler som gjør det mulig basert på innsamlet statistikk.

Fuhr, N. et al. [51] har brukt automatisk konfigurasjon ved at datatypene som brukes hentes ut ifra XML Schema tilknyttet til dokumentet. Det er derfor ikke nødvendig for systemansvarlig å gjøre noe for å tilpasse dokumentets struktur til de andre ettersom dette kan gjøres basert på XML Schema med felles datatyper. De nevner dog ikke noe om det er heterogene eller homogene samlinger de har eksperimentert på.

Automatisk konfigurasjon av indekseringsprosessen er et lite studert tema siden de fleste holder seg innenfor INEX initiativet og ikke forsker på hvordan ukjente, heterogene samlinger kan inkluderes i en eksisterende samling. De aller fleste digitale biblioteker fokuserer på spesifikke strukturer som eksisterer i forskjellige metadataformat, og har ikke behov for å automatisere konfigurasjon.

3.5.3 Bruk av datatyper i spørrespråk

Spørrespråk er ikke tema i denne oppgaven men muliggjør som tidligere nevnt å bekrefte hvorfor det er viktig å støtte søk basert på innhold, også i heterogene XML dokumenter.

Som Google og Z39.50 sin søkeprotokoll viser er det relevant å kunne søke basert på strukturert informasjon, noe som kan være vanskelig med XML grunnet forskjellig struktur som vanskeliggjør denne typen søk uten at en effektiv metode brukes.

Spørringer innenfor INEX muliggjør akkurat det samme som Google og Z39.50, men begrenser det til homogene XML dokumenter siden strukturelle begrensninger må oppgis. Spørrespråkene kan derfor brukes som en begrunnelse for at det er viktig å kunne tilby slike søkemuligheter også i heterogene XML dokumenter.

3.6 Utfordringer ved informasjonsgjenfinning mot XML

De mangler som XML har i forhold til å støtte effektiv informasjonsgjenfinning går spesielt på problemet med heterogene samlinger, manglende strukturell informasjon om selve oppbyggingen av fulltekstdokumenter og hvordan heterogene samlinger skal behandles for å tilby søk uansett strukturell oppbygging.

Luk, R. et al. [34] påpeker at mange semistrukturerte dokumenter bare inneholder metadata i form av forfatter, navn, tittel og publiseringsdato fremfor strukturerte data om hvordan fulltekstdokumentet er strukturert med paragrafer, navn, datoer og lignende inne i selve teksten. Lehtonen, M. [77] har studert heterogene samlinger med XML dokumenter inneholdende fulltekst med mål om å øke søkepresisjon. Han konkluderer med at det eneste med XML som kan stoles på og er felles for alle XML dokumenter er strukturen i hvert enkelt dokument, uansett dokumenttype. Dette gjør skjema unødvendig så lenge du ikke begrenser hvilke typer dokumenter det er ønskelig å gjøre tilgjengelige.

3.7 Oppsummering state-of-the-art

På generelt grunnlag kan det sies at XML tilbyr mye, men så lenge antallet standarder og muligheter som eksisterer spenner over store områder er det flere som mener at de ikke kan bruke de mulighetene som er der. Hovedårsaken er de begrensninger store standarder legger på hvilke dokumenter de kan bruke teknikkene på siden innholdet kan variere fra tilfelle til tilfelle. Det er derfor nødvendig å sette krav til de data som nyttes for å kunne oppnå gode konsistente resultat.

4 Analyse

Basert på erfaringer ifra eksisterende systemer tilegnet igjennom studien i forrige kapittel og problembeskrivelsen og krav fra avgrensningen i kapittel 1 analyseres forskjellige muligheter for hvordan oppgavens målsetning kan nåes. Hvordan datatypene spesifikt kan angis sees her også nærmere på. Mer spesifikt undersøkes:

- Momenter oppdaget igjennom state-of-the-art.
- En beskrivelse av forutsetninger identifisert for systemet og valg basert på dem.

4.1 Momenter fra eksisterende løsninger

Igjennom studie av eksisterende løsninger kom det frem momenter som er relevant å se nærmere på for å kunne identifisere mulige løsninger relatert til denne oppgaven.

4.1.1 Databaser

Innlesing av XML filer krever store ressurser, dette grunnet at hele dokumenter må gjøres tilgjengelig i minnet før innholdet til en bestemt lokasjon kan ekstraheres. Dette er lite effektivt ved søk der små deler ifra mange dokumenter kan være aktuelle som resultat på en spørring. Bruk av en XML database muliggjør å hente ut data direkte ifra hvilken som helst plass i et dokument uten å måtte lese hele dokumentet først, noe som øker effektiviteten drastisk. XML dokumenter bør derfor legges inn i en XML database slik at enkeltelementer kan hentes ut raskt uansett dokumentstørrelse. Tidligere erfaringer viser at å bruke søkefunksjonalitet i slike databaser er lite effektivt og kan til nød brukes for å begrense utvalget som må rangeres dersom selve søkene er effektive nok.

Tidligere har det vært nevnt at Berkeley DB XML støtter validering av tilhørende XML Schema men at denne informasjonen ikke er tilgjengelig eksternt. Ettersom XML Schema er et av de skjema som er relevant for oppgaven og nødvendig å kunne uthente informasjon ifra, er det nødvendig å kunne få tak i interessant innhold helst igjennom eksisterende systemer for rask implementasjon i en prototyp. Igjennom kontakt med brukerstøtten til Sleepycat som utvikler Berkeley DBXML har jeg fått bekreftet at dette ikke er mulig.

> *BDB XML does not directly expose the schema information, also known as the PSVI (post schema validation info). Also, at this time, BDB XML does not store that information. Documents may be validated, but the type information is lost once the parsing is done.* [78]

Berkeley DB XML har likevel noen fordeler fremfor andre tilsvarende systemer. Blant de fordelene Sleepycat oppgir for å gjøre spørringer mot og navigere i XML dokumenter som gjør den relevant for bruk:

- XQuery 1.0 og XPath 2.0 (April 2005 utgave)
- Spørringer over en eller flere samlinger
- Spørringer over samlinger og nettverkslokasjoner av XML data
- Permanente dokumentidentifikatorer for direkte tilgang
- Optimering av spørringer
- Prekompilerte spørringer inkludert variabler for raskere utføring av spørringer
- Dokumentlesing fra URI, minne eller fil
- DOM-lik navigasjon av XML resultatsett

4.1.2 XML skjema

De to XML skjemastandardene DTD og XML Schema er gjennomgått for å klargjøre hvilken av de som er mest relevante for definering av datatyper. DTD viser seg å ha mangelfull støtte for denne type bruk mens XML Schema har bedre støtte.

For DTD viser det seg at den mangler mulighet til å angi datatyper og krav til innhold. Et av de problemene som kan oppstå er at norske dokumenter vil ha datoer som oftest på formatet DD-MM-YYYY mens amerikanske dokumenter har dato på formatet MM-DD-YYYY. Ved bruk av DTD som skjema er det problematisk å automatisk generere en dato på formen ”20050101” ut i fra slik dato er definert i for eksempel INEX samlingen. XML Schema løser dette problemet ved å ha en standard datotype, men løser ikke problemer med å relatere mange forskjellige datoformater til hverandre på noen direkte måte.

Siden en standard måte å gjøre dette på ikke eksisterer i DTD, fører det til kompliserte løsninger for å tolke denne type data med risiko for store feilmarginer og redusert søkepresisjon. Som eksempel kan INEX samlingen sin DTD og dennes beskrivelse av dato brukes som eksempel på dette:

```
DTD:
<!ELEMENT pdt          ((mo|day)*, yr)>
XML:
<pdt><mo>APRIL-JUNE</mo><yr>2000</yr></pdt>
<pdt><yr>1993.</yr></pdt>
<pdt><mo>Sept.</mo><day>23</day><yr>1897,</yr></pdt>
```

Tabell 14 - XML DTD eksempel på datoproblem

En generell tolkning av slike data er vanskelig å gjøre automatisk og har tidligere krevd at individuelle applikasjoner har måttet inkludere metoder som gjør det på en ad hoc basis. Tolkning av årstall som inneholder ugyldige tegn vanskeliggjør også annet enn ad hoc tolkning av innhold.

Med XML Schema slippes datatypeproblemer som nevnt ovenfor ettersom datatypen i XML Schema krever et vist format, eventuelt at beskrivelsen av formatet som er brukt blir inkludert i XML Schema dokumentet. Dette muliggjør dermed langt bedre presisjon ved søking i slike datatyper siden feil ved indeksering ikke skal forekomme dersom dokumentsamlingen forholder seg til det definerte skjemaets definisjon.

For indeksering av XML basert på XML Schema er det nødvendig å velge ut de datatyper som er ønskelig å indeksere ut i fra hva det er interessant og hensiktsmessig å søke etter. Det kan likevel være problematisk ettersom ikke alle metadataformater har definitive regler for feltinnhold, dette vanskeliggjør bruk av XML Schema med definerte datatyper.

Siden et XML dokument allerede kan være beskrive av et eller flere andre XML Schema og siden et XML dokument bare kan ha et XML Schema per navneområde, er det nødvendig å forme en standard for hvordan prosessoren finner det korrekt XML Schema dokumentet. Dette kan gjøres ut i fra lokasjonen til eventuelt andre definerte skjema som tidligere beskrive i kapittel 2.

> *The presence of these hints does not require the processor to obtain or use the cited schema documents, and the processor is free to use other schemas obtained by any suitable means, or to use no schema at all.* [25]

For å lokalisere ønsket XML Schema kan dette gjøres i et ferdig system ved å legge skjemaet i en egen underkatalog i forhold til der selve XML dokumentet ligger med navn ”XMLindex” og navn ”XMLindex.xsd”. Dette fører til at hver samling XML dokumenter må avskilles i egne kataloger siden hver samling får en ”XMLindex/XMLindex.xsd” fil. Det optimale hadde her vært å inkludert datatyper direkte i metadatastandarder for å automatisk kunne utføre korrekte søk på varierte dokumentformater, men dette krever mer detaljerte standarder og krav til datainnhold for metadatafelt.

XML Schema og mixed content

Undersøking av XML Schema har identifisert situasjoner der det ikke er mulig å beskrive datatyper angitt i elementer definert som inneholdende ”mixed content”. Dette betyr at bare visse XML dokumenter vil kunne annoteres med datatyper igjennom XML Schema dersom ikke andre løsninger nyttes i tillegg.

Årsakene til denne begrensningen i XML Schema er usikker og ikke spesielt godt diskutert så langt jeg har funnet på Internett. Jeanine Lilleng ved IDI/NTNU foreslår at grunnen er at innholdet kan være er ukjent når skjemaet blir generert og det kan derfor ikke begrenses til en datatype. Dersom datatype som blir brukt er kjent trengs ”mixed content” ikke å bli brukt. Teksten imellom underelementene er per definisjon definert som "character data" og kan ikke defineres som noe annet uten å legge de delene av ”character data” som er ønskelig å beskrive mer detaljert inn i et eget underelement.

Trond Aalberg (IDI/NTNU) mener dette er en akseptabel begrunnelse som er naturlig for DTD, men som burde vært støttet i XML Schema. Hans påstand er at så lenge datatyper i XML Schema skal kunne dekke syntaktiske begrensninger er det unaturlig at det eksisterer tilfeller det dette ikke er mulig. En forespørsel til W3C sin e-postliste for

XML Schema vedrørende hvorfor denne begrensningen eksisterer førte til følgende svar ifra Paul V.Biron, en av de som utarbeidet XML Schema standarden:

> *Actually, we tried a number of designs to support this sort of thing but could never reach concensus on how to support the range of use cases. Most of the use cases had i18n implications (i.e., someone want to control the character repertoire of the characters in mixed content), rather than cases like the price above. However, that price example is easier to illustrate the diffiulties, so I'll keep going with it.*
> *Suppose you wanted to say: type the character data as decimal with totalDigits=5 and fractionDigits=2. Then the above would seem to valid.*
> *However, what if you get an instance like:*
> *<price> 345.92 <currency>GBP</currency> 125.00 </price>*
> *Is that valid? One answer is yes, because each sequence of characters in the mixed content is valid according to the type. Another answer is no, because the text() value of <price> is ' 345.92 125.00 ', which is not valid according to the type.*
> *Some people want/need to define the type where the answer would be yes and others want/need no...and we couldn't reach concensus on how to support both (it is not as easy as saying, let the schema author specify with a switch...[..]* [79]

Basert på hva han sier kan det virke som om uenighet grunnet mange forskjellige bruksområder for feltet gjorde at de ikke vart enige for hvordan datatyper skulle kunne defineres for ”mixed content”. Dette er noe som må tas med videre i design av løsningen som forutsetning og begrensning av hva prototypen kan løse.

En mulig identifisert løsning så vidt nevnt av Fuhr et al. i forrige kapittel nevner bruk av xs:annotation støtten i XML Schema for å løse denne type problemer relatert til selve skjema standarden. Eksempelet under viser hvordan xs:annotation kan brukes til å definere forskjellige karakteristikker for et element. Dette kan igjen brukes til å definere datatyper for innholdet av et ”mixed content” element, normalisere datoformat og normalisere tallverdier der det brukes forskjellige desimalseparatorer. Ved bruk av xs:annotation kan det dermed unngåes blant annet mange av de mangler som eksistere med tanke på tolkning av datoer og andre datatyper på forskjellige formater.

```
<xs:complexType name="example1"><xs:annotation><xs:appinfo>
<characterProperties>
<characterSet>UTF-8</characterSet>
</characterProperties>
<numericTextProperties>
<decimalSeparator>.</decimalSeparator>
</numericTextProperties>
<groupProperties>
<fieldSeparator>,</fieldSeparator>
</groupProperties>
</xs:appinfo></xs:annotation>
```

Tabell 15 - Annotation i XML Schema

Støtte for å ekstrahere xs:annotation data er ikke vanlig i mange XML Schema tolkere men lesing av informasjonen kan gjøres siden skjemaet er et vanlig XML dokument.

> *The appinfo child of the annotation element is designed to pass information to a processing application, stylesheet, or other tool. This will be a particular advantage to schema users if XML Schema compliant parsers implement a way of passing this information to an application, [..] By allowing information to be put into the appinfo element, programmers can either pass information to the application about how the section of a conforming document should be processed, or they can add extra code inside the appinfo elements.* [80]

4.1.3 Datatyper

Momenter nevnt i kapittel 2.1 om hva en datatype kan være gjør at flere løsninger der datatyper brukes forskjellig kan vurderes som aktuelle, hver med sine egne definisjoner av en datatype og krav til dets verdier.

Datatyper som er mye brukt innenfor andre standarder relatert til indeksering og søking i informasjon bør foretrekkes som aktuelle løsninger ettersom disse har vist sin nytteverdi på området. Oppfinning av hjulet på nytt er med andre ord ikke nødvendig siden selve informasjonsinnholdet er det samme strukturert og lagret på en annen måte enn hva andre dokumentformat har gjort tidligere. De mulighetene for spesifikasjon av datatyper i XML ved hjelp av XML Schema som er identifisert ut i fra teoridelen og state-of-the-art kapittelet i denne rapporten er:

1. Standard datatyper tilsvarende de i databaser
2. Strukturerte datatyper
3. Semantiske datatyper
4. Identifisering av datatyper brukt i eksisterende XML samlinger

Andre standarder eksisterer og kan være aktuelle men har ikke blitt undersøkt. En gjennomgang av de forskjellige typene nevnt over sammen med hvorfor og hvordan de kan brukes følger.

Standard datatyper

Om datatyper sees på slik de blir brukt i relasjonsdatabaser kan enkle datatyper tilsvarende de i XML Schema brukes. Men siden oppgaven inneholder forutsetninger om at det er ønskelig å kunne bruke forskjellige operatorer på de samme datatypene får en behov for mange forskjellige typer tekst. De forskjellige datatypene, spesielt xs:string må dermed utvides ved bruk av xs:extension. xs:complexType eller xs:simpleType støtten i XML Schema til flere datatyper. Dette tilsier at bruk av XML Schema og tilsvarende enkle datatyper ikke er detaljerte nok for beskrivelse av datatyper forbundet med informasjon. Å se på andre eksisterende datatypesett som er mer passende fremfor å lage egne blir da mer naturlig. I kapittel 2.3 blir det også påpekt at det eksisterer for mange primitive datatyper i XML Schema.

Strukturerte datatyper

Bruk av strukturerte datatyper muliggjør et mer detaljert datatypesett enn vanlige datatyper gjør. Z39.50 sitt BIB-1 strukturattributtsett inneholder datatyper med krav til innholdet som dekker flere av kravene i oppgaven i motsetning til de datatyper som er standard i XML Schema. Årsaker til at strukturattributtene er interessante å se på er at de allerede er godt definerte og dekker de viktigste typene uten å gå inn på altfor semantiske datatyper som raskt kompliserer og fører til høyt antall datatyper.

Bruk av strukturattributter vil ikke muliggjøre nøyaktige søk basert på kontekst igjennom prototypen, men sikrer at en kan få høyere presisjon. Dette siden informasjon som kan forsvinne når blant annet stemming blir brukt ikke forekommer samtidig som begrensninger på eksakthet kan implementeres for noen datatyper.

Semantiske datatyper

Utnyttelse av datatyper med semantisk mening sees på av mange som det optimale for informasjonsgjenfinning siden detaljerte søk blir mulig basert på kontekst. Hovedulempen er at de krever detaljert annotering av tekst for å forstå konteksten og er derfor krevende å utnytte på en god måte.

Z39.50 sitt Bib-1 Use attributtsett inneholder et utfyllende sett typer og en utvidelse av det originale settet inneholder i tillegg Dublin Core sine simple felt. Dette gjør at det ikke er nødvendig å se på Dublin Core sine datatyper som en uavhengig løsning. Datatypene her er likevel mer som metadatafelt å regne og det kan dermed eksistere manglende felt samtidig som antallet felt kan føre til vanskeligheter når typen data det ønskes å søke etter skal velges. Mange av feltene kan sees på som redundante og for nøyaktige, det kan derfor være nødvendig å slå sammen felt. ”Personal name”, ”Corporate name”, ”Conference name” er blant annet relevante å se på som ”name” siden de alle er særnavn der det er naturlig å bruke de samme operatorene for på. Om nok begrensninger som dette gjøres vil settet kunne nærme seg strukturattributtene til BIB-1

I forhold til Bib-1 strukturattributt kan du ved bruk av Use attributter i tillegg gjøre kontekstsensitive søk [49].

MODS sitt XML Schema er fyldig annotert og delt opp etter datatyper noe som gjør bruk av MODS enkelt. En datatype i MODS er angitt som komplekse typer, blant annet nameType og titleInfoType, eller simple undertyper av de igjen som nameTypeAttribute.

Bruk av MODS for spesifisering av et felles sett datatyper for metadata er hensiktsmessig ettersom det dekker over de viktigste metadata. Dette gjør at datatypene angitt i MODS kan brukes til å spesifisere innhold i Dublin Core metadata og i MARC format. Hovedårsaken til at MODS likevel ikke er en god løsning for bruk er at det ikke er støtte for datatyper som er mer vanlige inne i et dokument. Dette gjør at vanlig tekst i et dokument kan ende opp med å bare bli kalt abstract om MODS overføres til fulltekstdokumenter som INEX samlingen. I INEX dekker MODS bare frontmatter og til en viss grad backmatter noe som gjør formatet lite utfyllende, selv om noen datatyper som nameType og locationType kan brukes over hele dokumenter. Bruk av datatyper som passer godt opp mot metadata er likevel relevant siden metadata i stor grad eksisterer som XML.

Identifisering av datatyper basert på eksisterende XML samling
Det er ikke alltid datatyper det er ønskelig å kunne søke etter er de det er mulig å søke på grunnet datagrunnlaget. Vanlige XML samlinger er vanligvis ikke annotert for å tilby søk basert på datatype. En forståelse av hvilke datatyper det derfor er naturlig å kunne uthente ifra eksisterende XML samlinger kan antas å oppdages ved studier av disse.

INEX samlingen er merket opp for å kunne brukes for å eksperimentere med informasjonsgjenfinning i strukturerte dokumenter og det er derfor mulig å kunne anta at de har valgt å legge inn en struktur som er relevant for informasjonsgjenfinning. Dette gjør det aktuelt å se på hvilke datatyper som kan hentes ut ifra strukturen. Siden INEX samlingen både inneholder metadata, fulltekstdokumenter og referanser har den i teorien alle de komponenter som er nødvendige for å generere et fornuftig sett med relevante datatyper.

Dersom det er ønskelig å se på datatyper med høy semantikk er det ikke nok å bare se på INEX samlingen. Dette siden informasjonsinnholdet er homogent og kriterier for hva de har valgt å gi struktur til ikke nødvendigvis passer for andre samlinger. Samtidig er strukturen som tidligere nevnt er påvirket av at opphavet er artikler.

4.1.4 Indeks- og søkemotorer

De fleste søkemotorene for strukturerte dokumenter fokuserer på bruken av struktur og ser ikke på datatyper. Der det gjøres sees det på manuell definering av innholdet. Automatisk utnyttelse av definisjoner igjennom XML Schema er derfor ikke en bruk som har blitt forsket mye på men som noen nevner er en mulighet.

Lucene har tidligere blitt sett på og har blitt brukt tidligere til XML søking, både ved å bruke hele strukturen eller bare deler av den. Forsøk på å koble sammen spørringer som baserer seg på flere deler av et dokument har ikke blitt gjort grunnet de begrensninger Lucene har på dette området. Dette kan gjøres eksternt men Lucene sin indeksoppbygging tillater ikke hurtig å hente ut absolutt alle treff på et søk dersom denne listen er stor.

Feltmuligheten i Lucene fungerer ut i fra de nevnte forsøkene i forrige kapittel fint til å indeksere XML og muliggjør derfor indeksering i Lucene basert på datatype. Lucene sin støtte for generering av egne metoder for analyse av data, formatering, normalisering og endring av spørringer muliggjør også at støtte for nødvendige datatyper kan implementeres.

I et produksjonssystem er det viktig å kunne oppdatere indeksene uten å måtte legge inn alle dokumenter på nytt, spesielt dersom dokumentsamlingen er stor. Hovedårsaken til dette er at dette fører til utilgjengelig system under oppdateringen og ressurs/tidsbruken som er nødvendig for en full oppdatering.

4.1.5 Oppsummering

Gjennomgangen av eksisterende systemer viser at oppgaven har to hovedretninger for hvordan den kan løses. Dette er å basere seg på dataorienterte løsninger i form av databaser, eller informasjonsorienterte i form av egne indekseringsmetoder. Bruk av

XML databaser til et slikt formål er som undersøkelsen viser ikke hensiktsmessig for informasjonsgjenfinning siden det mangler essensielle funksjoner som rangering, prosesseringsmetoder (Stemming, stoppord, omtrentlig søk) og effektive spørrespråk for gjenfinning. Datatypene som er tilgjengelig er også begrenset, selv om kravet om XML Schema fører til strukturert indeksering, og dermed skulle passe godt for databaser. XML Databaser fungerer likevel effektivt som lager for XML dokumenter, og kan godt brukes som et effektivt filsystem.

Kombinasjonssystemer som bruker både informasjonsgjenfinningssystemer og databaser har vist seg å fungere godt ved å ta i bruk hvert system sine fordeler.

Indeksering basert på datatyper, der datatyper er hentet fra Z39.50 og ifra XML Schema viser at det fungerer med manuell annotering og er hensiktsmessig på heterogene samlinger. Dette gjør det interessant å se på hvordan XML Schema kan brukes til indeksering av XML, forutsatt rikt annoterte skjema, noe som ikke alltid er tilfelle. En grunn til å bruke XML Schema er fleksibiliteten og gjenbruksverdien dette formatet tilbyr samtidig som flere angir at det øker indekseringskvaliteten.

For å finne ut hvordan datatyper kan brukes mest mulig effektivt er det nødvendig å se på de forskjellige måtene de kan brukes for å identifisere den beste løsningen. Fuhr og Großjohann [51] viser at datatyper fra informasjonsgjenfinning kan brukes for homogene dokumenter.

En av hovedfordelene med XML er at det kan brukes til å beskrive varierte typer informasjon. Dette er også en av bakdelene siden like typer informasjon sjelden blir annotert likt av to organisasjoner og muligheten for interoperabilitet blir mindre. Behov for en felles standard tilsvarende hva allerede eksisterer for metadata for hvordan en kan beskrive XML dokumenter sin struktur er derfor nødvendig, noe XML Schema har fasiliteter til å gjøre.

Det har flere ganger blitt påpekt at skjemadefinisjoner er lite brukt for XML dokumenter, og for de dokumenter der XML Schema er brukt, er angivelser av datatyper kraftig begrenset. Datatyper og skjema er noe mange ser vekk ifra grunnet begrenset utbredelse og nytte. Dette gjør at metoder og data som kunne blitt brukt til å forbedre søk i XML ikke fokuseres på. Desto mer interessant blir det dermed å utnytte XML Schema og datatyper på en ny måte igjennom en prototyp som kanskje kan forbedre informasjonsgjenfinning i semistrukturerte dokumenter.

4.2 Valgt løsning

Alle valg i dette kapittelet er tatt basert på kunnskap tilegnet underveis i arbeidet på oppgaven samt ønskede krav fra undertegnede. Ingen forutsetninger eller begrensninger er gitt fra noe annet hold utover forslag ifra veileder fortløpende under arbeid på oppgaven.

> *Det overordna målet er å vise at det er mulig å indeksere heterogene XML dokumenter uavhengig av elementnavn. Istedenfor skal informasjonen som beskriver innholdet gis som skjemainformasjon. Å relatere data basert på blant annet datatyper kan være en mulig løsning siden en datatype kan begrense et søke til dokumenter inneholdende en spesifikk type informasjon.*

De overordna valgene gjort for løsningen er:

- Bruk av forskjellige homogene XML samlinger med XML Schema inneholdende datatypedefinisjoner.
- Persistent lagring av XML data som muliggjør direkte tilgang til enkeltelementer i dokumentene.
- Bruk av søkemotor for ustrukturerte dokumenter til søk og indeksering.
- Bruk av relevante datatyper ut i fra ønsket definisjon av datatype.

Hvordan datatyper kan angis i XML Schema er likevel den viktigste delen av den valgte løsningen ut i fra slik målsetningen er utformet.

4.2.1 XML Samlinger

Oppgaven krever tilgang til heterogene XML dokumenter i form av flere samlinger. Det betyr at det er behov for å indeksere XML dokumenter som er basert på forskjellige XML Schema der nødvendige annotasjoner er gjort. Generering av XML Schema må gjøres for hver samling for at dokumentene skal kunne forstås av løsningen.

For at løsningen skal være hensiktsmessig er det et krav at datatilbyder genererer XML Schema for de data som gjøres tilgjengelig. Datatilbyder er den som kjenner en samling best og derfor er best egnet til å generere XML Schema for en samling. Dersom den som har søkesystemet må selv generere XML Schema for alle samlinger forsvinner en av grunntankene bak løsningen: Å tilby en generell modell for hvordan likt innhold i heterogene XML dokumenter kan indekseres korrekt.

For prototypen skal XML Schema lokaliseres igjennom skjema angitt for XML dokumentene, dersom det allerede eksisterer et skjema skal dette løses ved å fjerne eksisterende definisjoner for XML Schema og erstatte med egne.

En av grunnene til at bruk av flere samlinger forutsettes er at siden målet er å vise at datatyper kan spesifiseres i XML Schema for deretter å brukes til indeksering, er det nødvendig med forskjellige XML Schema som viser dette.

Samlingene som brukes må inneholde forskjellige typer informasjon, både fra forskjellige metadatastandarder og fulltekstdokumenter. De må være heterogene slik at prototypen kan brukes til å teste om foreslått løsning fungerer og dekke noen naturlige typer dokumenter som det kan være naturlig å måtte indeksere i et komplett system.

4.2.2 Persistent lagring av fulltekst

For effektiv tilgang til dokumentene må de legges inn i et persistent lager for deretter å kunne hentes ut under indeksering og kunne brukes til å gjenhente fulltekst eller deler av de ellers ved behov.

Det har tidligere blitt vist at bruk av databaser fungerer bedre enn filsystem for å slå opp tilfeldige plasser i en samling XML dokumenter. Bruk av en database for persistent lagring og gjenhenting av data er derfor nødvendig som en del av løsningen.

4.2.3 Bruk av XML Skjema

For denne løsningen er XML Schema er den XML skjemastandarden som er mest hensiktsmessig å nytte av de offisielle W3C standardene for skjema slik målsetningen er formulert. DTD er ikke ment for angivelse av datatyper mens XML Schema muliggjør dette i tillegg til applikasjonsspesifikk informasjon og er derfor et bedre valg. XML Schema skal dermed brukes av løsningen til å annotere nødvendig informasjon om et XML dokument slik at dette kan indekseres korrekt.

For alle felt i et XML dokument der det er naturlig å behandle datainnholdet ut i fra en spesielle datatyper må datatypeinformasjon formidles igjennom XML Schema sin simpletype eller xs:annotation støtte. Simpletype kan brukes dersom det ikke er nødvendig å gjennomføre noen normalisering av innholdet. XS:annotation skal nyttes dersom innholdet er på et annet format enn standardformatet.

Grunnet tidligere nevnte begrensninger i XML Schema relatert til elementer av typen ”mixed content” må XML dokumenter i en XML samling som benytter dette, slik som INEX samlingen, endres for å kunne beskrive datatypeinnhold i ”mixed content” felt. Eventuelt må andre løsninger for hvordan dette feltet skal forstås defineres. Bruk av xs:appinfo støtten i XML Schema kan brukes om det ikke er hensiktsmessig å endre dokumentstrukturen. Bruk av attributter er ikke noen god løsning siden dette ikke blir en del av XML Schema dokumentet som er målet å bruke for oppgaven.

Hvilken type data som kan annoteres i xs:appinfo feltet gjennomgås ikke som en del av valgt løsning. Parametrer som må støttes krever en grundig gjennomgang av eksisterende XML dokumenter for å identifisere dataformater som er nødvendig å kunne normalisere og er knyttet til hver enkelt støttet datatype. En identifisering av noen parametere som må støttes nevnes derfor som en del av testingen ettersom det da kan være behov for å normalisere datainnholdet i de valgte dokumentsamlingene.

```
<eventyear >2006-03-02</eventyear>
```

```
<xs:element name="eventyear" type="date"/>
  <xs:simpleType name="date">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
```

Tabell 16 - XML Schema simpletype i løsningen

```
<årstall >2004-01-02</årstall>
```

```
  <xs:simpleType name="årstall">
  <xs:annotation>
    <xs:appinfo>
     <datatype>date</datatype>
    </xs:appinfo>
  </xs:annotation>
    <xs:restriction base="xs:string"/></xs:simpleType>
```

Tabell 17 - XML Schema annotation i løsningen

Ved å angi både ”eventyear” og ”årstall” som en datatype ”date” muliggjør dette under indeksering å lagre begge datoene i samme indeks slik at de kan søkes etter tiltross for forskjellig struktur. Det blir da mulig å gjennomføre søking spesifikt etter datoer og tilby individuelle søkemetoder på hver enkelt datatype.

4.2.4 Datatyper

Tilgjengelige datatyper må være relevante for informasjonsgjenfinning og ha relasjon til datatyper brukt i andre systemer som det er naturlig å sammenligne med, alt etter hva det er ønskelig å bruke datatypene til.

Formålet med å angi datatyper er å kunne identifisere tilsvarende informasjon imellom heterogene XML dokumenter. De aller fleste XML Schema i dag angir ikke datatyper på et høyere nivå, og datatyper som øker nytteverdien av XML Schema må prioriteres. Målet er likevel ikke å finne et optimalt sett datatyper men å forsøke å vise at de kan brukes på denne måten.

På bakgrunn av fordeler og ulemper ved de forskjellige datatypesettene nevnt i begynnelsen av kapittelet er det mest interessant å se på hvordan Z39.50 sitt BIB-1 strukturattributtsett kan implementeres igjennom en prototyp som deretter kan testes og evalueres. Blant årsakene til at Z39.50 sine strukturattributt er å foretrekke er at de er passe detaljerte samtidig som krav for hvordan de skal brukes allerede eksisterer. Det er heller ikke ønskelig å bevege seg for mye inn på områder som direkte går på ideer om semantisk web, selv om denne metoden nok kunne blitt brukt på et noe begrenset del av dette.

En effektiv samling datatyper som støtter fulltekstdokumenter på samme måten som MODS og Dublin Core støtter metadata er komplisert å lage og ikke interessant å se på før bevis for at det faktisk er mulig ved å lage en prototyp basert på et enklere sett eksisterer. Ved å bruke BIB-1 settet får en utnyttet noen av de mulighetene datatyper gir uten å bli for komplekst for implementasjon i en prototyp. Datatypene må bli brukt på en måte som gjør at bruksområdene angitt i kapittel 1 er mulig.

En liste over de forskjellige strukturattributtene i Z39.50 BIB-1 følger sammen med hvilke metadatafelt som dekkes av de og begrunnelse for hvorfor de kan eller ikke kan brukes. En oversikt hvor forskjellige metadatatyper fra Dublin Core og MODS passer inn er også tatt med for å identifisere hvilke strukturattributt som er nødvendige. Grunnen til at de er tatt med i denne oversikten er for å gi en ide av hvilke datatyper er de grunnleggende for å kunne tilby tilpasset søking i informasjon. Krav til stoppordfjerning, stemming og lignende samt søkeoperatorer er også inkludert. De krav som Z39.50 BIB-1 stiller til bruk har blitt tatt i etterretning og tilpasset andre krav som eksisterer.

Ettersom dette struktursettet ikke inkluderer en datatype for nummer er en numerisk datatype inkludert i tillegg til de som eksisterer i Z39.50. Felt som er urelevante for løsningen er ikke fullstendig utfylt i tabellen under.

Har valgt å fokusere på entydige felt ettersom de lettere kan brukes i prototypen, og ikke grunnet begrensninger på hva som er mulig.

Z39.50 BIB-1	Bruk	Dublin Core	MODS	Krav	Nytte
Phrase	Bestående av distinkte grupper av ord som henger sammen	description	Abstract,	Stoppord Stemming Lowercase Fuzzy	Ja
Word	Bestående av enkeltord, kontrollert vokabular	Subject, type, format, Language, Relation	typeOfResource, genre, language, subject, classification	Lowercase	Ja
Key	Utsnitt av ord				Nei
Year	Årstall			4 tall <> år	Ja
Date (normalized)	Normalisert dato med tid	Date	originInfo	ISO format <> dato	Ja
word list	Liste med ord uten indre orden.				Nei
date (un-normalized)	Dato uten struktur				Nei
Name (normalized)	Navn med gitt orden	Creator, contributor	Name	Etternavn, Fornavn	Ja
name (un-normalized)	Navn uten orden				Nei
Structure	Term med struktur gitt av ytre forhold				Nei
Urx	Termen identifiserer dokumentet	Identifier	Location	URL	Ja
free-form-text	Term skrive inn av bruker				Nei
document-text	Term ekstrahert fra dokument				Nei
local number	Nummer som er signifikant for mottaker				Nei
String	Alle ord er eksakte og sammenhengende.	Title, publisher,	Title	Frasesøk, eksakt	Ja
numeric string	Nummer som skal tolkes som streng.	Identifier	Identifier	ISBN +	Ja
numeric	Nummer, heltall			<> verdi	Ja

Tabell 18 - Analyse av av Z39.50 BIB-1 strukturattributt

Som fordelingen viser er det enkelte attributter som er mer interessante enn andre basert på hvordan metadatatypene har blitt klassifisert. Mest interessant er: Phrase, Word, Year, Date, Name, URX, String, numeric string og numeric. Det er særs vanskelig å definere at noen felt hører til et av disse siden mange felt blant annet i enkel Dublin Core er for vage til å kunne spesifikt definere hvor de hører til uten bruk av kvalifikatorer. Grunnen til at de valgte datatypene ønskes brukt er for å ha et representativt utvalg av typer som det er relevant å søke i, noe det kan antas at de her er siden mange essensielle metadatatyper havner inn under de.

4.2.5 Indeksering av XML

Effektiv tilgang til de data som er nødvendig for å kunne utføre søk gjør at det er behov for systemer som muliggjør rask tilgang til de ønskede data. Valgt system må støtte de behov for søk ved bruk av datatyper som angitt i denne rapporten.

Indeksering av XML dokumentene skal gjøres basert på datatyper angitt i XML Schema for hvert dokument. Prosesseringsrutiner for hver datatype skal angis i applikasjonen siden definering av disse for hvert eneste XML Schema blir mye redundant informasjon, det er heller ikke en god løsning å prosessere to datatyper ifra to forskjellige XML Schema forskjellig. I så fall ville det ført til at begge måter å behandle denne datatypen på må tas i betraktning når et søk skal gjennomføres. Om nødvendig kan likevel informasjonen som skal indekseres forprosesseres for å normalisere innholdet før indeksering gjennomføres. Hvordan dette skal gjøres skal kunne forstås ut i fra XML Schema. Mulighet for å kunne endre hvordan datatyper skal behandles slik at prototypen lett kan støtte andre datatyper er aktuelt for å kunne støtte framtidige endringer. Lokal definisjon er derfor nødvendig.

Bruk av trestruktur er ikke en del av denne oppgaven og det er derfor ikke noe krav at det blir tatt i betraktning ved indeksering eller søk. Kombinering av forskjellige trestrukturer i heterogene samlinger gjør dette uansett vanskelig, noe som også da er målet å slippe ved bruk av datatyper.

Indeksering av alle elementer i en indeks fører til problemer ved normalisering av feltlengden ettersom faktoren som bestemmer normaliseringen vil være definert over alle elementer selv om de er forskjellige entiteter. Dette gjør at et element med lite innhold får langt større betydning enn et felt med mye innhold, selv om alle feltene i det første tilfellet kanskje har samme lengde på innholdet. Normalisering vil dermed føre til at dokument som inneholder kortere treff kommer høyere opp på resultatlisten enn andre. En løsning på dette er å definere en egen indeks for hvert felt. Siden dette i stor grad bare er relatert til avansert søking der det er ønskelig å søke i flere felt med forskjellig normalisering, er det et begrenset problem i denne prototypen.

4.2.6 Spørrespråk og generell søking

Tolkning av spørringer for å identifisere datatype det er ønskelig å søke etter skal ikke implementeres. Dette grunnet at fokuset på oppgaven ikke er lingvistisk tolkning av spørringer. Dersom målet hadde vært hvordan lingvistisk tolkning av spørringer kunne

blitt gjennomført kunne dette også blitt gjort på selve datainnholdet i XML dokumentene. Bruk av XML Schema til å identifisere typer hadde dermed ikke vært relevant siden tolkning av selve innholdet hadde kunne gitt samme informasjon.

For hver enkelt datatype gitt i spørringen blir det lagt til et område rundt gitt søkeord som antas å kunne være relevant som søkeresultat i tilfelle skrivefeil. Spørrespråket som skal brukes for å søke mot indeksert informasjon bestemmes av valgte søkemotor. Siden søkemotoren sine begrensninger derfor er avgjørende i design og utvikling av prototypen er det ikke nødvendig å stille store krav til spørrespråket. Muligheter for å angi spørringer slik at de blir mer tilpasset søk oppimot datatyper må likevel være mulig.

Studien av Lucene viser at løsningen kan nytte en ustrukturert søkemotor som Lucene til å søke i heterogene XML dokumenter slik den foreslåtte løsningen gjør dette.

Tilsvarende med spørringer er ikke søkegrensesnitt en viktig del av oppgaven, gjennomgangen i teoridelen viser likevel hvordan slike systemer i dag ser ut og er derfor veiledende.

4.2.7 Testing

Målet med testingen er å se på om metoden som er implementert fungerer opp imot bruksområdene for ønsket funksjonalitet stilt i kapittel 1. Ut i fra dette kan det deretter sies noe om teknologiene og datagrunnlag støtter foreslått og implementert løsning oppimot gitt målsetning.

5 Prototyp

For å finne ut om valgt løsning faktisk kan gjennomføres og kunne svare på oppgaven må en prototyp av løsningen utvikles. Formålet med en prototyp er å kunne teste ut nye ting, vise at en ide er realiserbar, og kan gjennomføres uten at alle funksjoner nødvendige for et produksjonssystem må implementeres fullt ut. Prototypen kan dermed brukes til å bevise løsningen eller vise at det ikke fungerer uten å måtte bruke for mye ressurser på utviklingen.

Prototypen er implementert basert på valgene gjort i kapittel 4 og systemspesifikasjonen i dette kapittelet. Alle deler av systemet inkludert brukergrensesnittet er implementert for å vise mulighetene foreslått løsning på problemstillingen gir og har utover det begrenset funksjonalitet.

Prototypen er ikke designet med tanke på at løsningen skal være perfekt tilpasset noen spesifikk XML samling, selv om enkelte funksjoner er lagt til for å kunne støtte de samlingene som prototypen er testet på fullt ut. Ettersom løsningen er såpass generell kan den brukes mot andre samlinger og viser fleksibiliteten den innehar.

For å teste om prototypen innfrir de forutsetninger definerte i analysekapittelet og krav i systemspesifikasjonen er en test gjennomført basert på i utgangspunktet antatt støttede data.

5.1 Systemspesifikasjon

Igjennom forutsetninger og valg gjort i forrige kapittel kan det utformes en systemspesifikasjon som angir ønsket funksjonalitet for prototypen som skal utvikles. Systemspesifikasjonen består av følgende deler:

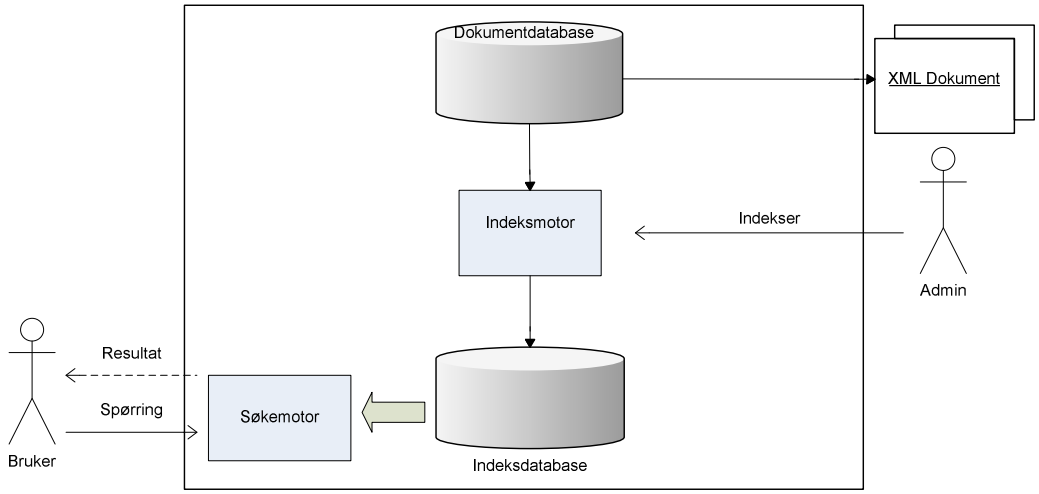
- Funksjonelle krav
- Begrensninger og antakelser

Siden målet med prototypen er å se om en gitt måte å indeksere XML dokumenter på fungerer og ikke skal ende i et fullstendig system, har ikke informasjon om hvordan prototypen kan brukes eller dokumentasjon av denne blitt fokusert nevneverdig på. Tilsvarende gjelder også ikke-funksjonelle krav. Prototypen som skal utvikles kan deles opp i tre deler:

1. Prototypen skal legge XML dokumenter som har korrekt annotert XML Schema inn i en XML database
2. Prototypen skal deretter indeksere de XML dokumenter som finnes i XML databasen. Hvert element skal indekseres basert på datatypen som er angitt for elementet i tilhørende XML Schema. Skjemaet må derfor lokaliseres og deretter prosesseres av systemet. Indekseringsprosessen gjennomføres en gang for en

spesifikk samling data. Indeksen kan deretter brukes så lenge denne samlingen ikke blir modifisert såpass at ny indeksering er nødvendig.

3. Basert på de indekserte data skal systemet igjennom et søkegrensesnitt kunne motta spørringer ifra brukerne og deretter utføre spørringene og returnere svar tilbake. Fulltekst for hvert treff skal hentes ifra det persistente lageret.



Figur 12 - Overordna konseptuell modell for systemet

5.1.1 Funksjonelle krav

De funksjonelle kravene beskriver hva systemet skal gjøre i forskjellige situasjoner basert på systemhendelser, input ifra bruker og hva responsen tilbake skal være. Krav essensielle for å kunne lage en prototyp som svarer til foreslått løsning følger.

Konfigurering

Definering av hva et element i et XML dokument inneholder må utformes av de som lager samlingen ved å angi dette i et XML Schema. Definering av hvordan en datatype gitt i XML Schema skal indekseres er angitt av applikasjonen. Denne konfigureringen brukes til å utføre stemming og andre operasjoner før data indekseres og gjennomføres også på spørringer. Dersom datainnhold som datoer ikke er i standardformat må dette angis slik at indekseren kan lokalisere denne informasjonen og deretter bruke den til å konvertere datoverdier til rett format. Andre konfigureringer er ikke nødvendige ettersom et av målene er å slippe kompleks konfigurering av systemoperator.

Database

Igjennom et grensesnitt skal systemadministrator kunne fortelle systemet at dokumenter lagret i en underkatalog ”XML” skal indekseres av databasen. Databasen skal da kunne motta hele dokumenter og legge disse inn dersom de ikke eksisterer fra før eller har blitt endret. Databasen skal også muliggjøre å returnere både hele dokumenter og enkeltelement basert på bruk av XPath uttrykk på forespørsel fra indekser og søkemotor.

Prosesseringsmetoder
De forskjellige datatypene kan ha behov for å bli prosessert før indeksering for å støtte forskjellige søkekrav, minimere indeksstørrelse og øke gjenfinningsgraden. De forskjellige metodene som prototypen må støtte for å vise mulighetene ved metoden er:

- Stemming
- Fjerne ikke-tekst
- Stoppord
- Muliggjør omtrentlig treff (tall rundt oppgitt tall, navn som ligner litt)
- Normalisere data til rett format

Indeksering
Indekseringsprosessen skal indeksere datatyper forskjellig ut i fra datatype angitt og muliggjøre angivelse av hvilken datatype det skal søkes i. Søk skal da utføres kun over data indeksert for den datatypen.

Siden systemets mål er å teste ut indeksering basert på datatyper trenger ikke systemet å støtte noen former for oppdatering av indekserer. Oppdatering av indeks kan gjøres ved å slette den gamle og lage ny. Dokumenter som skal indeksere må om nødvendig, modifiseres for å kunne bli akseptert av databasen før indeksering kan gjennomføres. Spesielt gjelder dette entitetsfelt og pekere til andre dokumenter.

Første gang et XML dokument med et nytt XML Schema indekseres lages en ordbok, som basert på innholdet av XML Schema inneholder alle unike stier som nøkkel og hvilken datatype som skal brukes som verdi. Dette muliggjør raskt oppslag på hvert element for å bestemme prosesseringsmetoder. Datatypen sendes sammen med datainnholdet til indekseren som lagrer det i sin indeks med datatypen som felt.

Grensesnittet for indekseringsmodulen skal kunne slette eksisterende indekser, legge nye dokumenter inn i datalageret og kunne indeksere disse.

Søkeprosessering
Siden systemet foretar ekstra prosessering av de indekserte data før de blir indeksert basert på datatype må også de samme prosessene gjøres på alle spørringer ut i fra datatypen det skal søkes i for å kunne gjenfinne de samme dokumentene i indeksen. Metoden som utfører denne prosesseringen skal:

- Motta en spørring ifra bruker igjennom brukergrensesnittet
- Transformere spørringen basert på termer i spørringen og dets datatype ved å gjennomføre stoppordfjerning, stemming og lignende basert på datatype oppgitt i spørringen.
- Sende endret spørringen til søkemotoren
- Søkemotoren skal så endre spørringen slik at også omtrentlige treff kan returneres, for eksempel datoer i umiddelbar nærhet.
- Søkemotoren gjennomfører den mottatte spørringen
- Systemet mottar resultat på spørringen fra søkemotoren
- Dersom flere spørringer har blitt gjort må systemet slå sammen disse basert på valgte boolske operatorer i grensesnittet.

- Hente fulltekst for hvert svar, formatere dette og returnere det til brukergrensesnittet.

Søkegrensesnitt
Søkegrensesnittet skal være et simpelt HTML grensesnitt med støtte for søk. Søking skal kunne gjøres med spørringer der datatypen det er ønskelig å søke i kan oppgis. Søkegrensesnittet er ikke essensielt for målet til oppgaven, men er nødvendig for å kunne teste og evaluere løsningen og er derfor en viktig komponent. Kravene til grensesnittet følger:

Overordnet:
- Fungere i en vanlig nettleser som støtter HTML
- Støtte både enkel og avansert søking

Enkelt Søk:
- Standard operatorer tilgjengelig i søkemotoren skal kunne brukes
- Skal kunne søke ved å angi datatype spørringen er på
- Ett XPath uttrykk er et unikt svar
- Simpelt søk returnerer enkeltelementer som svar, kan ikke søke på mer enn en datatype samtidig

Avansert Søk:
- Boolske operatorer som AND, OR, NOT må støttes der det skal kunne angis at forskjellige data skal være/ikke være i ett dokument.
- Avansert søk returnerer ett svar per faktisk XML dokument, muliggjør and/or/and not imellom elementer inne i dokumentet.

Presentasjon:
- Dokumenter som passer med søk rangeres etter søkemotoren sin standardmetode for dette. Noen tilpasning til XML er ikke nødvendig ettersom målet er å vise relativ forskjell mellom å ikke bruke datatyper og å bruke de.

5.1.2 Begrensninger

Begrensninger som påvirker hvordan prototypen skal utvikles inkludert avveininger og områder som ikke fokuseres på.

Programmeringsspråk
For utvikling av prototypen skal Python med Berkeley DB, Berkeley DB XML, XSV, Effbot Elementtree og PyLucene brukes. Bruk av disse applikasjonene muliggjør utvikling av en operativsystemuavhengig prototype. *"(Python is a)..interpreted, interactive, object-oriented programming language. It is often compared to Tcl, Perl, Scheme or Java."* [81].

Python er valgt grunnet de muligheter språket tilbyr for rask prototyping. I tillegg muliggjør bruk av Python å senere bytte ut krevende deler med C++ kode for økt

effektivitet om ønskelig dersom prototypen skal videreutvikles. Undersøkelser viser at design og programmering i Python tar halve tiden av tilsvarende arbeid i C, C++ eller JAVA, og at den ferdige koden bare er halve antallet linjer.

> *The often so-called "scripting languages" Perl, Python, Rexx, and Tcl can be reasonable alternatives to "conventional" languages such as C or C++ even for tasks that need to handle fair amounts of computation and data. Their relative run time and memory consumption overhead will often be acceptable and they may offer significant advantages with respect to programmer productivity [..]* [82]

Persistent XML lager / Database
Berkeley DB XML [83] skal brukes som native XML database [33]. Indekseringsfunksjonene i DB XML skal ikke brukes. XPath / XQuery skal brukes for spørringer mot databasen. Grunner til at Berkeley DB XML er valgt, er god støtte for Python og gratis tilgang i tillegg til at den er enkel å ta i bruk.

Persistent datalager
Berkeley DB ifra Sleepycat, utviklerne av Berkeley DB XML, brukes for enkel lagring av data til disk og grunnet at det er inkludert i Pythondistribusjonen. Data lagres for denne databasen i persistente hashtabeller på disk. Brukes for lagring av data som ikke skal søkes i men som det eksisterer behov for å kunne slå opp direkte.

Indekserings- og søkemotor
Indekserings og søkeløsningen skal basere seg på bruk av PyLucene [53] som har samme funksjonalitet som Lucene. Siden PyLucene ikke direkte støtter informasjonsgjenfinning i XML dokumenter må dokumentene bearbeides for å kunne indekseres og søkes i på en effektiv måte.

For å kunne søke i XML dokumentene så effektivt som mulig og samtidig støtte bruk av rangering av resultatene, må det lages en lokal indeks basert på de dokumenter som skal være søkbare. Siden oppgaven skisserer bruk av flere indekser er det nødvendig å skille de forskjellige datatypene i et dokument ifra hverandre, noe PyLucene skal brukes til. PyLucene indekserer alt som tekst, men muliggjør å skille de forskjellige datatypene fra hverandre enten ved å lage en indeks for hver datatype, eller ved å bruke et felt for hver datatype.

Ettersom PyLucene er ment å bli brukt på ustrukturerte dokumenter med et dokument som en enhet er det nødvendig å definere hva som menes med "ett dokument" i PyLucene i denne oppgaven. PyLucene sin terminologi sees derfor i denne oppgaven på som:

- Indeks
  - Hver installasjon av prototypen har en søkbar indeks
- Dokument
  - Ett element eller attributt i ett XML dokument
  - Hvert dokument i PyLucene kan relateres til et unikt XPath uttrykk
- Felt
  - Datatype, slik det er definert senere i kapittelet

Trestruktur behøves ikke å tas vare på ved indeksering og egne rangeringsalgoritmer for utrekning av vekting og normalisering av dokumentlengde skal ikke utvikles.

Hastighet og sikkerhet
Hurtighet og sikkerhet har ikke noen betydning for denne oppgaven.

Rangering
Kvaliteten på rangeringen av søkeresultatet er ikke en del av løsningen og skal derfor ikke utvikles i prototypen. Tilgjengelige standardmetoder kan derfor brukes.

Brukervennlighet
Ikke viktig i prototypen siden den ikke skal brukes av uerfarne brukere.

Datagrunnlag
I et fungerende system vil kravet til kvaliteten på datagrunnlaget og at grunnlaget er korrekt annotert ut i fra XML Schema være opphavets ansvar. Tilsvarende gjelder også prototypen selv om resultatene da blir basert på egenutviklede XML Schema som kan inneholde feil bruk av datatyper grunnet dårlig kjennskap til samlingenes oppbygging.

Dokumentstørrelse
Tester gjort viser at PyLucene kan krasje om store dokumenter forsøkes indeksert. Det er derfor en begrensning på størrelsen av dokumentene som kan indekseres av prototypen på rundt 1mb. Nøyere forsøk viser at minneproblemer er årsak til dette og kan være relatert til PyLucene og minnelekkasjer i relasjon til XML parser brukt.

5.2 Implementasjon

Implementasjonen er gjort på grunnlag av forutsetninger, krav i systemspesifikasjonen og begrensninger i kapittel 5.1 så langt dette var mulig.

5.2.1 Systemoversikt

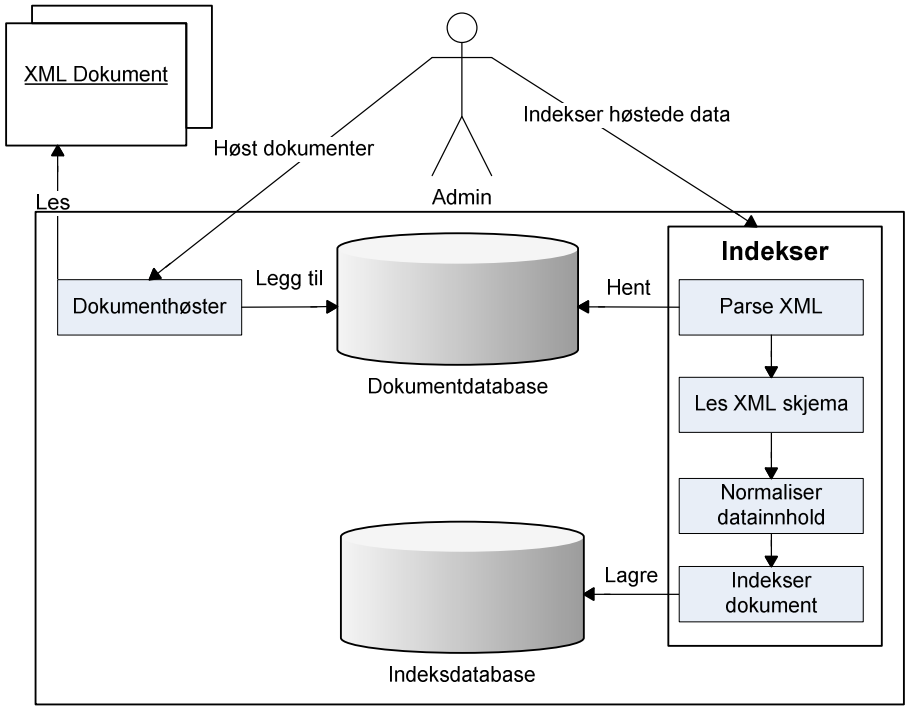
Prototypen er utviklet ved bruk av mest mulig eksisterende moduler / kode for å minimere nødvendig egenprogrammering. En tabell som inneholder hvilke verktøy som er brukt følger:

Område av systemet	**Brukt**
Programmeringsspråk	Python 2.4.3 [81]
XML Schema Validerer	Modifisert versjon av XSV 2.10 [84]
XML Database	Berkeley DB XML 2.2 [83]
Generell database	Berkeley Embedded Database 4.4 [83]
XML parser	Effbot Elementtree 1.2.6 [85]
Indekser / søker	PyLucene 1.9.1[53]
GUI	HTML
Utviklingsverktøy	Textpad 4

Tabell 19 - Bruk av tredjepartsteknologier i prototyp

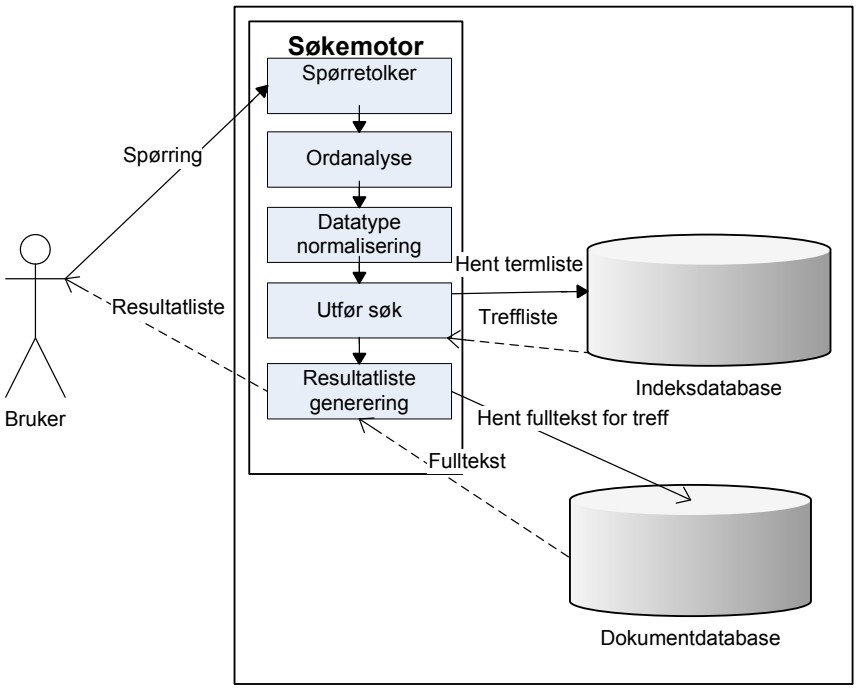
Systemet består av to hovedkomponenter som er en spesifisering av søkemotor og indeksmotor angitt i den konseptuelle modellen. Mer detaljerte diagrammer for de to hoved-komponentene følger under.

Overordna diagram som viser indekseringsmotoren i systemet er angitt under. Diagrammet er basert på systemspesifikasjonen for indeksering tidligere nevnt i dette kapittelet.



Figur 13 - Systemdiagram for indeksmotor i prototypen

Diagrammet under viser søkemotoren i systemet. Det er basert på systemspesifikasjonen tidligere nevnt i dette kapittelet for søking.



Figur 14 - Systemdiagram for søkemotoren i prototypen

5.2.2 Støtte for XML Schema

Datatypene som er spesifisert i kapittel 4 støttes ved at de kan defineres i XML Schema og forstås av prototypen.

Et element kan bare gjelde for ett navneområde, tilsvarende er det også for XML Schema. Linkingen til XML Schema dokumentet har derfor i de tilfeller det allerede eksisterte XML Schema måttet erstatte det eksisterende skjemaet, eller ført til endringer i eksisterende skjema. I et produksjonssystem kan regler for hvor de nødvendige skjema skal eksistere relativt til XML dokumentenes lokasjon angis for å slippe slike problemer som tidligere angitt.

Ifølge standarden er det også de som har valgt å følge et XML Schema sitt ansvar at XML dokumentene faktisk følger skjemaet. Dette gjør at prototypen kan kreve at alle dokumenter inneholder korrekte data. Endringer av kildedokumentene har likevel ikke blitt gjennomført for å kunne trekke fram eventuelle problemer relatert til å bruk av XML Schema. Spesielt gjelder dette tilfeller der dokumenter som har blitt brukt ikke støtter ønskede operasjoner.

For å kunne støtte til en viss grad slike dokumenter har støtte for bruk av xs:annotation i tillegg til datatyper i XML Schema blitt implementert og kan nyttes i skjemaet for samlingene. Bruk av xs:annotation muliggjør å nevne spesifikke operasjoner som må gjennomføres på et element i en samling uansett elementtype. Hvordan

xs:annotation og datatyper har blitt brukt i de forskjellige skjema angis under i kapittelet ”Konfigurering” og viser fleksibiliteten til XML Schema i denne prototypen.

5.2.3 Bruk av XML Database

Prototypen følger systemspesifikasjonen og bruker en XML database ifra Sleepycat som lager for XML dokumenter fremfor lagring av filer i filsystemet. Originalfilene kan etter at de er lagt inn slettes om ønskelig. Innholdet av XML dokumentene kan deretter hentes ut igjennom XPath uttrykk, enten enkeltelementer, attributter og fulltekst av et dokument inkludert strukturelle elementer. Prototypen bruker ikke indekseringsmuligheten databasen skal ha siden denne ikke kan brukes for oppgavens formål.

Alle enkeltfiler i DBXML må ha en identifikator slik at innholdet senere kan gjenfinnes. For å få tilgang til filenes innhold senere må denne identifikatoren lagres for å kunne gjenfinnes ved generering av resultatliste. Denne identifikatoren er for enkelhetsskyld tilsvarende stien der dokumentet eksisterer med alle ’/’ fjernet.

Innholdet i XML databasen aksesseres og leses to ganger av prototypen, en gang ved indeksering av PyLucene og hver gang et søk fører til at fullteksten av et dokument er gitt som en del av et søkeresultat. Data hentes ut ifra Berkeley DBXML ved behov, dette sparer ressurser kontra standardinnstillingen til Berkeley som leser alle data inn i minne før den returnerer resultatliste.

Berkeley DB XML innstillinger

qc.setEvaluationType(XMLQueryContext.Lazy)

- Databasen tilbyr mulighet til å lese alle resultater på et søk med en gang, eller først lese data for hvert enkelt resultat når dette etterspørres. Sistenevnte er brukt i prototypen ettersom minnebehovet da er mindre og prototypen kan levere svar raskere dersom antallet treff er stort.

container.setAllowValidation = True

- Databasen tilbyr mulighet til å validere XML dokumentene dersom XML Schema eksisterer men dette har visst seg å ikke være brukende siden det ikke er mulig å aksessere denne informasjonen i etterkant, databasen sletter informasjonen så raskt den er ferdig med å validere dokumentene. Ved bruk av denne valideringsmuligheten må altså prototypen dermed validere dokumentene en ekstra gang.

mgr = XMLManager(DBXML_ALLOW_EXTERNAL_ACCESS)

- Siden noen XML samlinger bruker stier til eksterne XML DTD og XML Schema må databasen opprettes med denne innstillingen for å gi databasen lov til å hente eksterne data. Krever at stiene angitt for DTD og XML Schema i dokumentene er gyldige.

5.2.4 Konfigurering av XML samling

Konfigurering er her ment som konfigurering i form av det å generere XML Schema for samlinger som skal indekseres av prototypen og ikke konfigurering av selve prototypen.

Prototypen inneholder ikke noen muligheter til å konfigurere oppførsel uten å endre på selve koden. Å gjøre dette er likevel enkelt grunnet at Python scripting er brukt og oppdeling av programmet i moduler gjør det lett å bytte ut enkeltdeler. Prototypen er derfor avhengig av at en datasamling er korrekt annotert, eventuelt uttrykker hvilket annet format datainnholdet er på. På grunn av dette er det ideelt sett dataleverandør sitt ansvar at data er på rett format.

Konfigureringen gjøres av de som lager en samling ved å bruke definisjonene for datatyper som angitt i forrige kapittel, og konfigurere "sin" samling ved å lage et XML Schema for denne som kan forstås av prototypen. Der datainnholdet i XML samlingen viker ifra datagrunnlaget prototypen slik krever det, har den noen muligheter for å konfigurere dette. Innholdet kan da normaliseres til et felles format.

Krav til bruk av datatyper

For best resultat bør hver datatype ha samme format uansett hvilken samling de er brukt i. Prototypen sjekker utenom for dato ikke at innholdet faktisk er i det angitte dataformatet. Ideelt datainnhold for hver datatype ved indeksering er:

Datatype	**Bruk**
Phrase	Lengre tekst, paragrafer o.l
Word	Nøkkelord, roller
Year	Årstall
Date (normalized)	Dato YYYYMMDD
Name (normalized)	Etternavn, Fornavn
Urx	HTTP link
String	Korte enkeltsetninger, tittel
numeric string	Alphamumeriske identifikatorer. Nummer som skal tolkes som streng. ISBN number f.eks
numeric	Nummer

Tabell 20 - Krav til datatyper i prototypen

Under er noen av de mulighetene prototypen har til å sikre seg at data er på rett format. Ikke alle datatypene er implementert men eksemplene under viser mulighetene og andre typer kan bruke tilsvarende løsninger.

Konfigurering ved bruk av XML Schema datatype

Definering av datatyper følger slik XML Schema standarden definerer dette og kreves derfor av prototypen angitt som i følgende eksempel. Restriksjoner på datatype er ikke brukt av indekseren men er nødvendig for å få godtatt datainnholdet av XML Schema validereren. På grunn av dette er den her satt til xs:string selv om et årstall er et heltall.

```
<xs:element name="årstall" type="year"/>
 <xs:simpleType name="year">
  <xs:restriction base="xs:string"/>
 </xs:simpleType>
```

Tabell 21 - Konfigurering av simple datatype i XML Schema

I tilfeller der et element er beskrive som ”mixed content” kan ikke innholdet spesifiseres ved hjelp av simple types i XML Schema, det kreves da av indekseren at datatype er angitt i xs:annotation elementet. Bruk av annoteringsmetoden kan også brukes i andre tilfeller der ovenstående metode fungerer og gjør dermed bruk av simpletype overflødig og mer et alternativ.

```
 <xs:annotation>
  <xs:appinfo>
   <datatype>date</datatype>
  </xs:appinfo>
 </xs:annotation>
```

Tabell 22 - Konfigurering av annotation datatype i XML Schema

Konfigurering ved bruk av XML Schema annotation

Indeksering av datoer der datoen eksisterer i ett felt men kan ha et annet format enn hva som er standard har blitt løst ved å angi format igjennom annotasjonsfeltet. Innholdet i dette feltet må være basert på de operatorer som Python støtter for slik bruk. Formatfeltet kan også brukes for å angi formatet til datainnhold i andre felt, men i prototypen er det bare implementert brukt av dato som eksempel. I eksempelet kan en dato på formatet ”1999.12” forstås av prototypen der 12 er måned.

```
 <xs:simpleType name="date">
 <xs:annotation>
  <xs:appinfo>
   <format>%Y.%m</format>
 </xs:appinfo>
 </xs:annotation>
  <xs:restriction base="xs:string"/>
 </xs:simpleType>
```

Tabell 23 - Normalisering av datatype annotation

For tilfeller der data er delt opp i to underelementer og det er ønskelig å sette disse sammen til ett felt kan tilsvarende gjøres og deretter definere datatype. Indekseren kan dermed indeksere innholdet korrekt selv om de er strukturelt atskilt i den opprinnelige samlingen. I eksempelet her blir innholdet av elementene snm og fnm i et mixed content element slått sammen til ett element som deretter kan indekseres som en datatype fremfor to.

```
<xs:annotation>
 <xs:appinfo>
  <datatype>name</datatype>
  <concenate>snm|fnm</concenate>
 </xs:appinfo>
</xs:annotation>
```

Tabell 24 - Sammenslåing av felt i mixed content

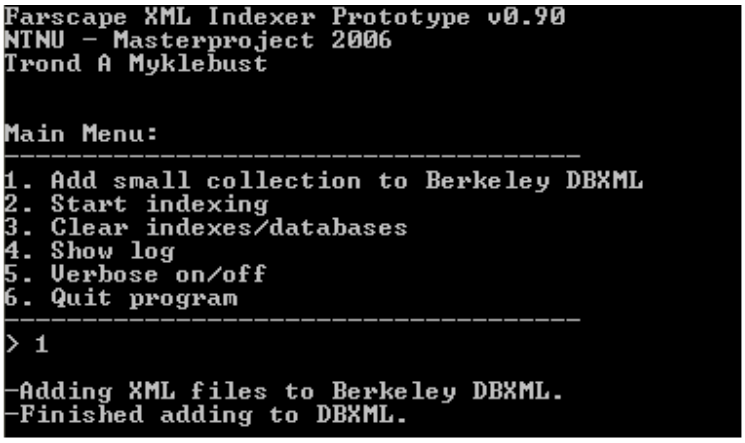
Tilsvarende kan gjøres for elementer som ikke er ønskelige å indeksere som egne datatyper, men der innholdet likevel er interessant som en del av et annet element. Elementer som ikke skal indekseres kan angis med mellomrom imellom hvert enkelt. Elementene blir da fjernet slik at innholdet i disse blir en del av foreldreelementet.

```
<xs:annotation>
  <xs:appinfo>
  <remove>scp b it h</remove>
 <datatype>string</datatype>
 </xs:appinfo>
</xs:annotation>
```

Tabell 25 - Fjerning av uønskede elementnavn

5.2.5 Indeksering av dokumenter

Indekseringen gjøres i prototypen ved å hente ut dokumenter enkeltvis ifra databasen. Deretter valideres XML Schema dokumentet og det hentes ut annotert informasjon, for deretter å gå igjennom alle XPath stier for hvert dokument og indeksere innholdet av de i en PyLucene indeks.



Figur 15 - Prototype indekser GUI

Indekseren i prototypen gir brukeren noen valg igjennom et tekstbasert grensesnitt. Grensesnittet er låst mens arbeid blir gjort. Prototypen gjør ikke arbeidet i en egen tråd men bruker samme som grensesnittet for indeksering. Årsaken til dette er at det ikke er

behov for å kunne gjøre flere ting samtidig i prototypen. Mulige handlinger som kan gjøres i indekseren implementert i prototypen er:

1. Legger XML dokumenter eksisterende i underkatalogen ”XML” inn i databasen
2. Indekserer XML dokumentene som er i databasen og gjør de søkbare
3. Sletter alle indekser og databaser som har blitt opprettet
4. Skriv hele arbeidsloggen til skjerm
5. Skriver arbeidslogg til skjerm etter hvert som noe blir gjort
6. Stenger alle databasetilkoblinger og avslutter programmet

Lesing og behandling av XML dokumenter

Indekseringen foregår ved at en tråd leser XML elementer ifra databasen, behandler de og legger de i en kø for innlegging i PyLucene. En annen tråd venter til køen har x elementer før tråden sender data til PyLucene. I tillegg til selve dokumentinnholdet er det nødvendig å indeksere ekstra informasjon om hvert element for å kunne gjenfinne fullteksten senere, dette grunnet at fullteksten ikke lagres sammen med PyLucene.

1. En spørring mot DBXML gjennomføres som returnerer alt innhold: collection('"databasepath"')
2. For hvert unike dokument i resultatsettet:
    a. Tolkes dokumentet og XML Schema valideringsinformasjon hentes ut
    b. Sjekk om det eksisterer elementer som skal fjernes på bakgrunn av valideringsinformasjon, i så fall, fjern de og tolk dokumentet pånytt.
3. For hvert element i hvert XML dokument
    a. Hent ut all tekst og attributter for elementet
    b. Sjekk om det eksisterer xs:annotation og datatypeinformasjon
    c. Utfør operasjoner nevnt i applikasjonsdelen av xs:annotation
    d. Legg elementet med data og opprinnelig lokasjon i form av et XPath uttrykk i en kø for innleggelse i PyLucene

Prosessering av XML Schema

For å validere og hente ut informasjonen fra skjemaene brukes en modifisert utgave av XSV fra W3C. Denne er modifisert for å støtte returnering av XML Schema informasjon tilbake til prototypen ettersom originalen ikke støttet dette. Til tross for endringene er ikke den opphavlige funksjonaliteten påvirket.

Årsakene til at XSV vart valgt for dette ligger i at det ikke vart funnet noen eksisterende skjematolker som hadde ønsket funksjonalitet samtidig som at XSV hadde kildekoden tilgjengelig og var utviklet i Python. Dette gjorde det lett å tolke kildekoden og endre denne til ønsket formål.

XSV er endret fra å kun returnere om skjemaet vart validert eller ikke til å sende informasjonen, kalt PSVI [86] data, brukt til å validere dokumentet tilbake til prototypen. Denne informasjonen vart tidligere ikke tatt vare på etter at dokumentet var ferdig validert, endringene som har blitt gjort samler derfor sammen ønsket informasjon under validering og returnerer denne til applikasjonen som stod for kallet.

Dersom skjema ikke er tilgjengelig indekserer prototypen innholdet uten bruk av datatyper. Ved feil under validering blir dette gitt beskjed om av indekseren som deretter indekserer uten bruk av XML Schema. Den endrede versjonen av XSV returnerer mer spesifikt:

- Antall feil som vart funnet under validering.
- Eventuell datatype / simpletype for hvert felt.
- Eventuell xs:annotation informasjon for hvert felt.
- Eventuell komplekstype elementet er definert som.
- En sti over komplekse typer i tilfelle de er sammensatt.

Normalisering

Dersom ekstrahert informasjon fra XML Schema dokumentet inneholder annoteringsdata kan dette brukes av prototypen til å normalisere datainnholdet. Normaliseringen muliggjør at det blant annet kan søkes i europeiske og amerikanske datoer uten problem. Normaliseringen gjøres utenfor PyLucene siden PyLucene ikke har standardmetoder som tillater å sende ekstra informasjon utover datatypenavn og datainnhold til PyLucene.

En av normaliseringsmetodene som er implementert i prototypen er normalisering av dato, basert på at dette er angitt ved bruk av xs:annotation under konfigureringen slik nevnt tidligere. Prototypen normaliserer datoen ifra XML dokumentet og tolker denne igjennom formatet angitt ved bruk av datofunksjonene i Python. Datoen blir deretter skrive ut på standardformat ”YYYYMMDD” som sendes til PyLucene for indeksering. Hovedproblemet er at datatypen må være angitt på samme format i alle like elementer i all dokumenter i en homogen samling, dersom dataformatet varierer er ikke dette mulig å vite for prototypen som da ikke vil kunne normalisere datoen.

Normaliseringsregler for alle datatyper er ikke blitt lagt inn i prototypen, men eksempelet med datoformat viser mulighetene som eksisterer.

Indeksering av PyLucene

Lucene lagrer alt internt som tekst, dette gjør at prototypen må gjennomføre noen endringer ved indeksering av andre datatyper siden for eksempel tekst og nummer har forskjellige sammenligningsmetoder.

Noen endringer av data som gjøres og begrensninger i prototypen sees i tabellen under. Felles for alle er at formålet er å muliggjøre korrekt og bedre søk for hver datatype. De metodene nevnt er de som er lagt inn i prototypen for å vise mulighetene det å skille innhold på datatyper gir, og er ikke nødvendigvis de beste.

Datatype	**Filtreringsmetoder**
Phrase	Stemming, stoppord, fjerne tegn, lowercase, fuzzy
Word	Lowercase
Year	Årstall, padde nummer, muliggjøre vag spørring
Date (normalized)	Dato = 20060303 muliggjøre vag spørring

Name (normalized)	Fjerne komma, lowercase
Urx	Ta vare på alle tegn
String	Fjerne alle non-alphanumeric tegn, Lowercase
numeric string	Lagres som de er, med alle tegn uten å deles opp
Numeric	Padde nummer, positive, negative 10 == ”00010” muliggjør liket med ”10000”. Negative må påplusses slik at laveste negative som er mulig blir 0.

Tabell 26 - Krav til datatyper i PyLucene

For hvert dokument, eller XPath uttrykk basert på strukturen i et XML dokument, indekserer PyLucene en trelokasjon, dokument id, sti id, datainnhold med og uten case og et felt basert på datatype. Fulltekstdata kunne også vært lagret i PyLucene sin indeks men hadde ført til stor indeks grunnet dobbel lagring av datainnholdet i ett dokument og problemer med gjenhenting av hele dokumenter.

Trelokasjon

For å spare plass når antallet elementer blir stort har en plassbesparende strategi blitt brukt for å vise til lokasjonen i et dokument tre. Fremfor å lagre hele stien i PyLucene lagres den som nummer. Eksempelet kan tolkes som at lokasjonen i treet er det fjerde underelementet av det andre underelementet til det første underelementet av rotnoden. Begrenset nytte i denne prototypen men er i bruk ved avansert søk. Eksempel:

Rot[0]/data[0]/info[1]/element[3] -> 1 2 4

Dokument ID

Fremfor å lagre id til et dokument i PyLucene lagres bare en identifikator som peker på et eksternt Berkeley DB lager der det fulle navnet ligger. Hovedgrunnen til at det er en fordel å gjøre dette er måten elementer blir lagret i PyLucene som fører dette til at dokument id må lagres en gang per element, noe som fører til at å lagre hele dokument identifikatorer raskt tar stor plass.

Sti id

Tilsvarende som for dokumentidentifikatorer, er det nødvendig med en identifikator som angir en spesifikk XPath. Ved indeksering av mange homogene dokumenter er det naturlig at det eksisterer mange like XPath uttrykk for hvert dokument. Dette kan utnyttes tilsvarende som for dokumentidentifikatoren ved å bytte ut med en nummeridentifikator.

Innhold med case

Ettersom det kan være ønskelig å kunne søke uten å angi datatype indekseres alt uavhengig av datatype også inn i et ”content” felt der store og små bokstaver tas vare på. I tillegg indekseres alt i et ”contentL” felt der alt er i små bokstaver. Dette muliggjør å sammenligne søkeresultat ved bruk av datatyper versus å ikke bruke de om ønskelig.

Basert på datatype

Der datatyper er angitt blir innholdet indeksert i et felt i PyLucene med samme navn.

Berkeley DB
I Berkeley DB lagres forskjellige typer data i en hashtabell for å effektivisere søk og gjenhenting i PyLucene som angitt over.

- En struktur har en sti id som nøkkel og XPath uttrykk som verdi
- En struktur har XPath uttrykk som nøkkel og id som verdi
- En struktur som muliggjør oversettelse av dokument ID i PyLucene til dokument ID i Berkeley DBXML
- En komprimert tre struktur som kan traverseres og plottes treff ut i fra trelokasjonen angitt i PyLucene.

5.2.6 Dokumentsøking

For å støtte søk i de indekserte data har prototypen et enkelt HTML grensesnitt som muliggjør å lage spørringer, be om å få de gjennomført og motta resultat.

Spørresyntaksen som brukes er den samme som PyLucene støtter siden den er brukt som søkemotor. Ettersom PyLucene er lett å utvide med egne metoder har dette blitt benyttet for å tilpasse PyLucene bedre til de krav prototypen har.

De samme metodene som utføres på indeksert data basert på datatype gjøres også på ord i spørringen for at det skal være mulig å få treff på disse i indeksen. Søkemotoren behandler deretter spørringen i en egen tolker som ut i fra datatype bestemmer om det er ønskelig med eksakt eller omtrentlige treff på spørringen.

Det var ønskelig at prototypen skulle muliggjøre vage spørringer over dokumentsamlingen også på nummer og få resultater tilbake selv om presise resultater ikke vart funnet. Dette støttes av prototypen grunnet bruk av PyLucene og gjør at prototypen kan transformere en spørring basert datatypen det søkes i. Vage spørringer er implementert i prototypen ved at det ved søk etter numeric og year inkluderes > X% < område rundt opprinnelig søkeverdi. For frasesøk utnyttes fuzzyoperatoren i PyLucene sammen med stemming for å finne treff nær opprinnelig søketerm.

Prototypen støtter søk i feltene angitt i kapittel 4 under design av systemet. Søk utføres i et felt ved å angi hvilket felt det er ønskelig å søke i etterfulgt av søkeord; ”Felt:spørring” eller ”year:1950”.

Behandling av søk

Ved mottatt spørring prosesseres søket altså på følgende måte av prototypen og PyLucene:

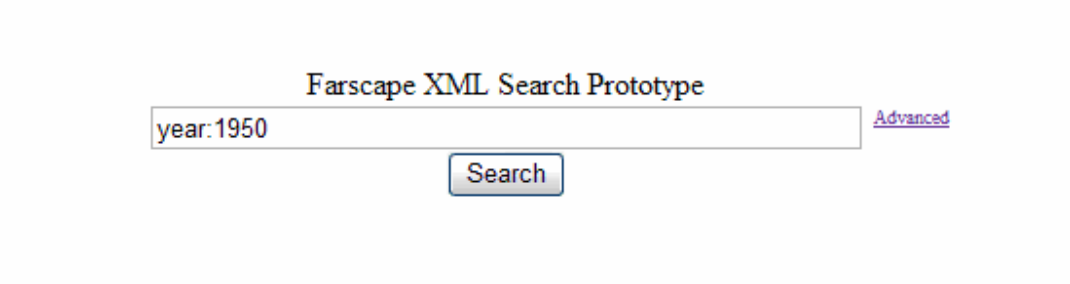
1. Mottar spørring fra bruker igjennom grensesnittet.
2. Parser spørringen til PyLucene format
3. Konverterer og normaliserer innholdet i spørringen basert på gitt datatype.
4. Endrer spørring til omtrentlig ut i fra gitt datatype.
5. Utfører spørringen.
6. Returnerer svarsett til bruker.

Ettersom et søk kan bestå av flere spørringer går prototypen deretter igjennom hvert mottatt svarsett for å generere en rangert liste som returneres til bruker:

1. Dersom flere spørringer, gå igjennom resultatsettene basert på boolske operatorer brukt imellom de og legg sammen vektingen for hvert element slik at en liste med dokumenter er igjen. Sorter så gjenstående liste etter vekting.
2. Gå igjennom den sorterte listen, for hvert enkelt resultat hent ut hvilket dokument og XPath'en til resultatet. Hent så fulltekst fra Berkeley DBXML og generer resultatlisten som deretter sendes til bruker.
3. Bruker kan så velge å gjøre nye søk, eller se fullteksten av dokumentene hvert enkelt resultat stammer fra.

Enkelt søk

Prototypen muliggjør søk basert på enkelttyper der datatype sammen med verdi angis i et tekstfelt. Krav til søkeordene er de samme som for datatypene under indeksering. Dersom ikke datatype oppgis søkes det automatisk i alle indekserte data uansett datatype.



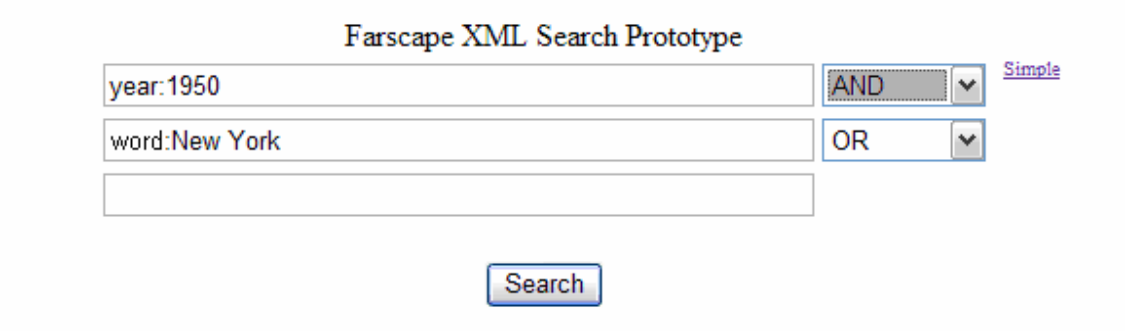
Figur 16 - Enkelt søkegrensesnitt i prototypen

Ved enkelt søk returneres treff på en spørring der hvert treff er det samme som en unik XPath spørring mot databasen. Søk på flere datatyper med AND vil her ikke returnere svar dersom flere datatyper søkes på siden hver XPath / indeksert element bare kan være av en datatype.

Avansert søk

Dersom ønskelig kan et avansert grensesnitt vises med flere muligheter enn hva et enkelt søk tilbyr. Ved avansert søk returneres treff på en spørring der hvert treff er det samme som en unikt indeksert fil. AND/OR/AND NOT muliggjør å søke på flere felt innenfor ett dokument. Resultatlisten er sortert basert på den totale vektingen av alle elementer gjenfunnet for hvert dokument.

Bilde frå avansert søk etter dokumenter ifra rundt 1950 om New York:



Figur 17 - Avansert søkegrensesnitt for prototypen

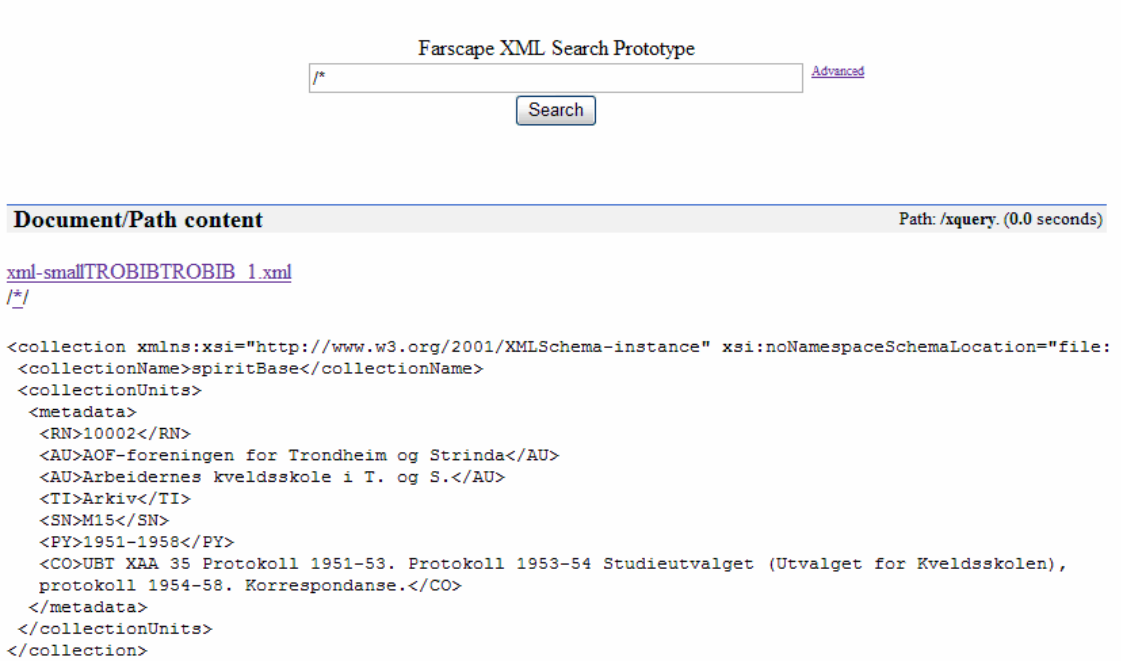
Søkeresultat

Et søkeresultat etter et enkelt søk returnerer en liste over treff med angivelse av dokument treffet var i og nøyaktig XPath for treffet sammen med fulltekst for dette feltet. Listen er rangert etter relevans. Både attributter, elementer og hele dokumenter er enheter som kan forekomme i resultatlisten, alt etter om det er enkelt eller avansert søk. De forskjellige delene av resultatgrensesnittet er:

- Antall treff av antall mulige og tid søket tok
- For hvert treff:
  - Dokumentnavn
  - Poengsum
  - XPath for treffet
  - Datainnhold i elementet, dokumentet eller attributtet

Figur 18 - Søkeresultat i prototypen

Søket over viser et søk etter 1950 som årstall og returnerer treff som er i området rundt ønsket årstall. Denne listen er ikke rangert etter årstall nærmest ønsket søk, dette siden søket blir omgjort til et omtrentlig søk, noe som gjør at PyLucene ikke ser forskjell på om 1945 eller 1951 er mer relevant for 1950 eller ikke.



Figur 19 - Dokumentvisning i prototypen

Ved å velge et treff fås dokumentets innhold opp slik det var i den indekserte filen. Dokumentet presenteres da med XML struktur og innhold og er ikke brukervennlig. Dette kan løses i fremtidig versjon ved å kreve at dokumentene skal ha et XSL Stylesheet som forteller hvordan dokumentene skal vises.

5.3 Testing av prototyp

For å teste om prototypen innfrir de krav stilt til dens oppførsel er en test gjennomført basert på en samling av i utgangspunktet antatt støttede data. Hver enkelt del av systemet har blitt testet i den rekkefølgen prosessene må gjøres for å få et fungerende system. Før kapittelet avsluttes oppsummeres erfaringer gjort under testingen av prototypen.

Hva skal testes

Prototypen skal testes for å se at den kan håndtere og indeksere XML dokumenter slik at de blir søkbare ut i fra valgene i kapittel 4 og systemspesifikasjonen. Fra hver valgte samling har det blitt indeksert ti dokumenter som er representativt for samlingen. Flere dokumenter har ikke blitt valgt fra hver samling siden ti dokumenter er nok til å kunne vise at prototypen fungerer ut i fra definert kravspesifikasjon. De ønskede datasamlingene er som tidligere nevnt valgt på basis av hva som var tilgjengelig, og er ikke nødvendigvis

optimale for bruk mot skissert løsning implementert igjennom prototypen. De tilsvarer likevel datasamlinger som det er normalt å skulle ønske å søke i.

Verifiseringen av at prototypen håndterer data og indekseringen korrekt kan et søk på datatyper bekrefte ved å se om resultatet stemmer med forventet resultat.

Hva skal ikke testes

Brukervennlighet, ytelse og kvaliteten på prosessene som prototypen utfører skal ikke testes. Grunner til dette er at prototypen ikke implementerer fullt ut delene av et komplett system som gjør at statistisk materiale kunne vært en god indikator på kvaliteten til prototypen. Prototypen fokuserer på hvordan informasjon som kan angis igjennom en XML standard kan brukes til å forbedre indekseringen av et XML dokuments innhold, og ikke direkte på hvordan søkeresultat kan forbedres. Det endelige målet er likevel bedre søkeresultat.

5.3.1 Kodetesting

De forskjellige modulene som prototypen er basert på har underveis i utviklingen blitt utprøvd på en testsamling for å verifisere at ønsket funksjonalitet fungerer. Siden prototypen er utviklet i et scriptspråk som ikke kompileres har alle deler av systemet måttet blitt testet siden noen feil ikke oppstår før koden faktisk blir kjørt. Dette er spesielt viktig i Python som ikke støtter definering på forhånd hvilken datatype en variabel kan inneholde. Syntaksfeil og feil under kjøring av prototypen er altså blitt fikset underveis men kan likevel ikke ansees som ikke-eksisterende siden det er avhengig av de data systemet mottar. Koden fungerer derfor for de definerte samlingene dersom de samme tredjepartskomponentene tas i bruk.

5.3.2 Datagrunnlag

De dokumentsamlingene som prototypen skal testes på er valgt på bakgrunn av hva som var tilgjengelig og ikke hvilke samlinger som nødvendigvis er mest hensiktsmessige å bruke mot systemet slik målsetningen og dette kapittelet definerer dette som.

Årsaken til at de fleste inneholder metadata er god tilgang til denne type dokumenter samtidig som de inneholder relevante datatyper for oppgaven. Selv om mange av samlingene virker like har de forskjellig struktur og forskjellige elementnavn som gjør de relevante for bruk som heterogene samlinger. Siden bruk av struktur ikke er en del av foreslått løsning er det nødvendig å dele opp dokumentene i enkeltenheter ut i fra hva som kan defineres som en enhet i de forskjellige samlingene.

OAI Dublin Core høstet fra CERN ved bruk av OAI-PMH (metadata)

Samling på 1000 dokumenter hentet ifra CERN ved bruk av OAI-PMH metadatahøster utviklet av undertegnede i faget Digitale Bibliotek. Denne samlingen bruker Dublin Core som metadataformat.

INEX TD/1998 (fulltekst)
INEX samlingen har i tidligere kapitler blitt gjennomgått. I denne oppgaven brukes td/1998 av samlingen slik at raskere testing kan gjennomføres på en mindre samling. Denne samlingen inneholder bruk av "mixed content" elementer og blir rammet av problemene ved å beskrive innholdet av slike elementer i XML Schema. Spesielle forholdsregler må derfor tas ved indeksering av denne samlingen kontra de andre.

KPRO15 (metadata)
Metadataposter med beskrivelse av flyfoto fra Trondheimsområdet ifra Falkanger Widerøe. Inneholder felt om forfatter, dato, sted, klassifikasjonsdata og beskrivelse blant annet.

Spirit (metadata)
Metadatasamling av lyddokumenter inneholdende metadataposter om kilde, emne, medvirkende, ansvarlig, publikasjonsdata, beskrivelse og annet.

Topbase (metadata)
Samling av metadata ifra Universitetsbiblioteket i Trondheim inneholdende metadataposter om flyfoto som sted, dato, klassifikasjon og eksemplardata.

Trobib (metadata)
Samling av metadata. Trobib [87] omfatter litteratur om Trondheim og dekker forskjellige typer materiale: bøker, manuskript, kart, avis- og tidsskriftartikler. Inneholder metadata om lokale publikasjoner og data som tittel, forfatter og årstall.

5.3.3 Tilpasning av datagrunnlag

For å ha datagrunnlag for å kunne teste ut prototypen og undersøke de definerte målene var det nødvendig å lage et XML Schema for hver samling tilpasset de datatyper som prototypen skulle testes på. I noen tilfeller var det også nødvendig å endre på innholdet i XML dokumentene for å få testet skikkelig, dette grunnet som tidligere nevnt stor variasjon i innholdet til enkelte dokumenter og begrensninger i forhold til enkelte komponenter brukt av prototypen. En gjennomgang av de forskjellige samlingene som er testet følger sammen med en beskrivelse av hvordan de er tilpasset bruk mot prototypen og hvordan prototypen håndterer samlingene. De genererte XML Schema for hver samling er lagt ved rapporten på CD og nærmere nevnt i appendiksene til slutt i rapporten.

INEX TD/1998
INEX samlingen har et komplekst sett elementer der mange er av "mixed content" type definert i en DTD og er derfor tidkrevende å manuelt generere et XML Schema for. Skjemaet vart derfor generert ved å konvertert DTD til XML Schema ved bruk av "Syntext DTD2XS". Det genererte skjemaet inneholdt mye overflødig annotasjoner men var likevel brukbart, det måtte likevel tilpasses bruk oppimot prototypen siden datatyper ikke var definert i dette skjemaet.

I noen tilfeller var det vanskelig å definere datatyper grunnet at dokumentinnholdet ikke var godt nok annotert. I eksempelet under kunne Ian Foster vært indeksert som navn, men siden navnet ikke er i et eget element er ikke dette mulig ved bruk av metoden prototypen nytter.

```
<ip1><b>Ian Foster</b> received his PhD in computer science from Imperial College, London.</ip1>
```

Datoer er en annen datatype som var vanskelig å ekstrahere fra INEX siden denne er delt inn i tre forskjellige underelement av et ”mixed content” element. I tillegg inneholder feltet variasjoner mellom tidsperioder og spesifikke datoer, noe som vanskeliggjør indeksering. En såpass stor variasjon umuliggjør generiske metoder for indeksering ut i fra datatype. Datainnholdet i et ”mixed content” element kan likevel indekseres basert på datatype ved bruk av xs:annotation.

INEX samlinga inneholder i tillegg mange elementer det ikke er ønskelig å indeksere separat, som blant annet ”<b>” i eksempelet over, disse elementene må derfor fjernes og fører til dobbel tolkning av XML Schema tolkeren siden disse taggene først må identifiseres og fjernes og deretter tolkes på nytt.

Samlinga vart også delt opp i enkeltdokumenter der ett dokument er en artikkel for å ha en felles målestokk for alle samlingene.

KPRO15

Som med INEX samlingen vart KPRO15 delt opp i enkeltdokumenter. I tillegg måtte ø, æ og å fjernes fra elementnavn siden XML Schema tolkeren ikke taklet disse tegnene. Gård, Gårdsnavn og gårdsnummer vart derfor gard, gardsnavn og gardsnummer.

KPRO15 skapte også problemer med datoer siden det er forskjellige formater ute og går for innholdet. I noen tilfeller er det eksakte datoer og i andre omtrentlige med forskjellig skrivemåte: ”<datering>1936-1949</datering>” og ”<datering>1936-08</datering>”.

Spirit

Spiritsamlingen vart også delt opp i enkeltdokumenter der ett dokument var en metadatapost. Denne samlingen inneholder feiltagginger som skaper feilindeksering dersom innholdet indekseres ut i fra datatype. I eksempelet under ville da 1947 ikke vært mulig å gjenfinne som årstall ved å bruke implementert indekseringsmetode.

```
<hendelse>
  <type>New York</type>
  <lokasjon>1947</lokasjon>
  <dato />
</hendelse>
```

Spirit samlingen blander også datatyper i ett felt, noe som kanskje er grunnet dårlig metadataformat. Det umuliggjør uansett korrekt indeksering av innholdet i tilfellene under:

```
<funksjon>Petch, Gladys, gb, d. 1986</funksjon>
<navn>Mjelva, Gunvor, no, 1902-</navn>
```

Topbase
Som de andre vart Topbase delt opp. Det var her brukt annet datoformat enn hva systemet opererer med, men dette var mulig å normalisere igjennom XML Schema. Ellers måtte også her ÆØÅ fjernes ifra elementnavn, dette har med moduler som prototypen bruker og vil kunne fikses i et ferdig system.

Trobib
Også delt opp i enkeltdokumenter. Samlingen inneholdt mange kryptiske elementnavn og datainnhold med forkortelser og tegn som vanskeliggjør datatypebasert indeksering.

```
<PU>Theim, 1971. -</PU>
<PU>[Thjem u.å.] -</PU>
```

OAI metadata
Dokumenter som allerede inneholdt XML Schema informasjon vart modifisert til å bruke andre skjema. I et endelig system kan metoder tidligere nevnt på grunnlag av XML Schema standarden brukes. Prototypen støtter bruk av valgt datatypesett, og muliggjør med små endringer nye datatyper.

Igjennom mulighetene for normalisering og formatering av dataverdier som prototypen viser er mulig å implementere igjennom XML Schema, støtter prototypen konfigurering gjort av datatilbyder. Det er likevel ikke tvil om at valgte dokumentsamlinger støtter dårlig ønsket metode og gjør det naturlig å tro at tilsvarende kan gjelde for mange andre samlinger også.

5.3.4 Databasetesting

Berkeley DB XML databasen er testet ved at prototypen lagrer de nevnte dokumentene i denne for deretter å hente ut data igjennom XPath uttrykk under indeksering både mot enkeltdokumenter og over hele samlingen. Databasen tillater også XML dokumenter uten eller med XML Schema å bli lagt inn også, dette bryter mot systemspesifikasjonen. Ellers fungerer databasen ut i fra angitte spesifikasjoner.

5.3.5 Indeksering av dokumenter

Indeksering av dokumenter er hovedoppgaven til prototypen og dette har blitt testet for å se hvor godt prototypen gjør dette ut i fra stilte krav ved å:

1. Legger dokumenter ifra hver av de seks forskjellige samlingene med XML Schema angitt inn i databasen. Dette fungerte for alle dokumenter forsøkt lagt inn i databasen. Prototypen støtter likevel ikke fullt ut kravspesifikasjonen siden

prototypen legger til grunn at dokumentsamlingenes datainnhold ikke skal måtte modifiseres, deriblant sti til XML Schema. Dette er likevel ikke en alvorlig mangel siden det er tilbyder sitt ansvar og ikke mottaker.

2. Henter ut dokumentene fra databasen, leser XML Schema angitt og for hvert element med datainnhold, normalisere dette om angitt og indeksert som definert datatype i XML Schema. For alle dokumenter forsøkt indeksert hentet prototypen ut alle data, tolket tilhørende XML Schema og indekserte innholdet slik dette var konfigurert.

Testingen viser at indeksering fungerer der datatyper er angitt. Enten ved bruk av datatype i XML Schema eller ved bruk av xs:annotation der innholdet i tillegg blir indeksert, normalisert og behandlet basert på datatype ved bruk av ønskede prosesseringsmetoder.

Rett indeksering krever likevel at dokumentene følger noen regler for hva en datatype kan inneholde. Dette er dårlig støttet av datasamlingene som er testet mot systemet. I de fleste tilfeller kan datainnholdet normaliseres ved å angi format tilsvarende som prototypen muliggjør. Disse viser fleksibiliteten ved bruk av xs:annotation fremfor typeparameter i prototypen. Er likevel problematisk ettersom flere av samlingene viser at formater brukt i felt varierer. Informasjonsinnholdet er ikke beregnet for forståelse av en datamaskin i mange tilfeller. Prototypen er derfor avhengig av kvaliteten på datagrunnlaget for godt resultat.

Tillegging av nye datatyper i prototypen er lett ettersom xs:annotation kan inneholde vanlig XML som ønskelig. Dette gjør prototypen robust kontra å takle variasjoner i datainnholdet dersom en effektiv protokoll for bruk av denne muligheten utvikles.

Indekseringen fungerer som ønsket ut i fra systemspesifikasjonen selv om enkelte problemer ikke er løst fullt ut i prototypen men bare implementert igjennom forslag til hvordan de kan løses.

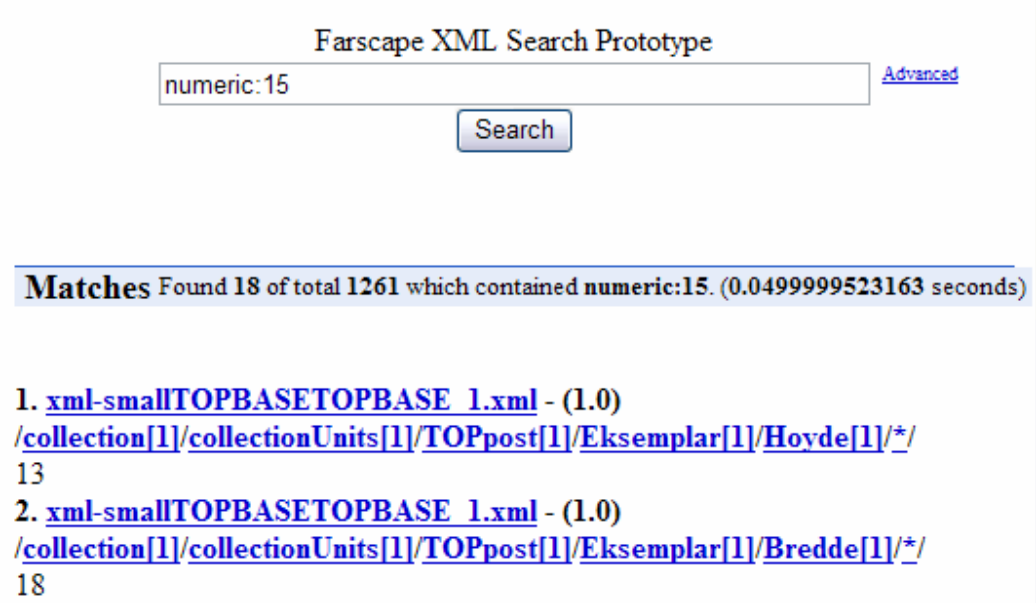
5.3.6 Testing av søk

Søk i prototypen har primært som hensikt å verifisere at indekseringen fungerer, sammen med å vise de tilfellene der prototypen feiler. Igjennom testing av søkemodulen kan det bekreftes at de andre delene av systemet fungerer siden denne modulen er avhengig av at de følger kravspesifikasjonen.

Prototypen støtter de krav som blir stilt til både enkelt og avansert søk. Det fokuseres derfor primært på hvilke områder indekseringen ikke fungerer slik at søking gir et annet resultat enn hva som er ønskelig. De problemene som her er nevnt er på grunn av valgt datagrunnlag og prototypen som mangler generelle rutiner til å indeksere innholdet korrekt.

Omtrentlig tallsøk

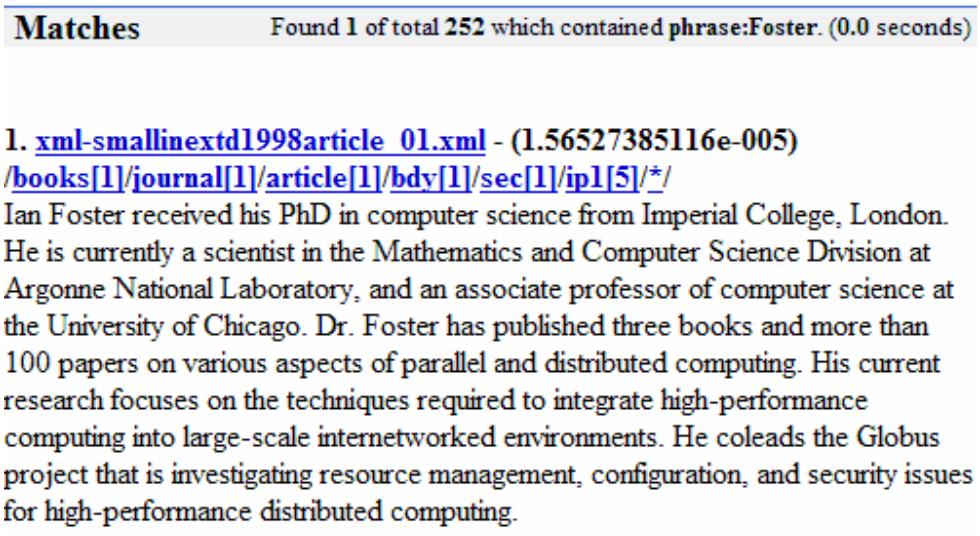
Eksempelet under viser et søk etter nummer der søkemotoren returnerer indekserte data som er i nærheten av faktisk søkt verdi ved numerisk sammenligning. Dette er derfor et eksempel på søk der alt fungerer som ønskelig.



Figur 20 - Søkeeksempel omtrentlig tallsøk

Feilindeksering av data

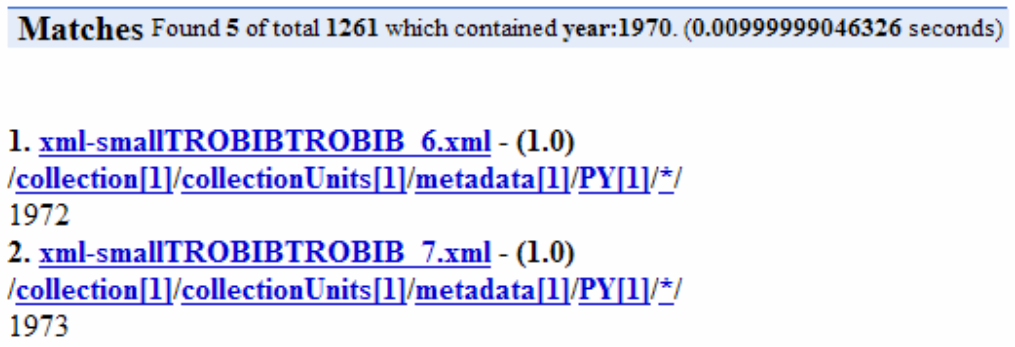
Søk i datatypen "name" etter navnet Foster returnerer ingen treff og det er derfor naturlig å anta at det ikke eksisterer noen som innehar navnet det søkes etter. Eksempelet under viser søk etter navn i en frasedatatype. Et søk etter frase returnerer likevel treff. Dette viser at brukt indekseringsmetode ikke fungerer på alle datasamlinger der innholdet ikke er godt nok annotert. I dette tilfellet er ikke navnet i et eget element i XML filen.



Figur 21 - Søkeeksempel feilindeksering av navn

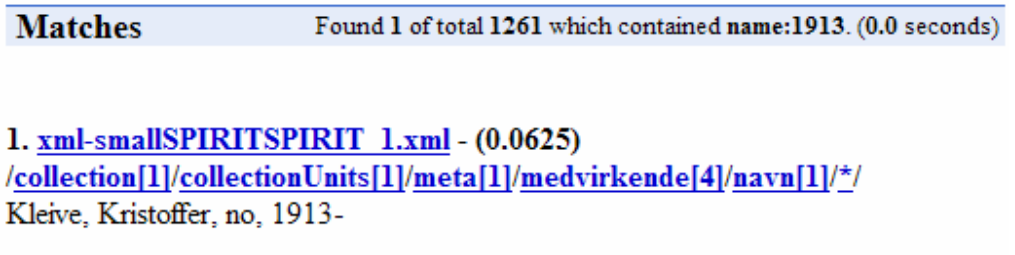
Delvis gjenfinning i dårlig annoterte data

Søkesystemet støtter i tillegg å søke etter årstall ved bruk av datatypen år. Dette fungerer fint selv om feltene inneholder ukorrekt data som fjernes siden kun numeriske verdier godtas.



Figur 22 - Søkeeksempel på år

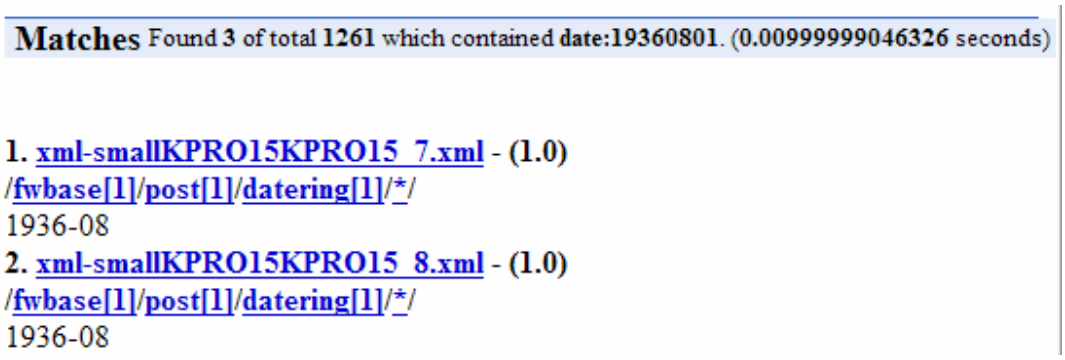
I nederste eksempel vises at felt kan inneholde mye ekstra informasjon enn bare det et elementnavn kan gi uttrykk for. Treff på årstall ved søk i navnefelt bør i et effektivt system ikke forekomme.



Figur 23 - Søkeeksempel datatype name

Søk i mixed content

Eksempelet under viser at det er mulig å normalisere datoer slik at de likevel kan søkes i til tross for at de innehar annet format i selve XML dokumentene.



Figur 24 - Søkeeksempel normaliserte datatyper og mixed content

5.3.7 Oppsummering

Testingen viser at de valgte datasamlingene passer dårlig til bruk oppimot prototypen grunnet det varierte innholdet de har. De muliggjør derimot til å vise hvilke problemer valgt indekseringsmetode innehar og fungerer til å teste ut prototypen sin støtte for håndtering av XML dokumenter, indeksering, tolkning av XML Schema og søking. De metoder som scenarioene i kapittel 1 stilles kan også gjennomføres.

Ut i fra søkeresultatene oppfyller prototypen kravene stilt og muliggjør derfor evaluering av løsningen ved bruk av utviklet prototyp og valgte data.

6 Evaluering

Dette kapittelet evaluerer forslaget til løsning på hvordan utfordringene nevnt i problemformuleringen kan løses. I forrige kapittel vart forslaget til løsning testet ut i fra systemkrav og det vart vist at prototypen oppfyller de krav satt til denne. Prototypen kan derfor brukes til å evaluere foreslått løsning. I kapittel 1 beskrives målsetningen som:

> *Det overordna målet er å vise at det er mulig å indeksere heterogene XML dokumenter uavhengig av elementnavn. Istedenfor skal informasjonen som beskriver innholdet gis som skjemainformasjon. Å relatere data basert på blant annet datatyper kan være en mulig løsning siden en datatype kan begrense et søke til dokumenter inneholdende en spesifikk type informasjon. Et hovedmål med oppgaven er derfor å se på hvor godt eksisterende standarder og teknologier, inkludert XML dokumenter slik de er oppbygd i dag, støtter denne metoden med informasjonsgjenfinning som mål.*

Like interessant som hva som faktisk er gjennomførbart og kan oppnåes er hva som ikke kan gjøres og hva som ikke er hensiktsmessig å gjøre. Hvordan de forskjellige delene av systemet fungerer, spesielle forutsetninger og erfaringer er evaluert.

6.1.1 Prototyp som helhet

Hovedmålsetningen for denne oppgaven er angitt over og er utgangspunkt for utviklet system. På bakgrunn i testingen av systemet kan det sies at det som helhet fungerer sammen med de forskjellige komponentene det er sammensatt av. Mer spesifikt viser prototypen at foreslått metode fungerer for utføring av søk i samlinger av heterogene semistrukturerte dokumenter, gitt et godt datagrunnlag. Arbeidet med å utvikle en systemarkitektur og en prototyp har resultert i noen interessante momenter. Noen av de viktigste er:

- Bruk av xs:annotation i XML Schema er veldig fleksibelt og har stort potensiale.
- XML Schema sin datatypestøtte støtter ikke alle støttede strukturer som er mulig å bruke i XML.
- Det er mulig å søke i heterogene XML dokumenter basert på datatyper ved bruk av PyLucene.
- Eksisterende XML samlinger har stor sannsynlighet for dårlig strukturkvalitet og det kan derfor ikke stoles på at innholdet er konsistent.
- Det er derfor vanskelig å angi mange typer som en datatype.
- Det er vanskelig å finne eksisterende bibliotek som implementerer XML standarder fullt ut.

Selv om systemet fungerer er det mange forutsetninger og begrensninger som måtte tas for få et system som var mulig å gjennomføre. Et godt eksempel på dette er at de aller fleste problemer er relatert til eksterne systemer og data, og ikke til foreslått løsning.

Det kan diskuteres om de testede data har vært for begrenset i omfang til å kunne fremsette påstandene gitt ovenfor. Jeg vil likevel påstå at de har vært nok til å kunne teste og vise at datainnholdet i de brukte dokumentene til en viss grad muliggjør valgt metode. Den valgte metoden fungerer, men er avhengig av sterkt strukturerte XML dokumenter for å være en optimal løsning. Dette kan antyde at foreslått metode fungerer best mot spesifikke dokumentsamlinger med høy kvalitet eller begrenset til enkelte deler av et dokument og i visse situasjoner.

De forskjellige komponentene har vist seg å fungere for prototypen men vil kunne skape problemer ved videreutvikling, spesielt vedrørende hastighet. Andre løsninger må derfor vurderes for et operasjonelt system.

For rask prototyping har de brukte komponentene fungert og vært stabile sammen med brukte programmeringsspråk. Noen problemer har likevel oppstått, spesielt relatert til minnebruk ved indeksering av store dokumentsamlinger siden PyLucene har en del feil og tolkning av store XML dokumenter er ressursskrevende.

6.1.2 XML og angivelse av datatyper

Oppgaven viser at XML og tilhørende XML Schema standard muliggjør homogen indeksering til tross for heterogen struktur ved bruk av datatyper, selv om godt resultat er avhengig av datatyper og datainnholdet i dokumentene. Det kan også sies at mange av XML standardene er for komplekse til at alle deler av de har blitt implementert i mange XML biblioteker noe som vanskeliggjorde implementasjon av løsningen. XML Schema tolkning og uthenting av informasjonen etterpå har fungert takket være modifisert versjon av W3C sin offisielle tolker. Dette kan også illustreres ved at XML Schema standarden er så stor at den muliggjør egne XML dokumenter uten begrensninger inne i et XML Schema.

Grunnet kompleksiteten til XML Schema er det mye av funksjonaliteten som vart oppdaget underveis og førte til at oppgavens fokus endret seg når nye momenter dukket opp. Dette kan illustreres ved at begrensninger for annotering av datatype for ”mixed content” ikke vart oppdaget før sent i arbeid på oppgaven. Bruk av xs:annotation har likevel vist seg å være å foretrekke for å angi datatyper. Denne har større fleksibilitet til å definere andre former for data som muliggjør formatering og normalisering til et homogent format som kan indekseres korrekt.

Siden muligheten til å angi datatyper og metoder for hver enkelt datatype var viktigst, er de datatyper som har blitt valgt i denne oppgaven en forenkling og kun valgt for å ha noe å teste metodikken på. Datatypene som brukes i et operasjonelt system må derfor vurderes uavhengig av de brukt i denne oppgaven.

Det kan derfor sies at den valgte metoden fungerer bra, er mulig ut i fra standardene og har fordeler som gjør at det kan søkes i mange dokumenter selv om strukturen er forskjellig.

6.1.3 Eksisterende dokumentsamlinger

De valgte dokumentsamlingene eksemplifisert igjennom de dokumentene som er brukt under testing av prototypen viser at eksisterende samlinger av den typen brukt i prototypen ikke er optimale for implementert løsningsmetode.

Det er vanskelig å definere de valgte datatypene og usikkerhet vedrørende hvilken datatype et felts innhold kan defineres som, dette mye grunnet dårlig valg av datatyper men også valg av testsamlinger. Det kan derfor diskuteres om valgt metode fungerer for den type dokumenter oppgaven baserer seg på for mange eksisterende samlinger. De viser likevel at løsningen muliggjør de scenarioer angitt i første kapitlet.

Forsøkene på indeksering av datainnhold viser at de lite rigide kravene til datainnholdet som XML stiller ikke oppfyller de krav som informasjonsgjenfinning har. Gjennomførte regler er nødvendig for at dette skal gå. Det er uansett vanskelig å utføre kontroll i en ”name” datatype om etternavn faktisk står til slutt eller først. Systemer som baserer seg på innhold er derfor avhengig av at dataleverandør faktisk har holdt seg til ønskede krav til formater, ideelt sett hadde hver enkelt enhet data i et XML dokument vært inne i en eget element med egen datatype.

Prototypen og testene viser likevel at det er mulig å indeksere dokumentinnhold ut i fra XML Schema annotert av datatilbyder og minsker derfor det manuelle arbeidet, dette muliggjør rask sammenkobling av data fra to like systemer med ulik struktur. Men vanlige XML samlinger er ikke mulig å bruke for denne type søk over hele dokumenter dersom ikke innholdet er strengt strukturert og følger en datatypes regler.

Indeksering basert på datatype kan forbedre spørringer ved å indeksere et XML dokument på vanlig måte, mens enkelte elementer indekseres med datatyper og dermed muliggjør mer effektiv søking i disse. Eksempel på dette er at prisinformasjon kan indekseres som nummer og dermed muliggjør omtrentlige søk mens resten fungerer som normalt. Dette gjør det lettere å koble sammen forskjellige typer XML dokumenter som er ulikt annotert men har likt innhold. En generell søkemotor kan dermed lett utvikles for bruk mot mange forskjellige dokumenttyper og scenarioer.

Felles forståelse for strukturen i et dokument har likevel liten relevans så lenge innholdet som beskrives varierer kraftig. En bruker kan ikke vite at ”Tr.Hjem” og ”Trondheim” er forskjellige skrivemåter av det samme, altså kreves det i tillegg andre metoder for å normalisere informasjonen beskrive av strukturen.

6.1.4 Indeks- og søkemotor

Prototypen viser at det fungerer bra å bruke en vanlig søkemotor for ustrukturert tekst ved søking basert på datatype til tross for at denne ikke er beregnet på dette. I PyLucene sitt tilfelle fungerte det bra å bruke eksisterende muligheter til å indeksere forskjellige datatyper i samme indeks, tilsvarende kan gjøres ved å ha egne indekser i andre systemer. Der datatypene ikke er direkte støttet i PyLucene er dette løst på andre måter ved å endre de indekserte data basert på datatype for å støtte nødvendige operatorer. Dette viser at vanlige ustrukturerte søkemotorer muliggjør søking basert på datatype selv om de lagrer data internt som tekst. Prototypen er likevel avhengig av at datagrunnlaget er korrekt

dersom dette skal være mulig, spesielt siden data forstås av tekst må datainnholdet være korrekt for at normalisering skal kunne utføres.

Prototypen viser også at ustrukturerte søkemotorer som PyLucene ikke er beregnet for ekstra behandling av data i etterkant av et søk som krever tilgang til absolutt alle treff, tiden det tar å få tilgang til data om resultatsettet øker raskt dersom et sett inneholder mange dokumenter. Mer effektiv tilgang til data er derfor nødvendig for å kunne støtte XML søking dersom det er ønskelig å utnytte struktur i tillegg.

For søking viser prototypen at en slik søkemotor kan fungere ved bruk av datatyper dersom den tilbyr mulighet til å tilpasse søk før dette utføres og muligheter for vage spørringer. Standardinnstillinger fungerer ellers dårlig, spesielt for rangering av resultater på grunn av store variasjoner i datalengde i et XML dokument siden dette fører til feil normalisering av lengde. Andre metoder for dette er derfor nødvendig, noe da også PyLucene muliggjør.

Kontra ustrukturerte tekstdokumenter er det også for XML dokumenter mer komplisert å presentere søkeresultat siden datainnholdet ikke kan presenteres direkte inkludert tagger. Dette kan løses ved å kreve XML Stylesheet for hvordan hver enkelt dokumentsamling skal presenteres.

Til tross for de ulemper nevnt fungerer en slik søkemotor fint mot XML dokumenter slik her benyttet. Selv om den brukte søkemotoren muliggjorde modifisering var ikke noe av det som vart implementert forstås som ikke utelukker at andre tilsvarende søkemotorer kan brukes.

7 Konklusjon

Den utviklede prototypen utnytter deler av XML Schema som er lite brukt i informasjonsforvaltning. Min løsning viser at bruk av XML Schema slik gjort i prototypen fungerer for å løse de utfordringene problemstillingen beskriver.

Blant forbeholdene for løsningen er at den implementerte metoden krever dokumenter som er detaljert markert for alle datatyper. Dette viser seg å ikke være tilfelle og fører til mange feilindekseringer. De valgte samlingene viste likevel hvor løsningen fungerer og hvor den fungerer mindre bra.

Metodene brukt har stor nytteverdi for å raskt muliggjøre søk i heterogene samlinger som har forskjellig struktur men likevel likt datainnhold. Sammen med bruk av eksisterende søkemotorer muliggjør dette raskt å kunne implementere systemer for søk i ulike typer heterogene dokumenter, enten ved bruk av datatyper over hele eller deler av dokumentene.

Basert på hva som er oppnådd under arbeid på oppgaven og hva det var ønskelig å oppnå ifra starten av, vil jeg konkludere med at oppgavens mål har blitt nådd. XML, XML Schema og brukt søkemotor innehar støtte for indeksering av heterogene XML dokumenter ved bruk av foreslått indekseringsmetode.

7.1 Evaluering av tilnærming

Evalueringen av tilnærmingen tar for seg hvordan arbeidet på oppgaven har blitt gjort og fordeler og ulemper med nyttet tilnærmingsmetode.

Som tilnærmingsmetoden i kapittel 1 viser har en lineær tilnærming til det teoretiske bak oppgaven vært nyttet mens det praktiske arbeidet på prototypen har foregått ut i fra en spiralmodell. Denne tilnærmingen muliggjorde større forståelse for oppgaven siden ny teoretisk kunnskap hele tiden vart nyttet under arbeid på prototypen. Dette førte også til at ny kunnskap noen ganger krevde at tidligere implementerte funksjoner måtte fjernes siden de vart redundante. Noe tid brukt og arbeid var derfor unødvendig for den endelige prototypen og problemformulering, men jeg kan likevel ikke se vekk ifra at dette arbeidet var til hjelp for den endelige løsningen på en eller annen måte.

Hovedulempen med tilnærmingen var nok likevel at problemformuleringen ikke var endelig ved prosjektets start men vart endret underveis. Dette skapte dog en mer fleksibel løsning og en bedre avhandling enn om problemformuleringen hadde vært endelig før oppgavens start. Dette illustreres godt av at det første utkastet til problemformulering var å implementere en prototyp som inneholdt all funksjonalitet nødvendig for gjenfinning i XML. Noe jeg med dagens viten ikke tror hadde vært sannsynlig å fått fullført tidsnok.

Min primære erfaring med tilnærmingen er at den har gitt meg mulighet til snevre inn oppgavens mål underveis og fokusere avhandlingen på et spesifikt problem uten at dette har gått ut over kvaliteten på oppgaven. Hovedårsaken til at den fungerte godt for meg er nok mye takket være at arbeidet kunne varieres mellom teoretisk og praktisk ved behov, noe som var med på å opprettholde motivasjonen. Jeg tror nok ikke at den ville fungert like godt dersom jeg ikke hadde vært alene om oppgaven.

7.2 Videre arbeid

Siden mange avgrensninger av oppgaven måtte tas underveis for å kunne fokusere arbeidet i en retning, eksisterer det mange områder for videre utvikling basert på hva denne oppgaven har funnet ut.

En oversikt over identifiserte områder for videre arbeid følger under, både videre utvikling av løsningen på denne oppgaven og andre aktuelle muligheter.

- Automatisk identifisering av datatyper til datainnholdet for å tilby bedre søking på datatyper uavhengig av skjemainnhold.
- Bruk av XML Schema sin xs:annotation til å angi datatype fremfor xs:simpletype for å spesifisere hvordan innholdet skal indekseres. Det er dette xs:annotation er beregnet på og det muliggjør å kunne definere flere parametere. Det vil likevel føre til mye ekstra innhold i XML skjema dersom alle applikasjoner skal ha sine egne definisjoner noe som kan være negativt i enkelte tilfeller.
- Utvikle fullstendig XML søkesystem basert på prototypen
- Utarbeide bedre datatyper og scenarioer som er optimale for løsningen
- Utvikling av automatisk tolkning av datatype i en spørring slik at det ikke er behov for å spesifisere dette sammen med en spørring
- Under testing og evaluering av prototyp har ikke evaluering av kvalitative begreper relatert til søkeresultat vært relevant å se på. Videre arbeid bør derfor fokusere på å se på dette.

8 Referanseliste

[1] Overview of SGML Resources.
http://www.w3.org/MarkUp/SGML/ (8. Mai 2006)

[2] Shaorong Liu., Zou, Q., og Wesley W. Chu. Configurable indexing and ranking for XML information The 27th Annual International ACM SIGIR Conference (Sheffield, England, 2004), 88-95.

[3] Dewey Decimal Classification System
http://www.tnrdlib.bc.ca/dewey.html (8. Mai 2006)

[4] Encyclopedia: Information retrieval (IR)
http://encyclopedia.laborlawtalk.com/Information_retrieval (8. Mai 2006)

[5] C. J. van Rijsbergen. Information Retrieval
http://www.dcs.gla.ac.uk/Keith/Chapter.1/Ch.1.html (8. Mai 2006)

[6] Wikipedia Datatype.
http://en.wikipedia.org/wiki/Datatype (8. Mai 2006)

[7] Dictionary.com Datatype
http://dictionary.reference.com/search?q=data%20type (8. Mai 2006)

[8] Guojun Lu. Multimedia database management systems. Artech House Publishers (October 1999)

[9] C. Buckley, G. Salton og J. Allan. The SMART Information Retrieval Project. Proceedings of the workshop on Human Language Technology HLT '93.

[10] Text REtrieval Conference
http://trec.nist.gov/ (8. Mai 2006)

[11] Herindrasana Ramampiaro. Foiler fra IT2801 Informasjonsgjenf V-06.

[12] Charming Python: Developing a full-text indexer in Python
http://www-128.ibm.com/developerworks/XML/library/l-pyind.html (8. Mai 2006)

[13] Google Help
http://www.google.no/help/index.html (8. Mai 2006)

[14] Ray R. Larson. XML Element Retrieval and Heterogeneous Retrieval: In Pursuit of the Impossible?. INEX 2005 Workshop on Element Retrieval Methodology.

[15] Jovan Pehcevski, James A. Thom og Anne-Marie Vercoustre. Hybrid XML Retrieval: Combining Information Retrieval and a Native XML Database. Information Retrieval, 8(4):571-600, December 2005.

[16] INitiative for the Evaluation of XML Retrieval
http://inex.is.informatik.uni-duisburg.de/ (8. Mai 2006)

[17] Open Archives Initiative
http://www.openarchives.org/ (8. Mai 2006)

[18] Extensible Markup Language (XML).
http://www.w3.org/XML/ (8. Mai 2006)

[19] HyperText Markup Language (HTML) Home Page.
http://www.w3.org/MarkUp (8. Mai 2006)

[20] J. Alfredo Sanchez, Carlos Proal og Fernanda Maldonado-Naude. Supporting Structured, Semi-Structured and Unstructured Data in Digital Libraries. Proceedings of the Fifth Mexican International Conference in Computer Science (ENC'04) - Volume 00 Pages: 368 - 375.

[21] Extensible Stylesheet Language (XSL) Version 1.0
http://www.w3.org/TR/2001/REC-xsl-20011015/ (18.Mai 2006)

[22] Cascading Style Sheets, level 2 CSS2 Specification
http://www.w3.org/TR/1998/REC-CSS2-19980512/ (18.Mai 2006)

[23] W3Schools Introduction to DTD.
http://www.w3schools.com/dtd/dtd_intro.asp (8. Mai 2006)

[24] W3Schools Introduction to XML Schema..
http://www.w3schools.com/schema/schema_intro.asp (8. Mai 2006)

[25] XML Schema Part 0: Primer Second Edition.
http://www.w3.org/TR/XMLschema-0/ (8. Mai 2006)

[26] Schematron
http://www.schematron.com/ (8. Mai 2006)

[27] W3C World Wide Web Consortium.
http://www.w3.org (8. Mai 2006)

[28] XML Schema Datatypes in RDF and OWL.
http://www.w3.org/TR/2006/NOTE-swbp-xsch-datatypes-20060314 (8. Mai 2006)

[29] XML Schema Part 2: Datatypes Second Edition
http://www.w3.org/TR/XMLschema-2/#built-in-datatypes (8. Mai 2006)

[30] Dublin Core Metadata Initiative
http://dublincore.org/ (8. Mai 2006)

[31] MODS: Uses and Features
http://www.loc.gov/standards/mods/mods-overview.html (8. Mai 2006)

[32] MARC standards
http://www.loc.gov/marc/ (8. Mai 2006)

[33] Kimbro Staken. Introduction to Native XML Databases
*http://www.XML.com/lpt/a/2001/10/31/nativeXMLdb.*html (8. Mai 2006)

[34] Robert W.P. Luk, H.V. Leong, Tharam S. Dillon, Alvin T.S. Chan, W. Bruce Croft og James Allan. A survey in indexing and searching XML documents. Journal Of The American Society For Information Science And Technology, 53(6):415–437, 2002.

[35] Al-Khalifa, S., Yu, C., and Jagadish, H. V. 2003. Querying structured text in an XML database. In *Proceedings of the 2003 ACM SIGMOD international Conference on Management of Data* (San Diego, California, June 09 - 12, 2003). SIGMOD '03. ACM Press, New York, NY, 4-15.

[36] XML Path Language (XPath)
http://www.w3.org/TR/XPath (8. Mai 2006)

[37] W3C XML Query (XQuery)
http://www.w3.org/XML/Query/ (8. Mai 2006)

[38] W3Schools Introduction to XQuery.
http://www.w3schools.com/XQuery/XQuery_intro.asp (8. Mai 2006)

[39] Evan Lenz. XQuery: Reinventing the Wheel?
http://www.XMLportfolio.com/XQuery.html#d39e45 (8. Mai 2006)

[40] DELOS Digital Library.
http://www.delos.info/ (8. Mai 2006)

[41] Andrew Trotman og Börkur Sigurbjörnsson. Narrowed Extended XPath I (NEXI). INEX 2004 Workshop Proceedings pp. 16-40.

[42] Andrew Trotman og Richard A. O'Keefe. Identifying and Ranking Relevant Document Elements. INEX 2003 Workshop Proceedings pp. 149-154.

[43] Anastasios Tombros, Birger Larsen og Saadia Malik. The Interactive Track at INEX 2004
http://www.is.informatik.uni-duisburg.de/bib/pdf/ir/Tombros_etal:05.pdf (8. Mai 2006)

[44] Z39.50 Maintenance Agency Page
http://www.loc.gov/z3950/agency/ (8. Mai 2006)

[45] XML: Metadata For the Rest of Us (Part 1)
http://www.wired.com/news/technology/0,1282,4997,00.html (8. Mai 2006)

[46] Antony Corfield, Matthew Dovey, Richard Mawby, og Colin Tatham. Z39.50 and XML - Bridging the old and the new.
http://www2002.org/CDROM/alternate/XS2/ (8. Mai 2006)

[47] XER (XML Encoding Rules)
http://asf.gils.net/xer/ (8. Mai 2006)

[48] Paul Miller. Z39.50 for All.
http://www.ariadne.ac.uk/issue21/z3950/ (8. Mai 2006)

[49] Global Information Locator Service (GILS) Metadata Elements
http://www.gils.net/elements.html (8. Mai 2006)

[50] Ray R. Larson. Cheshire II at INEX '04: Fusion and Feedback for the Adhoc and Heterogeneous Tracks. Lecture Notes in Computer Science Volume 3493 / 2005.

[51] Norbert Fuhr og Kai Großjohann. XIRQL: An XML Query Language Based on Information Retrieval Concepts. ACM Transactions on Information Systems 22. p 313—356, 2004.

[52] Apache Lucene
http://lucene.apache.org/java/docs/index.html (16.Mai 2006)

[53] PyLucene project
http://pylucene.osafoundation.org/ (8. Mai 2006)

[54] eXist - Open Source Native XML Database
http://exist.sourceforge.net/ (8. Mai 2006)

[55] Ralf Schenkel, Anja Theobald og Gerhard Weikum. Semantic Similarity Search on Semistructured Data with the XXL Search Engine. Information Retrieval, 8, 521–545, 2005.

[56] Felix Weigel, Holger Meuss, Klaus U. Schulz og Francois Bry. Content and Structure in Indexing and Ranking XML. Seventh International Workshop on the Web and Databases (WebDB 2004) June 1718, 2004, Paris, France.

[57] Jovan Pehcevski, James A. Thom og Anne-Marie Vercoustre. RMIT INEX experiments: XML Retrieval using Lucy/eXist. INEX 2003 Workshop Proceedings, Dagstuhl, Germany, December 15-17, 2003. pp 134 - 141.

[58] Selim Mimaroglu. Open Source Database Special Feature: An Introduction to Berkeley DB XML
http://au.sys-con.com/read/164567.htm (8. Mai 2006)

[59] XQEngine - XML Query Engine
http://sourceforge.net/projects/xqengine/ (8. Mai 2006)

[60] Ronald Bourret. Native XML Databases
http://www.rpbourret.com/XML/ProdsNative.htm (8. Mai 2006)

[61] Norbert Fuhr. XML Information retrieval
http://www.is.informatik.uni-duisburg.de/courses/ir_ss05/folien/irXML.pdf (8. Mai 2006)

[62] Guidelines for implementing Dublin Core in XML
http://dublincore.org/documents/dc-XML-guidelines/ (8. Mai 2006)

[63] Denilson Barbosa, Attila Barta og Alberto Mendelzon. ToX – The Toronto XML Engine. International Workshop on Information Integration on the Web, Rio de Janeiro, 2001. p. 66-73.

[64] Shankar Pal, Istvan Cseri, Oliver Seeliger, Gideon Schaller, Leo Giakoumakis og Vasili Zolotov. Indexing XML Data Stored in a Relational Database. Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004.

[65] RDF Primer W3C Recommendation 10 February 2004.
http://www.w3.org/TR/2004/REC-rdf-primer-20040210/#statements (8. Mai 2006)

[66] Henry S. Thompson og Lars Marius Garshol. Re: XML Schema date syntax.
http://www.stylusstudio.com/XMLdev/200011/post30120.html (8. Mai 2006)

[67] Rick Jelliffe. W3C XML Schema Datatypes Reference.
*http://www.XML.com/pub/a/2000/11/29/schemas/dataref.*html (8. Mai 2006)

[68] Jeni Tennison og Richard Liu. Re: Restricting text in mixed content element.
http://lists.w3.org/Archives/Public/XMLschema-dev/2001Dec/0121.html (8. Mai 2006)

[69] Eric van der Vlist. Using W3C XML Schema - Part 2.
http://www.XML.com/lpt/a/2000/12/13/schemas/part2.html (8. Mai 2006)

[70] ISO/IEC JTC 1/SC 34. Information Technology - Document Description and Processing Languages
http://www.jtc1sc34.org/repository/0392.htm (8. Mai 2006)

[71] Brandon Jockman, W. Eliot Kimber og Joshua Reynolds. Case Study: Enabling Low-Cost XML-Aware Searching Capable of Complex Querying.
http://www.idealliance.org/papers/XMLe02/dx_XMLe02/papers/03-02-08/03-02-08.html (8. Mai 2006)

[72] Otis Gospodnetic. Parsing, indexing, and searching XML with Digester and PyLucene
http://www-128.ibm.com/developerworks/java/library/j-lucene/ (8. Mai 2006)

[73] R. A. O'Keefe and A. Trotman. The Simplest Query Language That Could Possibly Work. In Proceedings of the 2nd INEX Workshop, 2004.

[74] Börkur Sigurbjörnsson og Andrew Trotman. Queries: INEX 2003 working group report. In: INEX 2003 Workshop Proceedings. Pages: 167-170. 2004.

[75] Daniela Florescu, Ioana Manolescu og Donald Kossmann. Integrating Keyword Search into XML Query Processing.
http://www9.org/w9cdrom/324/324.html (8. Mai 2006)

[76] Shaorong Liu og Wesley W. Chu. Cooperative XML (CoXML) Query Answering at INEX 03. Proceedings of the 2nd Initiative of the Evaluation of XML Retrieval (INEX 2003) Workshop.

[77] Miro Lehtonen. Adapting heterogeneous XML to Information Retrieval with techniques independent of document type.
http://www.cs.helsinki.fi/hecse/Students/Abstracts-2004/Lehtonen_Miro.pdf (8. Mai 2006)

[78] Epostkorrespondanse med support hos Sleepycat. Referanse ikke tilgjengelig grunnet eposthavari.

[79] Epostkorrespondanse med Paul V.Biron
http://lists.w3.org/Archives/Public/xmlschema-dev/2006Mar/0071.html (18.Mai 2006)

[80] Jon Duckett. Getting Started with XML Schemas.
http://www.topXML.com/schema/articles/XMLschemas/default.asp (8. Mai 2006)

[81] The Python Programming Language
http://www.python.org (16. Mai 2006)

[82]Prechelt, L. 2000. An Empirical Comparison of Seven Programming Languages. *Computer* 33, 10 (Oct. 2000), 23-29. DOI= *http://dx.doi.org/10.1109/2.876288*

[83] Sleepycat Software
http://www.sleepycat.com (8. Mai 2006)

[84] XSV - Validator for XML Schema
http://www.w3.org/2001/03/webdata/xsv (18. Mai 2006)

[85] Effbot Elementtree
http://effbot.org/zone/element-index.htm (18. Mai 2006)

[86] Dare Obasanjo. A Data Model for Strongly Typed XML
http://www.XML.com/pub/a/2002/12/19/datamodel.html (8. Mai 2006)

[87] Universitetsbiblioteket i Trondheim
http://www.ub.ntnu.no/formidl/hist/trobib/ (25.05.2006)

A Appendix – XML testdata

Under testing av prototypen vart det nyttet testdata som kan finnes på den vedlagte CD'en i underkatalogen "appendix a - XML testdata". De forskjellige samlingene testdata iform av XML filer er vedlagt for er:

- Ett dokument fra INEX
- Ti dokumenter fra KPRO15
- Ti dokumenter fra OAI
- Ti dokumenter fra SPIRIT
- Ti dokumenter fra TOPBASE
- Ti dokumenter fra TROBIB

Dersom det er ønskelig å teste systemet basert på de testdata her nevnt må disse kopieres over til XML katalogen i prototypen. I tillegg må sti til XML Schema og DTD endres der dette er aktuelt.

XML er i noen tilfeller endret iforhold til originalene for å være mulig å bruke i oppgaven. Hva som er endret er nevnt i rapportens kapittel 5.

B Appendix – XML Schema

De forskjellige samlingene av XML dokumenter som vart nyttet under testing av systemet har hvert sitt tilhørende XML Schema som ligger sammen med hver samling i katalogen ” appendix b - XML Schema” på vedlagt CD. De forskjellige samlingene testdata iform av XML Schema filer er vedlagt for er:

- INEX
- KPRO15
- OAI
- SPIRIT
- TOPBASE
- TROBIB

Dersom det er ønskelig å teste systemet basert på de testdata her nevnt må disse kopieres over til XML katalogen i prototypen. Ikke alle skjema er komplette eller korrekt utfylt, dette siden det ikke var hensiktsmessig å annotere alle elementer for alle dokumentsamlinger for å bruke de for testing av systemet. I mange tilfeller var det heller ikke mulig å angi en datatype for et element som fungerte for alle dokumentene testet i en samling.

C Appendix – Prototypkode

For prototypen er det på vedlagt CD lagt ved kode for prototypen i underkatalogen ”appendix c - Kode for prototyp”. I tillegg til egen kode er det vedlagt fullstendig kode for den modifiserte versjonen av XSV (XML Schema Validator) som er brukt for tolkning av XML Schema.

Filer i prototypen	**Oppgave**
HTTPController	Inngangsport for http server og søk
IndexerController	Inngangsport for indeksering
Modules / Annotations	Ansvarlig for data ifra xs:annotation feltet
Modules / BerkeleyDBXML	Klasse for import av filter til DBXML
Modules / CommonFunctions	Ofte brukte funksjoner
Modules / Database	Oppretter og stenger databasetilkoblinger
Modules / DataTypeNormalizer	Normaliserer basert på datatype
Modules / HTTPInterface	Inneholder kode for html grensesnitt
Modules / HTTPProcessor	Tar seg av behandling av søk
Modules / Indexer	Forbehandler XML før indeksering
Modules / Logger	Logging av beskjeder
Modules / Lucene /CustomQueryParser	Egen parser for modifisering av spørringer
Modules / Lucene / dateFilter	Datofilter
Modules / Lucene / myAnalyzer	Analyserer spørringer
Modules / Lucene / numberFilter	Nummerfilter, forbereder de til bokstavsøk
Modules / Lucene / NumberUtils	Padder nummer til en bestemt lengde
Modules / Lucene / ScoreMethod	Muliggjør endring av rangering
Modules / Lucene / Streams	Tar seg av filbehandling
Modules / Lucene / yearFilter	Filter for årstall i PyLucene