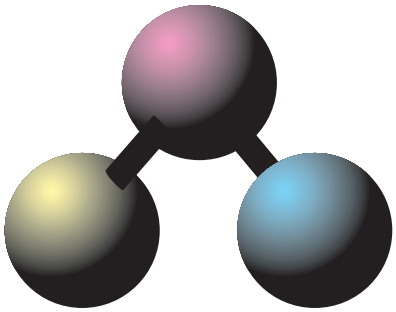
Relasjonssystem

Lenker mellom informasjonsobjekter i det digitale bibliotek



Einar W. M. Lichtenberg

Hovedfagsavhandling i Informasjonsforvaltning ved
institutt for data og informasjonsvitenskap,
NTNU

Juni 2000

Sammendrag

Avhandlingen tar for seg lenking mellom informasjonsobjekter i digitale bibliotek, og presenterer et system for relasjonsbasert lagring i det digitale bibliotekspro-sjektet ved IDI (DigLib).

Sammenlenking av informasjon basert på angivelse av forhold gjennom retnings- og type-bestemmelse gir mulighet for knyttinger mellom ulike informasjons typer. I et digitalt bibliotek kan slike *relasjoner* tilby oppdaging av informasjon utover vanlig søking og gjennhenting.

Det foreslås en todelt modell for lagring og organisering i et informasjons-system:

Lagringsmodell. En spesialisert modell for lagring av informasjon basert på semi-strukturert lagring i objektorientert database.

Relasjonsmodell. En spesialisert modell for relasjoner mellom informasjonsobjek-ter basert på avansert lenketeknikk og angivelse av relasjonsforhold.

Systemet som presenteres bygger på ideen om at lenker mellom ulik informasjon i et nettverk kan legges utenfor informasjonen som knytter lenken. Et slikt system gir mulighet for å bygge relasjoner mellom informasjonstyper som ikke har en intern lenke-struktur, f.eks.: bilder, lyder og andre binære filer uten lenkeangi-velse. Man trenger ikke å endre, eller å ha adgangsrettigheter til et informasjons-objekt for å knytte relasjoner til det. Systemet gir også mulighet for å uttrykke informasjon om forholdet mellom informasjonsobjektene.

Erfaringer fra systemet antyder at relasjoner mellom informasjonsobjekter er sær-lig anvendelig når man har samlinger med flere typer informasjonsobjekter (*mixed content)*. Eksempelvis blir en knytting mellom et dokument og metadata om doku-mentet ikke avhengig av å lagre lenke-informasjon i dokument-objektet eller metadata-objektet. Relasjoner mellom disse kan enkelt administreres uten å endre informasjonsobjektene. Også muligheten for å angi relasjons-type har vist seg å gi verdifull informasjon utover vanlig lenking.

Systemet muliggjør lagring av informasjon i informasjonsobjekter som bygger på *Dexter Hypertext Model* og Arms' arkitektur for informasjon i digitale bibliotek. Modellen utvider disse ved bruk av objektorientert oppbygging, basert på lagring i klasser med egne funksjoner for manipulering av innhold. Aksesseringsklasser for innlegging og uthenting av informasjonobjekter tillater bruk i fremtidige tjenester i DigLib.

Knyttinger mellom informasjonsobjekter skjer gjennom bruk av relasjoner. Slike

relasjoner uttrykker typebestemte lenker som bygger på prinsipper fra hypermedi-asystemer. Dette er realisert gjennom bruk av XLink spesifikasjonen for XML, og er slik utvidbart ved senere bruk av nye relasjonstyper. Det er også laget egne klasser for aksessering av slike relasjoner for fremtidig bruk i DigLib.

Innholdsfortegnelse

Innholdsfortegnelse

Innholdsfortegnelse

Innholdsfortegnelse

Innholdsfortegnelse

Innholdsfortegnelse

Liste over figurer

Liste over tabeller

Liste over eksempler

Kapittel 1

Innledning

Denne avhandlingen tar for seg lenking mellom informasjonsobjekter i digitale bibliotek. Systemet som foreslås bygger på det arbeid som forfatteren har gjort innenfor det digitale biblioteksprosjektet ved IDI.

I denne innledningen vil følgende punkter bli gjennomgått:

- Bakgrunn og motivasjon
- Problemområde
- Oppbygning av avhandlingen
- Oppsumering av arkitektur og systemer sett på i avhandlingen

1.1 Generelt om Oppgaven

1.1.1 *Bakgrunn og motivasjon*

Forfatteren av denne oppgaven har vært sterkt engasjert i det digitale biblioteksprosjektet ved IDI (*DigLib)*, og gjennom dette arbeidet fått muligheten til å delta i oppbyggingen av de systemer og arkitekturer som utgjør prosjektet. Man utviklet blant annet en modell og implementasjon av et system som prosesserer informasjon strukturert i *markup*-språket XML for lagring i en objektorientert database. Etter hvert har dette systemet vært testet med flere typer samlinger, og spørsmål om hvordan individuelle objekter i disse samlingene skal knyttes sammen har gjort seg gjeldende. Et system for relasjoner mellom disse objektene ble identifisert som et interessant område å å se nærmere på i denne forbindelse.

1.1.2 *Bruk av begrepet relasjon*

Relasjoner blir i denne hovedfagsavhandlingen brukt for å uttrykke: *et forhold mellom to informasjonsobjekter i en lenke*.

Relasjoner er også kjent som lenker fra blant annet HTML og andre hypermedia-systemer. Begrepet lenke blir i denne avhandlingen brukt med følgende betydning: *En lenke er en forbindelse mellom to informasjonsobjekter.* En slik forbindelse uttrykker at de to informasjonsobjektene henger sammen. Begrepet informasjonsobjekt blir gjennomgått i kapittel **3.3**. Begrepet lenke og relasjon blir tatt opp i kapittel **2.1**.

1.1.3 *Problemidentifisering*

Et av problemene som har dukket opp gjennom arbeidet med DigLib er hvordan en skal kunne knytte sammen et dokument og metadata om dokument. Dette problemet oppstod fordi DigLib-modellen (kapittel **3**) åpner for en modell som benytter "mixed content", det vil si en samling kan bestå av ulike typer media som bilder, lyder og tekster.

Da et relasjonssystem skulle være mulig å implementere ved å ta utgangspunkt i modellen for XML-basert lagring som var utprøvd, fant forfatteren det interessant å bruke hovedfagsavhandlingen til å utrede om dette temaet, utvikle en modell, og finne en mulig løsning på implementering. Et slikt system vil også kunne åpne opp for en ny innfalsvinkel til søking i et digitalt biblioteks-system. Det kan enten brukes som tillegg til query-baserte søk, eller som egen søke funksjon.

1.1.4 *Problemstilling*

Utifra den identifisering av problemet som er gitt over er problemstillingen konkretisert gjennom følgende spørsmål:

> *"Hvordan kan man knytte sammen ulike komponenter i samlinger på en slik måte at det vil kunne gi mening for sluttbruker, samtidig som det er prosseserbart av et digitalt biblioteks-system?"*

1.2 Dokumenter i nettverk.

1.2.1 *Lenking og dokumentformater*

Noe som gjennom tiden har vært et problem for forfattere, er hvilken forståelse mottaker av et verk vil ha av verket. Flere faktorer spiller inn her, type språk som er brukt, nivået på språket, utformingen av teksten, illustrasjoner, skriftstørrelse, referanser, osv. I trykket form har forfatteren hatt en viss kontroll med hvordan verket blir presentert, men med World Wide Web's (WWW) inntreden har de ulike verktøy for visning av dokumenter presentert disse på en måte som ofte ikke tilsvarer den utforming forfatteren ga det. Dette kan være katastrofalt for opple-

velsen av dokumentet og kan influere hvordan dokumentets budskap blir oppfattet. Særlig er dette viktig når man gjennom hypertextsystemer får muligheten til å legge inn lenker til bilder, lyd, animasjoner og andre dokument, slik at dokumentet blir dynamisk. Skulle disse innlenkede dokumenter forsvinne eller endre seg vil det gjellende dokumentet miste sin integritet.

Standardisering av format

En hypertext stiller krav til dokumentvisningen for at den skal opptrer slik forfatteren ønsket det, særlig under ulike hardware og software konfigurasjoner. For å ha en slik fremstilling trengs et universelt språk, eller en standard. Flere ulike språk for dokumentvisning finnes, men de fleste er proprietære og fungerer bare under et spesielt software- eller hardware miljø. Eksempler på dette er MS Word-format og PDF-format. For å unngå at man må bruke ett bestemt lisensiert produkt for å gjenvise et dokument har er det utviklet standardiserte format som kan brukes i flere miljøer.

SGML og WWW

SGML er en slik standard som gir forfatteren mulighet til strukturere dokumentersom oppfører seg likedan i alle miljø **[Oasis]** . Her kan brukeren opprette sin egen dokumenttype-definisjon (DTD) slik at dokumentet vil opptre på samme måte under forskjellige situasjoner. Eksempel på dette er noteark-definisjoner og definisjon for matematiske formler. SGML er ikke særlig tilrettelagt for et medium som WWW, da den har en meget kompleks struktur. To arkitekturer som har sprunget ut fra SGML har utmerket seg i forhold til WWW:

- *Hypertext Markup Language (HTML)* **[HTML4]** er en dokument-type definisjon gjort i SGML som er spesielt egnet for web distribuerte dokumenter. HTML tillater enkel dokument formatering, og har vist seg brukbar for sammensatte og interaktive dokument.

- *Extensible Markup Language* (XML) **[W3C-XML]** er den nye standarden som skal ta SGML prinsippene inn i den nye internett-verden. Under XML utvikles det en ny type lenkearkitektur kalt XLink som er sentral for de relasjoner som presenteres i denne avhandlingen.

1.2.2 *Dokumenter i digitale bibliotek.*

Begrepet digitalt bibliotek

Digitalt bibliotek er et meget brukt begrep som er vanskelig å definere. Momenter som inngår er *kvalitetsikring* og *langtidslagring*, og ikke minst bruk av teknologier som gir hensiktsmessige tilbakemeldinger til brukeren. Et system for digitale bibliotek skal hovedsakelig være et brukersentrert system, og i denne forbindelse

er gjenfinning av relevant informasjon viktig.

Utveksling og lagring av informasjon

En viktig del av det digitale bibliotek vil være utveksling og lagring av data, helst gjennom et standardisert format. Det har etter hvert kommet en del forslag til understøttende teknologi på dette området. En av de viktigste er XML **[W3C-XML]** , et format som i stor grad ser ut til å revolusjonere måten vi bruker nettverk på, både i forbindelse med datautveksling og datalagring. XML er meget fleksibelt og kan fungere som et slags rammeverk for all overføring og behandling av data, både for vanlige dokumenter og metadata. XML vil ha en sentral plass i denne oppgaven og vil bli nærmere gjennomgått i kapittel **2.6**.

1.2.3 *Gjenfinning av dokumenter i nettverk*

Den teknologi vi opererer med i dag tilbyr en mengde avanserte teknikker for søking og gjenfinning av informasjon. Det som kan virke problematisk med en del av de metoder og verktøy som man har er brukbarheten for sluttbruker. Noen verktøy bærer preg av et avansert nivå som gjør søke- og gjenfinning prossessen vanskelig uten forkunnskaper. Andre kan virke svært tilgjengelige men vil på grunn av sin enkelthet gi lav søkepresisjon og en stor gjenfunnet dokumentmengde (høy *recall)*. Resultatet av disse kan også ofte oppfattes som noe kaotisk. Ofte må man drive en "prøv og feil"-taktikk blant de lenker som er gjenfunnet, for å finne det man er ute etter.

Metadata i digitale bibliotek

Fra den trykte verden har det vært vanlig med store indeks-baser som inneholder metadata om dokumenter, katalogisert av kyndige personer som leser gjennom dokumentene. Denne virksomheten har sikret kvalitet og autentisitet av dokumenter.

Metadata på internett

En slik form for kontroll og katalogisering har man ikke hatt på internett. Det har derfor vært opp til forfatteren å sørge for at det gjeldende dokument har en virkelig autentisitet og kredibilitet. Med dagens mylder av dokumenter på internett, vil en kontrollert konstruksjon av metadata være nærmest umulig. Det er opp til forfatteren selv å lage metadata om sitt dokument, og dette må legges inn som en del av dokumentet (f.eks. meta-tag i HTML). Denne måten å tilorden metadata på, gir dårlig kontroll over hvorvidt metadataen er korrekt, og man mister noe av den kvaliteten som tidligere kunne forventes når en organisasjon med tyngde innenfor det gitte felt hadde akkseptert et dokument. Dokumentet må vurderes mot andre dokument av samme genre, eller som omhandler samme emne for å avgjøre sannheten av innholdet.

Indeksering av dokumenter på internett

Indeksering har til nå har vært foretatt av tunge *indekserings-roboter*. Dette er program som traverserer web-adresser (URL) og katalogiserer treff etter en enkel innholdsanalyse. Disse tar hensyn til innlagt metadata, men dette blir brukt som tillegg til en fulltekst-indeksering. Gjenfinning av dokumenter blir vanskeliggjort og viktige dokumenter kan ofte bli sidestilt av dokumenter som ikke har noen virkelig kredibilitet. Et friskt pust i denne sammenheng er *Google* som bruker referanser fra andre dokument til å angi relevans av et dokument.

1.3 Erfaringer fra systemer for lenking

Erfaringer fra tidligere systemer for lenker finnes særlig i to forskingsområder: *Hypertext* og *referanse-lenking*. Begge disse områdene har eksistert lenge og har en lang merritt-liste å vise til.

Lenker som del av dokumentet

Det som ofte har vært et problem med lenker er at de finnes som en del av et dokument. Denne situasjonen gir problemer i forbindelse med oppdatering av gamle lenker, innlegging av nye lenker, etc. Dette kan skyldes faktorer som at man ikke har adgangsrettigheter til dokumentet eller at ressurser blir slettet eller får nye tilholdssteder med ny adresse.

Utenforliggende lenker

Gjennom utvikling av modeller og systemer for lenking har man prøvd å finne mulige løsninger på denne situasjonen. Man har laget systemer for utenforliggende relasjoner som tar vare på den lenke informasjonen som er knyttet til et dokument. Disse relasjonsobjektene kan ligge i databaser styrt av andre organisasjoner som f.eks. utgivere og spesielle interesse organisasjoner. Slike systemer for relasjoner muliggjør også å lenke sammen komponenter som ikke har mulighet for å uttrykke relasjonsinformasjon. Dette gjelder media som bilder, animasjoner og lyder hvor informasjon består av en binærfil som oversettes til ikke-tekstelig informasjon. En annen mulighet ved et slikt system er at man kan bygge opp relasjoner mellom ressurser som går utover referanser og lenker tilhørende et dokument. Man kan la brukere lage egendefinerte lenker mellom ressurser, da særlig i forbindelse med forskning, men også i andre sammenhenger som f.eks. presentasjon-systemer for organisasjoner som museer o.l.

Distribuerte relasjoner

En har noen steder valgt å kalle et slikt relasjonssystem for et distribuert system. I dette ligger det at relasjoner ligger i egne lenke-samlinger utenfor de komponenter som det lenkes til og fra. *Distribuerte relasjoner* betyr utifra denne forklaring at

relasjonen ligger utenfor de komponenter som er med i relasjonen.

1.4 Oppbygning av hovedoppgaven

1.4.1 *Arbeid med oppgaven*

Arbeidet med dette hovedfagsprosjektet har vært todelt. Det ble lagt stor vekt på å lage en modell og implementasjon for lagring og transaksjon av semi-strukturert data. Den andre delen bestod i å lage en modell og implementasjon for relasjoner som var mulig å integrere i systemet for lagring. Der hvor komponenter i DigLib ikke eksisterte ennå, måtte det konstrueres prototyper på slike som var nødvendige for at relasjonsystemet skulle fungere. I andre tilfeller hvor komponentene utgjorde en for stor del, ble relasjonssystemet avgrenset. Det ble hele tiden lagt vekt på å utvikle modellen slik at den kan brukes på så bredt område som mulig. Hovedmålet for modellen er relasjoner mellom komponenter i DigLib, men modellen ble bygget for å kunne knytte relasjoner mellom alle typer komponenter som har en unik identifikator. Dette muliggjør relasjoner mellom web-ressurser.

Modell for DigLib

Det ble også utviklet en generell modell for DigLib slik at nødvendige komponenter kunne identifiseres og planlegges. Denne modellen og DigLib generelt er gjennomgått i kapittel **3**.

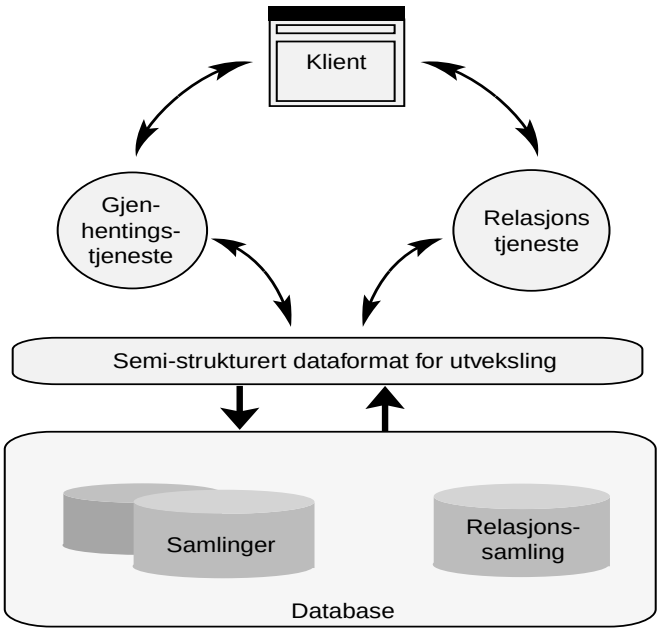
1.4.2 *Modell og prototyp*

Det blir i denne hovedfagsavhandlingen sett på noen av de teknologier som finnes for relasjoner, og en modell for et slikt system blir presentert. Det har også vært nødvendig å lage det system som komponenter lagres i, og selve lagringen har utgjort en stor del av arbeidet med prototypen. En tilhørende modell for lagring av komponenter blir også presentert. Systemet er laget med utgangspunkt i at det skal passe inn i DigLib.

Modellen som foreslås ser laget mest mulig generell og enkel slik at det kan bygges opp relasjoner mellom ulike komponenter og medietyper. Relasjoner ligger i objekter som kalles relasjonsobjekter og disse relasjonsobjektene vil kunne lagres i en egen relasjonssamling. Dette vil da ligge atskilt fra de samlinger som man har lokalt, og i første omgang benyttes til å knytte sammen ulike komponenter. Det er lagt vekt på konstruksjon av semi-strukturert lagringsmodell og modell for relasjoner. Grensesnitt mot sluttbruker er ikke utforsket i særlig grad, da et slik system som dette er tenkt å inngå i et overordnet søke-system.

I figur **1.4.2.1** er det satt opp en enkel modell over de komponenter som prototypen i denne avhandlingen involverer. Gjennom interaksjon med relasjonssystemet vil klienten få en adresse til et dokument i en samling som ligger i DigLib systemet. Klienten ber om å få tilsendt dokumentet og sender adressen til en gjenhentings tjeneste. Dokumentet sendes tilbake til klienten for gjennomsyn. For å finne flere dokument som har relasjoner til dokumentet kan klienten gi dokumentadressen til relasjonstjenesten som så gir tilbake en liste over de relasjoner dokumentet har

Fig 1.4.2.1 Interaksjon med relasjonssystemet.



Ved å bygge en relasjonstjeneste inn i DigLib-systemet kan en ved gjenhenting av dokumenter også motta de relasjoner som dokumentet har til andre ressurser lokalt og globalt. Dette vil være relasjoner som angir en retning og annen metadata om relasjonen. Slik kan en traversere seg gjennom dokumenter som har et forhold mellom seg, og håpet er at dette kan fremme oppdaging av ressurser i forhold til tradisjonell søking.

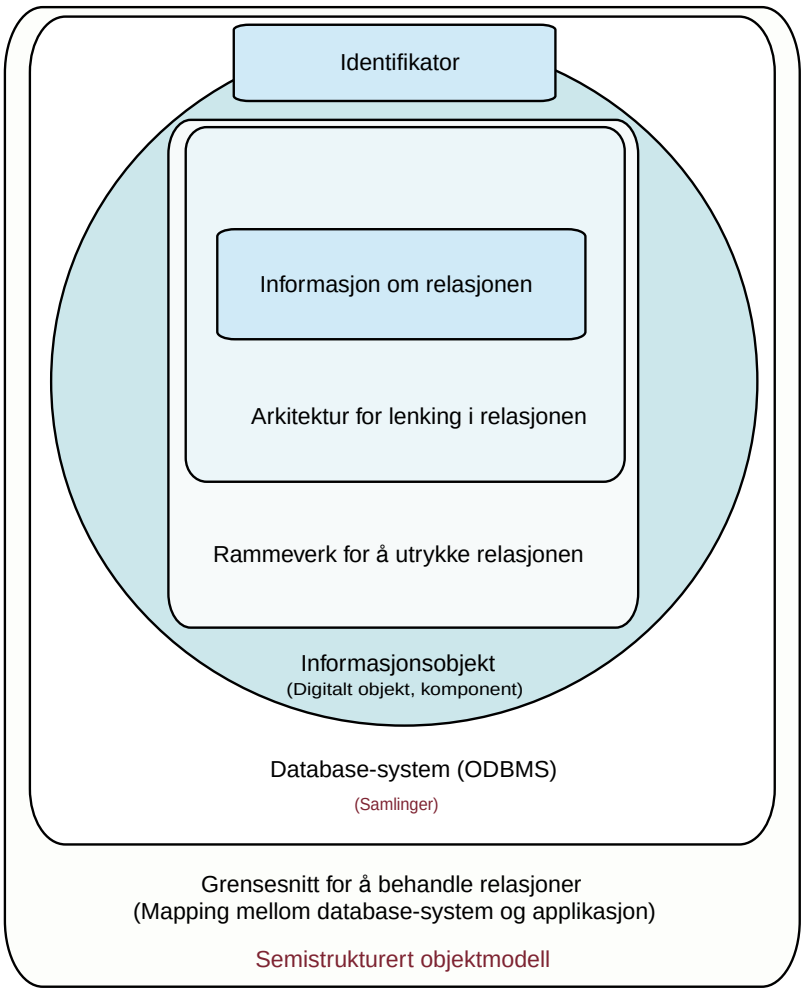
1.5 Arkitekturer i hovedfagsavhandlingen

I figur **1.5.0.1** summeres de arkitekturer som er belyst i denne hovedfagsavhandlingen. Modellen som vises her er lagdelt og ordnet slik at den samsvarer med modellen for relasjoner som blir presentert senere i denne oppgaven. En kan har

merke seg at modellen henviser til begrepet *informasjonsobjekt* som blir gjennomgått i kapittel **3.3**.

Modellen har to hovedgrupperinger: arkitektur for relasjoner og underliggende lagringsarkitektur. De komponenter som ligger innenfor informasjonsobjekt-sirkelen utgjør relasjonsarkitekturen, og det som ligger utenfor tilhører lagringsarkitekturen.

Fig 1.5.0.1 Arkitekturer i hovedfagsavhandlingen.



Konstruksjon og beskrivelse av relasjoner.

For å uttrykke relasjoner mellom informasjons-objekter trenger man komponenter for å beskrive en relasjon i et språk som er standardisert og utvidbart. Her følger en oversikt over de teknologier inngår på dette nivået.

- Arkitektur for lenker.
  Teknologier: HTML-links **[HTML4_Links]** , XPath/XLink **[W3C Xlink]** , Ad-hoc format.

- Et sett med metadata som uttrykker spesiell relasjons-informasjon utover selve lenken.
  Teknologier: XLink **[W3C Xlink]** , Dublin Core **[DC-Element_Set]** , Ad-hoc format.

- Et rammeverk for å uttrykke denne metadataen (relasjonene).
  Teknologier: XML **[W3C-XML]** .

- Lagringsmodell for å gjøre relasjoner persistente.
  Teknologier: Lagringsmodell (Kapittel 4) , XML **[W3C-XML]** .

Lagringsmodell for objekter.

Dette er teknologier som gir funksjonalitet i forhold til lagring og manipulasjon av data på tjener siden.

- Informasjonsobjekt.
  Teknologi: Digital objects **[Arms1997]** , Components **[Halasz1994]** , XML-strukturert modell **[Lichtenberg]** .

- Identifikator for å finne igjen objekter.
  Teknologi: CNRI's Handle system **[CNRI]** , DOI, URN, URI, URL.

- System for persistente objekter.
  Teknologi: Objektorientert database system : Versant ODBMS **[Versant]** , Ardent's O2 **[Ardent]** .

- Semi-strukturert språk for oppbygging av, og transaksjon med objekter.
  Teknologi: XML **[W3C-XML]** , Ad-hoc format.

Distribuert system.

Følgende komponenter kan brukes for å bygge et distribuert relasjons system :

- Format for datautveksling.
  Teknologi: InformasjonsObjekt **[Aalberg]** .

- Et system for distribusjon.
  Teknologi: CORBA **[CORBA]** , TSpaces **[Tspace]** .

1.6 Eksisterende systemer gjennomgått i Hovedfags-avhandlingen

Det blir sett på flere ulike typer systemer i denne hovedfagsavhandlingen. De som det legges mest vekt på er de to modellene: The Dexter Hypermedia Modell og Arms's Architectrue for Digital Libraries. Utover dette vil det bli sett på en del systemer som er intressante innenfor avgrensede områder.

Modeller for samlinger :

- The Dexter Hypertext Modell **[Halasz1994]** .
- Architechture for Digital Libraries **[Arms1997]** .

Systemer for referanse lenking :

- Distributed Link Service **[DLS1]** .
- Open Journal Project **[OpenJournal]** .

Systemer for ulike sider ved lenking :

- SFX **[VanDeSompel1]** .
- BibRelEx **[Landgraf1999]** .

Kapittel 2

Lenker, relasjoner og informasjon

Dette kapittelet tar for seg lenker, ulike lenkemekanismer og systemer for lenking. Følgende punkter blir gjennomgått:

- Hva er lenker og relasjoner?
- Lenke arkitektur for å uttrykke relasjoner.
- Lenke mekanismer.
- Lenke typer og relasjonstyper.
- Hypermedia/hypertext systemer: HTML og Dexter Hypertext model.
- Arms' Arkitektur for informasjon i digitale bibliotek.
- XLink lenkearkitektur i XML.
- Referanselenking og systemer: DSL og Open Journal.
- Andre typer lenkesystemer.
- Uttrykking av relasjoner.
- Kort om IFLA-modellen og relasjoner.
- Relasjonselementet i Dublin Core.

2.1 En nærmere forklaring på begrepene

2.1.1 *Hva er en lenke?*

En *lenke* er en forbindelse mellom en ressurs og en annen. En lenke har to *noder* og et forhold mellom disse nodene. Det tradisjonelle i blant annet HTML har vært at en kildenode peker til en målnode, som kan være en hvilken som helst web-ressurs (bilde, lyd, animasjon, tekst, program, hypertext, eller elementer i en hypertext) og angir en retning ved at kilde dokumentet *peker* på et annet dokument. I mer omfattende systemer for lenking kan denne retningen angis uavhengig av om lenken ligger i kildedokumentet, måldokumentet eller utenfor dokumentene.

2.1.2 *Hva er en relasjon?*

Begrepet *relasjon* blir i ofte brukt i sammenhenger hvor man snakker om lenketyper eller lenkeattributter. Relasjoner uttrykker et spesifisert forhold utover den informasjon en enkel lenke gir. Lenker har i den senere tiden utvidet seg til også å omfatte angivelser av semantikk. Relasjoner blir her brukt for å uttrykke en lenke med informasjon om sitt forhold. Lenker blir brukt om selve sammenknyttingen av entiteter. Det er i denne avhandlingen valgt å bruke begrepet relasjon på de lenkestrukturer som presenteres i kapittel **5**.

2.1.3 *Mekanismer for lenking*

Lenkemekanismer er ulike metoder som brukes for å knytte sammen informasjon. Disse metodene inngår i hypermedia begrepet, og i Jon Bosak's artikkel fra 1997 nevnes de klassiske lenkemekanismene **[Bosak1997]** :

- Location-independent naming (Identifikatorer).
- *Bidirectional links* (Lenker som kan spesifiseres og administreres utenfor dokumentet som de lenker fra/til).
- *N-ary hyperlinks* (En til mange forhold).
- *Aggregate links* (Flere kilder).
- *Transclusion* (Mål dokumentet for lenken utgjør en del av lenkens kildedokument).
- *Attributes on links* (Lenketyper).

Av disse mekanismene for lenking er det særlig *bidirectional links*, og *attributes on links* som vil bli undersøkt nærmere i forbindelse med prototypen av et relasjonssystem.

I tillegg til disse mekanismene har man også noen som inngår i de hypermediasystemer som gjennomgås i kapittel 2.2:

- *Unidirectional links* (Lenker som går en vei, motsetningen til bidirectional lenker)
- *Span-to-span links* **[Halasz1994]** (lenker fra et utsnitt til et annet utsnitt)

Attributes on links (Lenke typer)

En viktig komponent i lenkearkitekturen har vært angivelsen av *lenketyper.* Len-

ketyper legger semantikk til lenker mellom noder, og blir nærmere gjennomgått under kapittel **2.10.1**

Unidirectional Linking

Unidirectional links er lenker som peker i en retning. En kilde peker til en annen node som er mål for lenken. Enveis-lenking innebærer at den tillenkede ressursen ikke har kjennskap til lenken. Lenken er en del av kilden, og alle lenker går ut *fra* kilden. Eksempel på slike lenker er gjennomgått i kapittelet om lenking i HTML (kapittel **2.5**).

Bidirectional lenking

Bidirectional links er lenker hvor begge de tillenkede ressurser har kjennskap til lenken. Lenken kan ha en retning, men den semantiske forståelsen av lenken vil ligge i begge nodene. Dette løses enten ved at lenken legges inn i begge de tillenkede noder, eller ved å legge lenken utenfor nodene. Xlink standarden i XML benytter denne typen lenker, og dette blir sett nærmere på i kapittel **2.7.1**.

N-ary

N-ary lenker er lenker som lenker til flere komponenter. En komponent kan lenke til flere andre komponenter i samme lenken. Slik lenker tillater et “en til mange”-forhold i lenking.

Span-to-span lenker

Span to span lenker er lenker som lenker mellom utsnitt av et dokument. I tekst kan dette f.eks. være et avsnitt eller et ord **[Halasz1994]** . Slike lenker gir større presisjon. I HTML har man muligheten for å legge inn navngitte ankre i teksten som man kan lenke til, med dette er en meget begrenset måte å lage *span-to-span*-lenker. Meningen med slike lenker er at lenken vet om hvilket utsnitt det lenker fra, og hvilket utsnitt det lenker til. Denne typen lenker forekommer i Dexter-modellen **[Halasz1994]** , og i XPointer-spesifikasjonen for XML **[W3C Xpointer]** .

2.1.4 *Lenketyper fra referanselenking*

Fra forskning på referanselenking har man noen definerte lenketyper. Disse er som følger: **[Hitchcock1998]**

- *Citation linking.*
- *Content linking.*

- *Keyword links.*
- *User created links.*

Citation Linking

Lenker mellom et dokument som inneholder et sitat og det siterte dokument. Referanselenking blir gjennomgått i kapittel **2.8**.

Content links

Dette er lenker som er konstruert gjennom analyse av informasjon og bygger relasjoner basert på innholdet i informasjonen. Slike lenker krever at man har analyseverktøy (bildegjenkjenning, tekst-traversering, eller lydanalyse) som trekker ut ønsket informasjon, og slik kan bidra med å lage lenker mellom informasjon som har *lignende* innhold. Programmer for slik analyse kan enten være innebygd i systemet eller ligge som eksterne moduler.

Keyword links

Dette er lenker som knyttes gjennom nøkkelord i informasjon. Dette kan f.eks. være forekomst av ordet "Trondheim" i to dokumenter som således kan relateres. Dette har særlig hensikt når man knytter dokumenter opp mot ordlister og leksika, og hvis man ønsker å knytte metadata om ulike dokumenter sammen. Slike lenker kan genereres automatisk. **[Hitchcock 1997a]**

User created relations

Dette er lenker som brukere av et lenkesystem skaper. Slike lenker kan genereres ved å følge brukerens bevegelser, eller ved at brukeren selv legger inn lenken. Informasjon om hvordan brukeren snor seg fra dokument til dokument kan være meget verdifull informasjon. Lenkene kan forteller noe om hvilke dokumenter som ansees å være innenfor samme domene. Dette krever sofistikert programvare som spesielle *proxys*, og er en vanskelig prosess.

2.2 Hypertext og Hypermedia

2.2.1 *"As we may think"*

Vanevar Bush skrev rett etter krigen en artikkel som la grunnlaget for det vi i dag kaller hypertext. Bush innså at man trengte en ny type teknologi innenfor det å publisere, søke, forske og utveksle dokumenter i forbindelse med forskingen. Dette sprang ut ifra det faktum at slike behov oppstod under arbeidet med atom-

bomben. Bush pekte blant annet på viktigheten av systemer som kunne lenke sammen ulike ressurser. **[Bush1945]**

Nå flere ti-år etter Bushs epokegjørende uttalelser har vi et verdensomspennende nettverk hvor slik lenking spinner et nett så kaotisk at ingen har oversikt over det. Et behov for ny teknologi til å kunne gjøre all denne sammenlenkede informasjonen brukbar har oppstått.

2.2.2 *Hva er hypertext og hypermedia?*

Hypertext og hypermedia er to begrep som ofte brukes om samme definisjon. Generelt kan man si at hypertext og hypermedia er et system hvor ulike informasjonseneheter knyttes sammen gjennom lenker:

> *"Hypertext is the organization of information units into connected associations that a user can choose to make. An instance of such an association is called a link or hypertext link.*
>
> *Hypertext was the main concept that led to the invention of the World Wide Web, which is, after all, nothing more (or less) than an enormous amount of information content connected by an enormous number of hypertext links.*
>
> *The term was first used by Ted Nelson in describing his Xanadu system."* **www.whatis.com**

Forskjellen mellom hypertext og hypermedia ligger i at hypermedia utvider hypertext begrepet gjennom å ta hensyn til andre medietyper enn tekst. Dette kan være bilder, lyder og animasjoner.

En sentral del i et hvert hypermedia system er lenker, og en gjennomgang av noen slike systemer er naturlig for å få en forståelse av lenke-mekanismer og -struktur.

2.2.3 *Hypermedia modeller*

Her følger en kort gjennomgang av noen ulike hypertext/hypermedia systemer. Disse er tatt med fordi de presenterer noe interessante momenter i forhold til lenking. Følgende to modeller har vært av særlig interesse i forhold til hovedfagsprosjektet:

- The Dexter Hypertext Reference Model **[Halasz1994]** .
- Hypertext Markup Language (HTML) **[HTML4]** .

2.3 The Dexter Hypertext Reference Model.

Denne modellen utkom første gang i 1990, og senere i en nyutgivelse i 1994. **[Halasz1994]** Det er en gammel modell, men den har hatt betydning for videre forskning. Modellen tilbyr metoder for lenking som først er realisert gjennom Xlink standarden. Den ble til på en tid da HTML ikke var allment tilgjengelig. Dextermodellen har senere vært brukt i flere sammenhenger, blant annet i Amsterdam Hypermedia Model som legger tid og kontekst aspekter til hypermediamodellen. Her finner vi blant annet følgende forklaring på hypertext begrepet:

> *"...a hypertext is a network of components related through a set of links anchored in source and destination components"* **[Hardman1994]**

2.3.1 *Modell oppbygning*

Komponenter

Dextermodellen gjør bruk av begrepet *komponent* for å angi noder som kan adresseres innenfor hypermediasystemet. Dexter modellen angir tre typer av slike komponenter:

- *Atom.* Dette er enkle og selvstendige komponenter med intern struktur som ligger utenfor modellen.
- *Links.* Dette er entiteter som angir relasjoner mellom andre komponenter.
- *Composite components.* Disse komponentene består av andre komponenter, de er en slags superklasse som inneholder subbklasser av komponenter. Mao. er det snakk om sammensatte komponenter.

Identifikatorer

En hver komponent som finnes i hypertextsamlinger blir gitt en unik identifikator (UID) som angir adressen til komponenten.

Lagdeling

Disse komponentene lagres i et *storage layer*, som er et av tre lag modellen skiller mellom (**Fig 2.3.1.1)**:

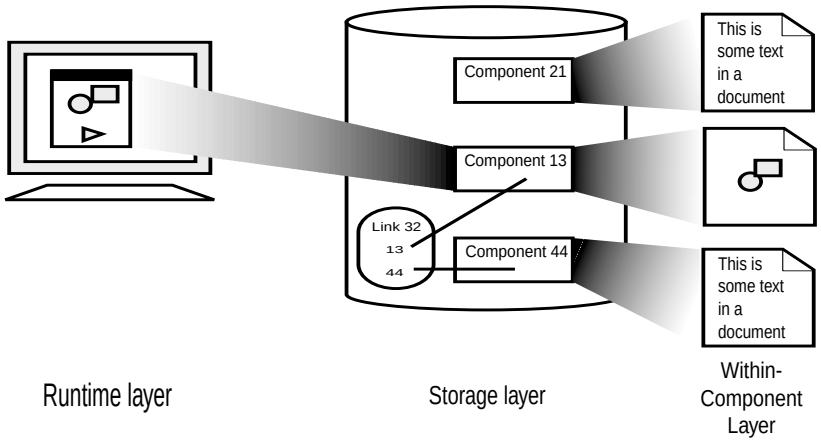
- *Run-time layer.* Dette laget sørger for aksess, visning og manipulering av hypertext. Denne delen ligger utenfor dexter modellen bortsett fra når det

gjelder lenking. Presentasjon av *span-to-span*-lenkede komponenter inngår her. Dexter modellen har muliggjort angivelse av presentasjons spesifikasjoner som kan kodes inn i *storage*-laget slik at komponentene vises på rett måte.

- *Storage layer.* Av disse tre lagene er dette kjernen i modellen. Dette laget inneholder de mekanismer som knytter lenker og lagrer komponenter. Disse komponentene blir behandlet uavhengig av innhold, slik at det ikke skilles mellom tekstlig og ikke tekstlig innhold i komponentene.

- *Within-component layer.* Dette laget tar for seg strukturen og innholdet til komponentene. Dexter modellen definerer dette til å ligge utenfor hypertekstmodellen. Et viktig aspekt ved dette laget er hvordan en knytter lenker til angitte steder inne i komponenten. En slik type lenking kalles *anchoring* og springer ut i fra ønsket om å tilby såkalt span-to-span lenking.

Fig 2.3.1.1 Dexter hypertext model - lag modell [Halasz1994] .



Aksessering av komponenter

I *storage layer* finnes to funksjoner som sammen er ansvarlig for gjenhenting av komponenter:

- *Resolver.* Denne funksjonen sørger for at hyperlenken aksesserer rett UID. Da lenker i dokumenter kan forandre seg ettersom dokumentet endres, trenger man en funksjon som kan "løse" (*resolve*) lenken inn i en UID.

- *Accessor.* Denne funksjonen tar for seg aksessering av komponenter utifra en gitt UID. Det dreier seg her om en gjenhentingsfunksjon.

2.3.2 *Lenking i Dexter modellen*

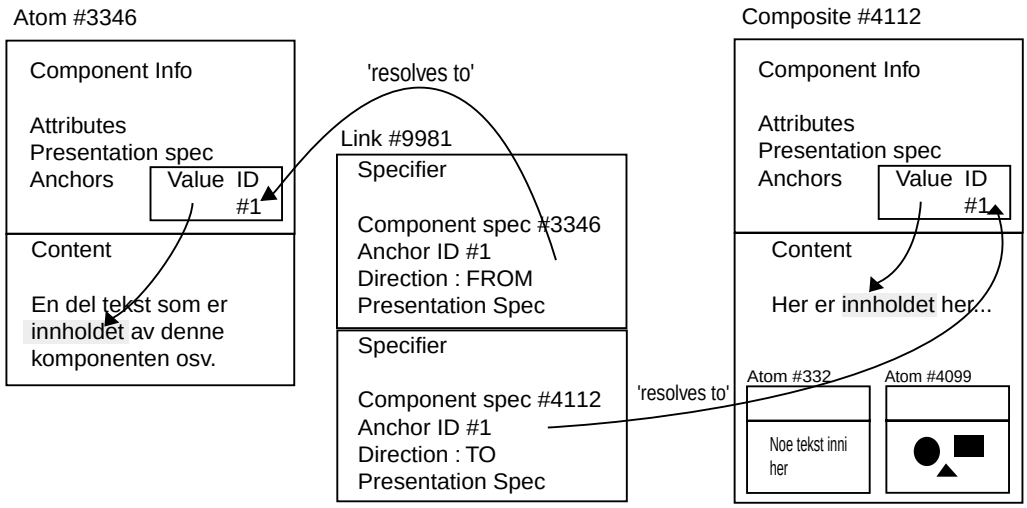
Dexter modellen opererer med noe som kalles *span-to-span* lenker. Dette er lenker som lenker mellom utsnitt av ulike komponenter. I tekst kan dette f.eks. være et avsnitt eller et ord. Slike lenker gir større presisjon på lenkeforholdet, gjennom å kunne spesifisere hva det er i komponenten som lenkes. Denne formen for lenking oppnås ved bruk av *anchor* som består av følgende attributter:

- *Anchor id.* Dette er en unik id som angir ankeret i komponenten.
- *Anchor value.* Denne "verdien" brukes til å angi en lokasjon, region eller del inne i komponenten, og er kun brukbar for det program som skal prosessere komponenten.

For å angi en lenke bygger man en komponent som består av en eller flere *specifiers*. En *specifier* inneholder:

- *Component spesification.* Denne angir en identifikator for komponenten (UID) som lenken spesifiserer.
- *Anchor id.* Denne angir en id innenfor komponenten.
- *Direction.* En retning angis ved å bruke følgende nøkkelord:
  - FROM. Denne angir at komponenten er kilde.
  - TO. Denne angir at komponenten er mål.
  - BIDIRECT. Denne angir at lenken går begge veier.
  - NONE. Denne angir at det ikke er noen traversering av lenken.
- *Presentation spesification.* Denne angir en spesifikasjon for fremvisning i run-time laget.

Fig 2.3.2.1 Modell for lagring og lenking i Dexter hypertext. [Halasz1994]



Modellen over (**Fig 2.3.2.1)** viser både hvordan de ulike komponentene "ser ut" og hvordan lenker spesifiseres og knytter sammen dokumenter. Man ser av denne modellen hvordan lenker er en egen entitet som ligger utenfor komponentene den lenker sammen. Noen av de ulike lenke mekanismene angitt i denne modellen vises i kapittel **2.1.3**.

2.3.3 *Kommentarer til Dexter hypertext model*

Modellen presenterer en del prinsipper som siden har gjenspeilet seg i blant annet Arms' modell for det digitale bibliotek **[Arms1997]** . Deler av denne blir gjennomgått i kapittel **4.2**, og en sammenligning av Arms' arkitektur og Dextermodellen er gjort i kapittel **6.2**.

2.4 Arms' arkitektur for informasjon i digitale bibliotek

Arms' artikkel fra 1997 **[Arms1997]** presenteres en arkitektur for informasjon i digitale bibliotek som har vært en viktig kilde for senere forskning. Det blir her gjennomgått de deler av arkitekturen som har relevans i forhold til denne avhandlingen, og det lagringssystem som presenteres i kapittel **4**.

2.4.1 *Det digitale biblioteks-systemet*

Arms setter opp fire ulike hovedkomponenter i et digitalt biblioteks-system:

- *User interface.* Dette er grensesnittet som presenteres brukeren.
- *Repository.* Her lagres og behandles *digitale objekt* og annen informasjon. Tilgang til informasjon skjer gjennom *Repository Access Protocol* (RAP).
- *Handle System.* Dette er et oppslags-system for identifikatorer på internett. Systemet tillater langtidslagring av *handles* som kan løses inn i faktiske adresser til *digitale objekt* og andre ressurser. Videreutviklingen av dette systemet er brukt i DigLib-systemet som presenteres i kapittel **3**,
- *Search System.* Dette er søkesystemer som brukeren søker i for å oppdage informasjon, og hente dette fra det *repository* det ligger i.

2.4.2 *Digitalt objekt*

Arms presenterer et rammeverk for digitale objekt hvor følgende beskrivelse blir lagt til grunn for begrepet:

> *"A digital object is a way of structuring information in digital form, some of which may be metadata, and includes a unique identifier, called a handle."* **[Arms1997]**

Et digitalt objekt er den fundamentale enhet i arkitekturen, og består av to deler:

- *Key metadata*. Denne delen inneholder informasjon som forteller hvordan det *digitale innholdet* av objektet skal behandles. Her finner vi også handelen til objektet.
- *Digital material*. Dette er innholdet av det digitale objektet.

Det digitale objektet har også en intern struktur som ligger i *digital material-*

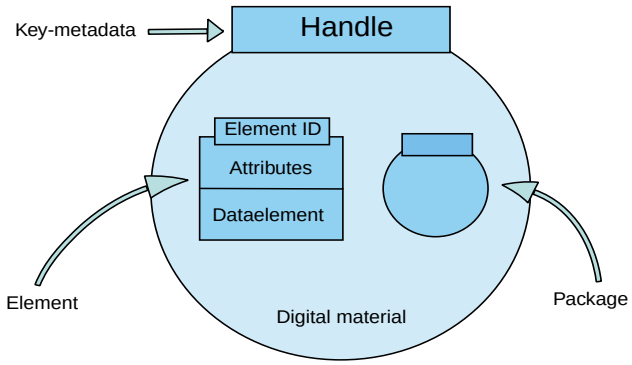
delen, og består av to forskjellig komponenter:

- *Element*. Et element har også en indre struktur, den består av en *element ID, attributes*, og et *data-element.* Element id er en universell id som kan brukes til å hente frem elementet. Attributtene forteller noe om hvordan elementet skal behandles, dette inkluderer *role* og *type*. *Data-element* er innholdet av elementet.

- *Package*. En slik pakke kan inneholde andre pakker og/eller et eller flere elementer. Et digitalt objekt er også en pakke.

Denne strukturen er vist i figur **2.4.2.1**. Her kan man se et digitalt objekt med nøkkeldata og et digitalt innhold bestående av et element og en pakke.

Fig 2.4.2.1 Digitalt objekt. [Arms1997]



For å oppsummere digitalt objekt begrepet kan vi si følgende:

- Et digitalt objekt består av nøkkel metadata og digitalt materiale.

- Et digitalt objekt er en pakke.

- En pakke har en pakke identifikator, og består av et eller flere elementer og/eller pakker.

- Et element består av en *element*-identifikator, attributter som gjelder elementet (rolle og type) og et data element som en bit sekvens.

Her brukes datadelen i et element til å inneholde en bit-strøm, dvs. det kan være et bilde, en word fil, eller hva som helst som er binært.

Lenking mellom digitale objekt

Arms beskriver to typer lenker som blir brukt for å lenke sammen ulike digitale objekt. En lenke benytter seg av handelen til det tillenkede objekt for å adressere objektet:

- *Child link.* Et digitalt objekt kan ha en ubegrenset mengde subbobjekter.
- *Parent link.* Et digitalt objekt kan ha et super-objekt.

Dette skillet mellom lenker ble ikke brukt prototypen på systemet, da det manglet fleksibilitet.

2.4.3 *Repository*

Repository er lagringsdelen av Arms' arkitektur. Denne består av en tre-delt modell:

- *Repository shell.* Dette laget består av en *repository access protocol* (RAP) som tar seg av interaksjon mellom indre og ytre objekt-struktur. RAP er protokollen for lagring, organisering og aksessering av digitale objekt.
- *Object management layer.* Dette laget er et grensesnitt mellom RAP og den underliggende databasen.
- *Persistent store.* Dette laget representerer er selve databasesystemet som ligger i bunn. Denne er skjult for brukeren som kun kommuniserer gjennom RAP.

Denne modellen av et repository er brukt som basis for oppbygningen av lagringssystemet i denne avhandlingen, som presenteres i kapittel **4**.

2.5 Hypertext Markup Language (HTML)

Foruten å være det medium som oftest brukes til presentasjon og interaksjon med bruker av et digitalt bibliotek, inneholder HTML også et system for lagring og lenking som er verdt å undersøke i lys av Dexter-modellen og lenker.

World Wide Web (WWW) ble opprinnelig utviklet som et informasjons rom, ikke bare brukbart for menneskelig kommunikasjon, men også for å gi maskinen mulighet til å delta og hjelpe. Et av de større problemene med dagens Web har vært at det meste som finnes i dette informasjons-rommet er ment for menneskelig tolkning. Tim Berners-Lee ved forskningssenteret CERN i Sveits som også er

grunnleggeren av WWW sier dette om denne problematikken (fritt oversatt til norsk):

> *"En mulig løsning på dette problemet ville være å trene maskiner til å tenke som mennesker, en annen og mye enklere måte er å konstruere et semantisk Web, hvor man utvikler språk for å uttrykke informasjon i en maskinprosesserbar form. Det semantiske Web kan sies å være et nettverk av data - ikke ulikt en global database.* **[Berners-Lee1]**

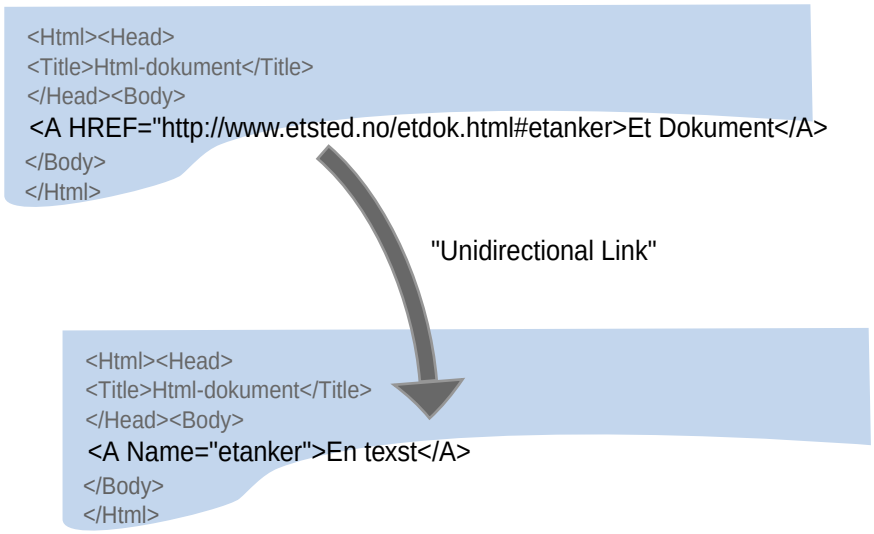
WWW benytter i hovedsak HTML og overførings protokollen HTTP for å tilby et hypertextsystem i stor skala. HTML ble utviklet på noenlunde samme tid som Dexter modellen og senere kjent under forskningsprogrammet MOSAIC, utviklet av NCSA. Grunnen til at det var denne modellen som tok over internett var den enkle og effektive løsningen dette systemet tilbød brukeren. HTML har siden vokst og utviklet seg til det som i dag er standarden for dokumentfremstilling på internett. Arkitekturen tilbyr lenking mellom HTML-dokument og mot andre dokument formater som bilder, lyder, tekstformater etc. HTML på tross av denne utviklingen ikke tilby en lenketeknologi i tråd med Dextermodellen.

2.5.1 *Lenker i HTML*

De lenkene man finner i HTML er tradisjonelt kjent som "unidirectional links" dvs. de går kun en vei fra det aktuelle dokument til en annen ressurs, og er ukjente for den tillenkede ressursen.

> *"A link has two ends -- called anchors -- and a direction. The link starts at the "source" anchor and points to the "destination" anchor, which may be any Web resource (e.g., an image, a video clip, a sound bite, a program, an HTML document, an element within an HTML document, etc.)"* **[HTML4_Links]**

Fig 2.5.1.1 HTML Anchor.



Man kan dele inn slike lenker i to typer. Disse typene er ikke definert i HTML, men man kan allikevel påpeke dem utfra de sammenhenger som lenken brukes i:

- *Eksterne* hyperlenker er lenker til andre ressurser som enten er av relevans til den gitte hypertext, eller som nevnes av andre grunner (for eksempel: “lenker til venner og kjente”, “ting jeg finner morsomme”, “her finner du et fint bilde” etc.)

- *Tilhørende* hyperlenker (Transclusion **[Bosak1997]**) er lenker til objekter som hører til innenfor den gitte hypertext. Dette er typisk bilder, lyder, animasjoner og tekster som tilhører eller utgjør en del av det aktuelle dokumentet (Composite components **[Halasz1994]**).

De hyperlenker som kan karakteriseres som *eksterne* er i HTML av typen **A** (Anchor) eller **LINK.**

Bruk av Anchor

A står for “Anchor” og oppfører seg noenlunde likt som *span-to-span* lenker i Dextermodellen, forskjellen er at her bygges lenken inn i dokumentet det lenker fra. Anchor i HTML har flere mulige attributter. Av disse er det følgende attributter som har relevans for lenke mekanismen:

- *HREF.* Angir en adresse som lenken “resolver”

- *Name* eller *Id*. Angir unikt navn på ankeret som kan brukes av andre ankre.

Utover disse attributtene har **A** også attributter for angivelse av informasjon om lenken, lenke type og tilsvarende attributter som tilhører relasjonsbegrepet. Man har ved hjelp av disse attributtene muligheten til å angi mer spesifikk data om den ressurs det refereres til. Særlig interessant er attributten *type* som beskriver innholdstype til den lenkede ressurs. Denne typen er angitt med *mimetypes* som er en standard for innholdsbeskrivelse. Man har i nyere versjoner av HTML også muligheten til å gi hver markup-tag en unik id som et anker kan henvise til. Utover dette har HTML-lenker ingen god mulighet til å angi lenketyper.

I eksempel **1** vises den mest brukte versjonen av anchor: “aktiver denne linken for å gå til denne ressursen.

Eksempel 1: “Anchor” lenke i HTML

```
<A HREF=”http://www.idi.ntnu.no>Institutt for Informatikk</A>
```

Bruk av Link

Link brukes i <HEAD> delen av HTML-dokumentet og beskriver generelle lenker til og fra dokumentet, det har ingen intern posisjon. Eksempel på bruk av link er vist i eksempel **2**.

Eksempel 2: “Link” lenke i HTML

```
<HTML>
<HEAD>
<LINK name="Prev" href="forrige.html">
<LINK name="Next" href="neste.html">
</HEAD>
```

2.5.2 *Muligheter og begrensninger for lenker i HTML*

Følgende momenter utgjør lenkemekanismen i HTML. **[W3C Xlink]** :

- Hyperlenken bruker URI’er som lokasjons teknologi.
- Hyperlenken blir uttrykt i en av dens to ender.
- Hyperlenken identifiserer den andre enden, men en tjener kan ha stor frihet når det gjelder å finne eller dynamisk konstruere destinasjonen til lenken.
- Brukere kan starte traversering bare fra den noden hvor hyperlenken er formulert, *til* den andre noden.

- Hyperlenkens effekt på vinduer, rammer, gå-tilbake lister, stylesheets, etc. er bestemt av brukeragenten og ikke av hyperlenken.

2.5.3 *Problemer med unidirectional links*

Typiske problemer med denne typen lenker kan en dedusere fra de problemene man har opplevd i forbindelse med HTML:

- Dokumenter opprettes og slettes i raskt tempo. Levetiden for en webside varierer kraftig ettersom nye versjoner av sidene opprettes, eller dokumenter slettes. Her kommer faktorer inn som f.eks. at URL endres ved endring av adresse til serveren.

- En lenkes gyldighet vil være tvilsom så lenge man lenker til dokumenter man selv ikke har kontroll over, da man ikke vet hvordan livsutsiktene til dokumentet er. Dette resulterer ofte i døde lenker.

- Lenking ut fra et dokument i en retning gjør det vanskelig for det tillenkede dokument å vite hvor det lenkes i fra. **[Hitchcock 1997a]** .

- Man kan ikke lenke til en spesiell del av et dokument uten å forandre dokumentet (ved å sette inn ankre). **[Harold1998] [Davis1999]**

- Brukere må ha rettighetene til en side, for å kunne lage lenker fra denne. **[Davis1999]**

2.6 Semi-strukturert språk - XML (Extensible Markup Language)

*E*xtensible Markup Language kom som *working draft* november 1996, og var et forsøk på å definere en ny standard som var bedre til å håndtere dokumenter på WWW, en det HTML var. XML er egentlig en enkel dialekt av den noe eldre "Standard Generalized Markup Language" (SGML) standarden for dokumenter, som definerer en formell måte å formatere et dokument på. Man kan si at XML er en enklere implementasjon av SGML, som åpner for muligheter HTML ikke kan gi blant annet gjennom dokument-type definisjoner og navnerom.

> *"XML is a technology that allows the creation of an unlimited number of different markup languages for different purposes"* **[XMLMyths]**

> *"Extensible Markup Language (XML) is descriptively identified as "an extremely simple dialect of SGML" the goal of which "is to enable generic SGML to be served, received, and processed on the Web in*

> *the way that is now possible with HTML," for which reason "XML has been designed for ease of implementation, and for interoperability with both SGML and HTML."* **[Oasis]**
>
> *"Extensible Markup Language, abbreviated XML, describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language. By construction, XML documents are conforming SGML documents."* **[Oasis]**

XML er med andre ord et markup-språk hvor man kan definere dialekter som feks. HTML. Dette er realisert gjennom definisjonen av XHTML.

Det er her sett på hvordan XML kan benyttes som et enkelt format/rammeverk for strukturering av metadata. Det vil settes spesielt fokus på Xlink definisjonen i XML i kapittel **2.7.1**.

2.6.1 *To typer XML-dokument*

Et XML dokument kan være i to former: *Velformulert* og *valid*. Forskjellen på de to formene ligger i at et velformulert XML-dokument holder seg til XML standarden uten bruk av en såkalt "Document Type Definition" (DTD). Et valid XML-dokument vil både være korrekt i forhold til XML standarden og en definert DTD. Det kreves at et hvert XML-dokument må inneholde et rotelement. Eksempel **3** viser et velformulert dokument, og eksempel **4** viser et valid dokument.

Eksempel 3: Velformulert dokument

```
<?xml version=”1.0” standalone="yes"?>
       <BOOK>
           <TITLE>Sometitle</TITLE>
           <AUTHOR>SomeAuthor</AUTHOR>
           <ISBN>0-7654-0141-0</ISBN>
       </BOOK>
```

Eksempel 4: Valid dokument.

```
<?xml version=”1.0” standalone="yes"?>
<!DOCTYPE BOOK document [
       <!ELEMENT TITTLE (#PCDATA)>
       <!ELEMENT AUTHOR (#PCDATA)>
       <!ELEMENT ISBN (#PCDATA)>
]>
<BOOK>
    <TITLE>Sometitle</TITLE>
    <AUTHOR>SomeAuthor</AUTHOR>
    <ISBN>0-7654-0141-0</ISBN>
</BOOK>
```

I eksempel **4** inneholder XML-dokumentet regler for elementene brukt i dokumentet. For å være valid må dokumentet følge disse reglene til punkt og prikke. Ved å ta i bruk valide XML-dokument kan man full kontroll over dataen som utveksles/lagres, men dette krever at man har en DTD som følger dokumentet.

Metadata i XML og Namespace

Disse eksemplene viser også hvordan en kan bruke XML som rammeverk for metadata. Direkte metadata i XML kalles for metadata i “Raw XML”. Problemet med dette er å uttrykke metadata i kjente formater som MARC eller Dublin Core ved bruk av XML. For å løse dette kan man ta i bruk noe som i XML blir kalt *Namespace*. Namespace gir muligheten for enkelt å kvalifisere elementer og attributter i XML dokumenter gjennom referanse til en URI. Gjennom bruk av namespace kan man blande ulike markup-språk. **[W3C-Namespace]** I eksempel **5** vises definering av et namespace i XML.

Eksempel 5: Eksempel på bruk av namespace

```
<element xmlns:ns=”URN:enplass:ns1”>
    <ns:Element>Element i navnerom en</ns:Element>
</element>
```

2.7 Lenking i XML

Det blir her gjennomgått den lenke arkitektur som er foreslått under XML-standarden beskrevet i kapittel **2.6**. Denne arkitekturen for lenking tillater alle de klassiske lenke-mekanismene som har blitt gjennomgått over.

Kilder til det følgende materialet er blant annet fra boka “*XML”* av E.R. Harold som ble skrevet under XLinks spede begynnelse. Lenke-arkitekturen i XML er

fortsatt under utvikling, og har tatt en større og større del av arbeidet med XML slik at W3C har opprettet en egen arbeidsgruppe for å se på denne arkitekturen. Det som presenters her bygger på en *working draft* av XLink, og arkitekturen kan forandre seg noe i tiden som kommer.

Lenkearkitekturen i XML deles inn i 3 ulike typer **[Cover99]** :

- X-Link. Denne gir lenking mellom ressurser både innenfor dokument og "out-of-line", dvs. gjennom egne lenk-dokumenter, og utgjør således selve lenke mekanismen.
- X-Path. Er et rammeverk for adressering som X-Pointer bygger videre på.
- X-Pointer. Gir mulighet for adressering av individuelle deler i et XML-dokument, slik at blant annet lenking av typen *span-to-span* muliggjøres (kapittel **2.1.3**).

2.7.1 *XML Linking Language (XLink)*

Xlink tillater kompleks lenking som blant annet gir muligheter for å lenke flere ressurser i samme lenke, og å legge lenkene utenfor ressursene. Dette står i kontrast til HTML hvor linkene må være innlagt i dokumentet og lenken kun har en retning. XLink bygger blant annet på Dexter modellen (kapittel **2.3**) og skal kunne: **[W3C Xlink]**

- Skape et lenket forhold mellom flere en to ressurser.
- Assosiere metadata med en lenke.
- Lage link-databaser som ligger utenfor de lenkede ressursene.

Xlink bruker syntaksen til XML, og for å kunne legge til rette for spesielle attributter/elementer må man brenytte seg av et såkalt *"namespace"*. Ved bruk av namespace kan man definere det attributtsett som Xlink bruker, som i eksempel **6**.

Eksempel 6: Angivelse av Xlink namespace

```
<mittElement    xmlns:xlink="http://www.w3.org/1999/xlink/name-
space">
...
</mittElement>
```

Xlink element typer

X-Link standarden definerer seks ulike typer av lenker gjengitt i tabell **1**. Disse

kan med unntak av “simple” brukes sammen etter visse regler. Sammen tillater disse lenketypene komplekse muligheter innenfor lenking.

Tabell 1: Xlink element-typer

Type	Beskrivelse
simple	Brukes til enkle “unidirectional” lenker som man har i f.eks.HTML.
extended	Gir mulighet for “bidirectional” lenker og lenker som ligger utenfor dokumentet ("out-of-line"). Denne gjør bruk av underelementene: locator, arc, resource, title.
locator	Data som angir, enten ved adresse eller identifikator, en utenforliggende ressurs som deltar i lenken. Brukes i sammenheng med *extended* lenketype.
arc	Spesifikasjon av regler for traversering av lenken, som retning og mulighet for å uttrykke traverserings-kontekst. Brukes i sammenheng med *extended* lenketype.
resource	En adresserbar tjeneste- eller informasjons-enhet. Dette kan være en fil, et bilde, dokumenter, programmer, søkeresultater, etc. Brukes i sammenheng med *extended* lenketype.
title	Angir en tittel. Brukes i sammenheng med *extended* lenketype.
“Xlink working draft” 21 februar 2000	

Xlink globale attributter

De ulike element-typene kan bruke et utvalg fra et attributtsett som er vist i tabell **2**.

Tabell 2: Xlink attributtsett.

Attributt	Beskrivelse
type	Angir lenke typen. (Simple, extended, arc, locator, resource, title)
href	Angir en identifikator.
role	Angir en rolle. Dette er en *parser*-avhengig attributt, og defineres ikke videre av Xlink.
title	Angivelse av tittel.

Tabell 2: Xlink attributtsett.

Attributt	Beskrivelse
show	Angir hvordan den tillenkede node skal vises. (feks. i et nytt vindu)
actuate	Angir hvordan lenken brukes, dvs. definering av oppførselen til lenken: onLoad, onRequest.
from	Angivelse av kilde. Brukt i lenker av type *arc*.
to	Angivelse av destinasjon. Brukt i lenker av type *arc*.

Simple links

Eksempel **7** viser hvordan man setter opp en enkel lenke i Xlink som samsvarer med en enkel lenke i HTML. Her kan man også se bruken av attributtene gjengitt i tabell **2**.

Eksempel 7: Simple link som i HTML

```
<xlink:simple
       xmlns:xlink="http://w3.org/1999/xlink/namespace/"
       href="etdokument.xml"
       role="enrole"
       title="en tittel på dokument"
       show="replace"
       actuate="onRequest" >
   En tekst som sier noe om lenken
</xlink:simple>
```

Extended og out-of-line lenker i Xlink

Extended links er lenker som går utover det HTML-lenker kan tilby. Her kan man blant annet sette opp *out-of-line* lenker, som er lenker utenfor dokumentene det lenker sammen. Eksempel på en out-of-line lenke er gjengitt i eksempel **8**. Den lenken som er beskrevet her tar i bruk *arc* for å angi en retning mellom de to adressene. Et element av type *extended* må inneholde minst et nøstet element av typen "locator". Legg merke til angivelse av roller i *locator* elementene. Ved å sette den ene til "kilde" og den andre til "maal" kan man danne en traverserings-rekkefølge mellom adressene. Beskrivelsen angir ingen form for informasjon om forholdet

utenom denne traverserings-rekkefølgen.

Eksempel 8: Extended link angivelse

```
<element xlink:type=”extended”>
  <adresse   xlink:type="locator" xlink:role="kilde" href="http://en.plass/">
  <adresse2 xlink:type="locator" xlink:role="maal" href="http://en.plass.2/">
  <retning  xlink:type="arc" xlink:from="kilde" xlink:to="maal">
</element>
```

Grupper.

Xlink muliggjør også angivelse av grupper av ressurser for en lenke. Slik kan man få konstruert N-ary type lenker (kapittel **2.1.3**).

2.7.2 *XML Pointer Language (XPointer)*

Xpointer teknologien tillater å spesifisere en bestemt plass eller et bestemt stykke (range) tekst innenfor et XML dokument **[W3C Xpointer]** . Dette kan ligne på *anchored text* i HTML og *span-to-span* i Dexter modellen, men XPointer trenger ikke å kjenne til innebygde ankre i det aktuelle dokument. Xpointer muliggjør således å lenke til innhold i et annet dokument ved angivelse av kjente holdepunkter i XML-dokumentet.

Det finnes to måter å peke til et segment av et XML dokument med Xpointer: *Absolutt lokasjon* og *relativ lokasjon.* Felles for begge er måten man adresserer på. I tabell 4 ser man en referanse til rot-elementet i et gitt XML-dokument. De delene av XML-dokumentet man vil ha tak i, adresseres etter den fullstendige URL/URI en til dokumentet atskilt med et nummertegn: #, eller et skilletegn: |.

Eksempel 9: Xpointer adressering med bruk av # eller |

```
A. http://www.some.com/tekst.xml#root()

B. http://www.some.com/tekst.xml|root()
```

For å angi en Absolutt lokasjon (“Absolute location”) har man følgende adresse-

rings termer som i tabell **3**.

Tabell 3: Absolute Location Terms

Term	Beskrivelse
root()	Angir rot-elementet og vil således angi hele dokumentet som lokasjon.
id()	Fungerer best når en har kontroll over både lenkedokument og dokument man lenker til. Id() angir et innebygd element med en gitt id fra forfatter.
html()	Velger et anker i HTML dokument (tatt med for å være bakoverkompatibelt).
origin()	Setter det elementet man er ved i en eventuell prosessering. (current element).

Skal man kunne angi Relative Location Terms går man utfra absolutt lokasjonsterm, og bruker så en av de lokasjonstermene som er vist i tabell **4**.

Tabell 4: Relative lokasjons termer

child()	Velger første barn-element under kilde elementet.
descendant()	Velger alle barn.
ancestor()	Søker etter alle tidligere foreldre: root().descendant(2, born).ancestor(1).
preceding()	Velger første forekomst *før* det spesifiserte elementet, uten å holde seg til hierarkiet.
following()	Velger første forekomst *etter* det spesifiserte elementet, uten å holde seg til hierarkiet.
npsibling()	Velger det element som kommer *før* current-elementet i under samme parent-element.
fsibling()	Velger det element som forekommer *etter* current-elementet i / under samme parent-element.

Bruk av lokasjon i XML-dokumenter er vist i eksempel **10**. Denne velger første

forekomst av elementet person etter rot-elementet i XML-dokumentet

Eksempel 10: Eksempel på lokasjon i dokument.

```
http://www.some.com/tekst.xml|root().child(1,person)
```

2.8 Systemer for referanse-lenking

Referanse-lenking er interessant i forbindelse med denne avhandlingen, da dette dreier seg om å lage lenker som ligger utenfor dokumenter. Det er her snakk om lenker mellom referanser, og refererte dokumenter. Referanselenkings-forskningen har også gått utover referanser til å dekke andre typer lenker **[Caplan1]** . Enkelte av prosjektene som har vært viktige på dette området blir her tatt opp. Dette gjelder særlig *Distributed Link Service* (DLS) og *Open Journal Project*.

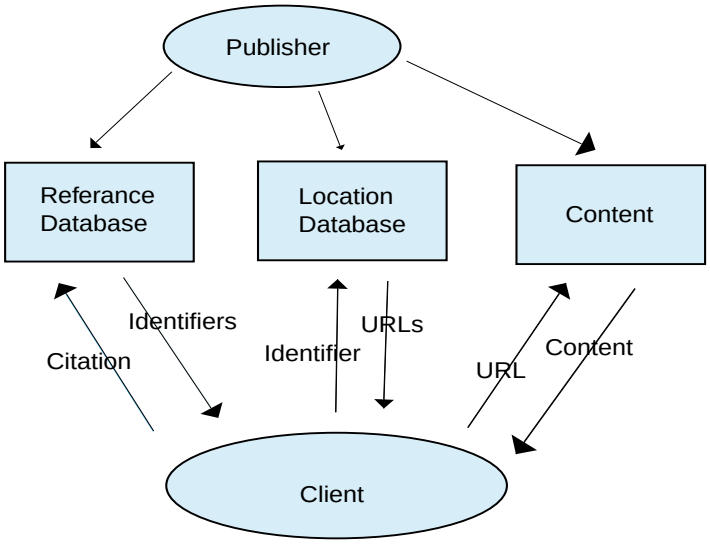
2.8.1 *Hva er referanselenking?*

Referanselenking oppstod som fenomen ved innførselen av elektroniske tidskrift. Denne typen lenking går ut på å bruke de referanser som forekommer i en artikkel til å knytte dette sammen med de refererte artiklene. Dette skaper nye måter å finne informasjon, da forskning ofte foregår ved å lete i referanselistene til artikkler.

2.8.2 *Generell modell for referanselenkings systemer*

En generell beskrivelse av hvordan et system for referanse-lenking fungerer er presentert i figuren under (**Fig 2.8.2.1)**. Denne er hentet fra en artikkel av P. Caplan og W.Y. Arms **[Caplan1]** . Figuren forsøker å sammenfatte hva de ulike systemene gjør, og hvilken arkitektur som ligger til grunn.

Fig 2.8.2.1 Referanse-lenking [Caplan1]



Hver ressurs har en unik identifikator og en eller flere kopier med hver sin URL/ URI. Metadata om ressursen blir laget av utgiver. Klienten henter frem disse ressursene gjennom interaksjon med ulike databaser. Modellen består av tre databaser:

- *Referanse-database* inneholder metadata om sitater og identifikatorer til kopier av referansene.
- *Lokasjons-database* inneholder URL'er for de enkelte identifikatorene som angir de ulike steder referansen kan hentes fra. Man har valgt å kalle dette identifikatorens resolusjon. Etter å ha funnet en URL til den aktuelle referansen, kan innholdet hentes ned.
- *Innholds-database* (samling) inneholder selve dokumentet.

Det er i dette systemet åpnet for at klienten vil velge aktuell URL utifra kriterier som adgangsrettigheter, kostnad, nærhet osv.

Felles for disse systemene er at de for det meste har konsentrert seg om hvordan en kan finne og trekke ut sitater i artikler, og koble disse opp mot referanser. Det har også vært satt en del fokus på metadata omkring referanser, og hvilke sett man skal bruke. Man har og prøvd å sette dette inn i metadatasettet til Dublin Core, uten å lykkes.

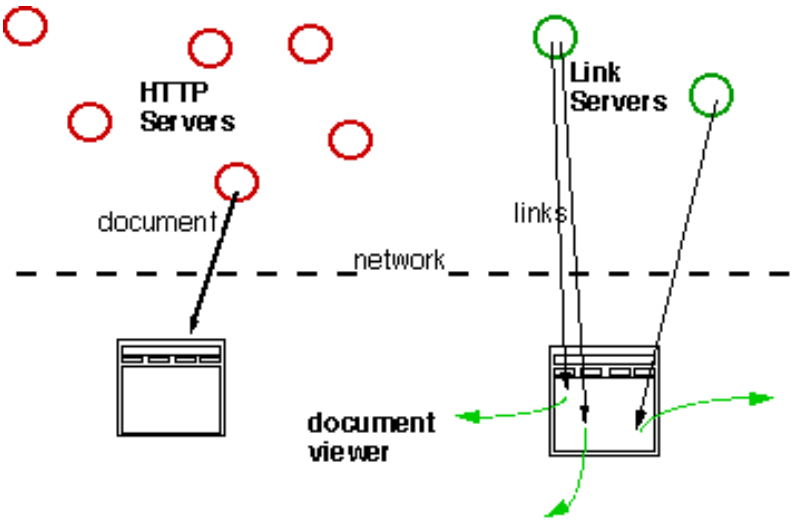
2.8.3 *Distributed Link Service (DLS)*

DLS er et lenke system som legger lenkene utenfor selve dokumentet. Det tillater en klient å koble seg til en lenke tjener og hente ned lenkeinformasjon som så kan legges inn i dokumentet lenken tilhører og vises til brukeren.

DLS bygger på *microcosm* som var en modell for "open hypermedia" laget ved University of Southampton. Dette systemet er nokså gammelt i denne sammenheng, og var et av de tidlig forsøk på å aksessere og integrere informasjon fra store og dynamiske datasett i et distribuert, heterogent system. **[Davis92]** .

Første versjon av DLS kom i 1995 og ble brukt hovedsakelig innenfor ERCIM WWW Working Group (W4G). Det har siden vert tatt i bruk av flere prosjekter, blant annet Open Journal Prosjektet. **[Hitchcock1998]** Tanken er at dette skal kunne brukes ved siden av det tradisjonelle HTML formatet. I figur **2.8.3.1** vises en modell fra Leslie Carr's artikkel om DLS. **[DLS1]**

Fig 2.8.3.1 DLS modell. [DLS1]



Modellen beskriver DLS-systemet. Klienten henter ned dokumentet fra en HTTP server, og henter ned lenke informasjon fra en lenke server. Disse vil da settes sammen, typisk av et CGI-script, og serveres brukeren. På denne måten kan en tilby lenker ikke bare inn i, men også utifra dokumenter som tradisjonelt ikke har en egen definisjon for lenker. Dette betyr at man kan legge inn lenker fra et HTML dokument til et bilde, og en lenke fra bildet ut igjen til en annen ressurs.

Systemet er basert på kommunikasjon via HTTP protokollen, hvor lenkeinforma-

sjon blir kodet inn i URL'en. Tjeneren er en rekke av CGI script som kjører på en standard server. Serveren har en hoved lenkedatabase og flere tilleggsbaser som brukeren kan velge. Tilleggsbasene tilbyr andre typer lenker en det hoveddatabasen gjør. Man har valgt å innføre begrepet "context" om de ulike lenkesettene i databasene. I tillegg har hver bruker en egen database hvor lenker som brukeren ønsker å ta vare på kan lagres. Lenkeinformasjon lagres i SGML, og tar vare på informasjon om adresse til kilde og destinasjon, type av lenke, tid når lenken ble opprettet og lenke beskrivelse.

Eksempel 11: DLS lenke beskrivelse [DLS1]

```
<link type=local>
<src><doc>http://diana.ecs.soton.ac.uk/~lac/cv.html
    <offset>
    <sel>Microcosm
<dest><doc>http://bedrock.ecs.soton.ac.uk/
    <offset>
    <sel>The Microcosm Home Page
<owner>Les@holly
<time-stamp>Fri Mar 31 13:32:34 GMT 1995
<title>Hypermedia Research at the University of Southampton
```

Hver adresse blir beskrevet gjennom tre elementer: adresse til ressurs, offset (sted) i ressursen og et valgt objekt i teksten. Offset ignoreres vanligvis i systemet.

Lærdommer fra DLS

En av de begrensninger man har med DSL er at det er basert på HTML og HTTP-protokollen. Dette begrenser distribusjon av lenker. DLS bruker et metadatasettet som ikke bygger på noen standard, men er en proprietær implementasjon som har brukt noen av prinsippene fra Hytime **[HyTime]** .

2.8.4 *The Open Journal Project*

Generelt om Open Journal

Open journal prosjektet var et prosjekt finansiert av eLib - det engelske programmet for digitalt bibliotek (UK Electronic Libraries programme). Prosjektet hadde en varighet på 3 år fra 1995 til mai 1998. Hovdemålet til prosjektet er sitert under:

> *"To build a framework for publishing applications enabling journals on the Web to be interlinked in ways which build on the traditional qualities and identities of the journals, and which increases the readers' ability to follow, search and access the literature for themed study and research using the maximum*

available online resources." **[OpenJournal]**

Open Journal prosjektet var et av de første prosjektene som tok for seg et rammeverk for referanse-lenking **[Hitchcock1998]** . Prosjektet bygde på DLS' teknologi for lenking. Målet var å bygge et system som tok i bruk lenker til å knytte de ulike deler (artikler) i elektroniske tidskrift sammen. Hitchcock nevner en del viktige momenter ved lenker i forbindelse med Open Journal:

> *"Links are important for a number of reasons:*
>
> *For users, links provide faster, more direct access to more information.*
>
> *For librarians, links support more effective information retrieval, especially from large archives, and can help with identified user phenomena such as "successive search episodes" [...]*
>
> *For publishers, links add value to works [...], but in this context links need to be applied in particular ways to make it easier to maintain and manage large numbers of documents."* **[Hitchcock1998]**

Det som er interessant her er påstanden om at et linksystem kan gi en høyere gjenfinningsrate i forhold til tradisjonelle systemer.

Lenketyper

Open Journal tok i bruk tre lenketyper. Disse er gjennomgått i kapittel **2.1.4**:

- *Citation linking* er lenker basert på sitater og deres tilhørende referanse.
- *Keyword linking* er lenker basert på forekomst av nøkkelord i teksten.
- *PDF linking* er identifisering av lenker i PDF-dokumenter.

Hva lærte man av Open Journal?

Ved en brukertest av systemet fant man en del begrensninger ved de ulike lenketypene. *Citation linking* gjorde seg innenfor rammen av et noe begrenset felt, men krevde en stor dokument-mengde for å gi noen effekt på gjenfinningsprosessen. *Keyword linking* viste seg å ikke fungere tilfredstillende i store dokumentmengder fordi man etter hvert fikk enorme mengder lenker fra et dokument. Systemet ble mettet av lenker og ikke særlig navigerbart. Man fant det hensiktsmessig å bruke *keyword linking* for å f.eks. knytte enkeltord i et dokument opp mot et online leksikon eller en ordbok.

2.9 Nyere Systemer for referanse-lenking

2.9.1 *SFX (Special Effects)*

SFX er et system for lenking utviklet ved University of Ghent i Belgia. Dette er et nytt system for referanse-lenking, som tar i bruk andre prinsipper enn DSL og Open Journal. SFX fokuserer ikke bare på rammeverk for lenking, men også på hvordan dette presenteres til bruker:

> *"The aim of SFX is to provide extended services in the hybrid library environment. The goal is to present information to the user in the context of the entire collection that is available."*

Arkitekturen som ligger bak lenkene har fått navnet "just-in-time" (JIT) lenking. Systemet benytter seg ikke av en sentral database med ferdig formaterte statiske lenker, men tar i bruk en form for dynamisk lenking som gjør systemet skalerbart. Det foreligger en del problemer med denne type lenking, som løses av brukergrensesnittet. SFX innfører begrepet konseptuelle lenker, som er forventede lenker som biblioteket antar vil være ønsket av dets brukere. Man har en gruppe lenkekilder med et tilhørende sett med konseptuelle lenker. Lenkene blir regnet ut når brukeren spør etter dem.

2.9.2 *BibRelEx - visualisering av lenking*

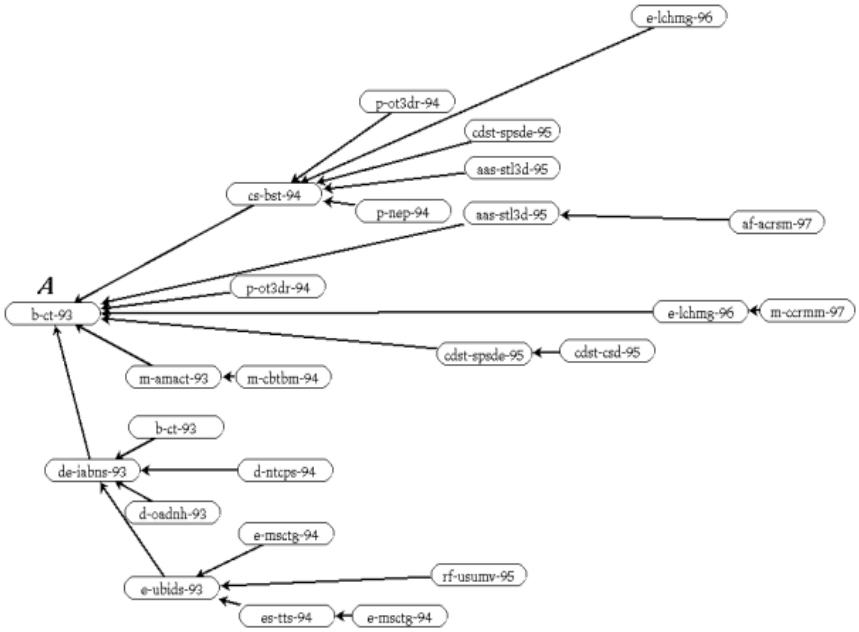
BibRelEx er et system for visualisering av referanselenker i et tre-lignende grafsystem. Det tilbyr en del muligheter når det gjelder å finne ut informasjon om tendenser og gruperinger innenfor et nett av lenker.

Hovedmålet med BibRelEx er å konstruere nye metoder ved visualisering av innholdsbaserte relasjoner **[Landgraf1999]** . I dette hensende er referanselenker nyttige for å oppdage feks. artikler om undersøkelser, fordi de siterer flere andre artikler innenfor et gitt fagområde. Fundamentale og viktig artikler innenfor et fagområde vil være sitert av mange. Også tematisk like artikler kan skilles ut, da disse vil ha en tendens til å ha samme referanser.

I figur **2.9.2.1** kan man se hvordan en visualisering av relasjoner kan gi verdifull informasjon. Hvis man tar utgangspunkt i dokument A kan en raskt se det som kalles A's "sphere of influence". Dette utgjør alle de dokumenter som A har påvirket.

Dette systemet foreslår også bruk av merknader som kan heftes ved dokumentet eller selve relasjonen. Disse kan legges inn av ekspert brukere, men også vanlige bruker vil få muligheten til å bidra på denne måten.

Fig 2.9.2.1 Modell fra BibRelEx: A's sphere of influence.



2.10 Beskrivelse av relasjoner

2.10.1 *Begrepet relasjonstyper*

Relasjonstyper er det Bosak angir som "attributes on links" eller lenketyper. Dette er metadata som angir informasjon om forholdet mellom to komponenter.

2.10.2 *IFLA - Relationships*

IFLA-modellen beskriver relasjoner som forhold mellom entiteter. Man har her relasjonstyper som: "konkretiserer", "spesifiserer" "utvider", "del av", etc. IFLA-modellens bruk av relasjoner knyttes til bibliografiske lister, og vi finner følgende forklaring på begrepet relasjon:

> *"In the context of the model, relationships serve as the vehicle for depicting the link between one entity and another, and thus as the means of assisting the user to "navigate" the universe that is represented in a bibliography, catalogue, or bibliographic database. Typically the user will formulate a search query using one or more attributes of the entity for which he or she is searching, and it is through the attribute that the user finds the entity sought. The relationships reflected in the bibliographic record provide additional information that assists the user in making connections between the entity*

found and other entities that are related to that entity." **[IFLA98]**

I IFLA modellen kalles nodene som lenkes sammen for entiteter. Relasjonene kan modelleres i entity-relationship modeller. Det er definert fire ulike typer entiteter: *Work, Expression, Manifestation, Item*. Mellom disse har IFLA-modellen definert en stor mengde relasjonstyper. Denne mengden av relasjonstyper er så omfattende at det er valgt å ikke ta dem i bruk. Målet med relasjonsmodellen er å ha et minimalt sett med relasjonsbeskrivelser, og i kapittel **2.10.3** blir et enklere relasjonssett presentert.

2.10.3 *Dublin Core relation*

Dublin Core (DC) er et metadatasett bestående av 15 basis elementer **[DC-Qualifiers]** . Disse elementene kan videre ved å bruke formen `DCelement.utvidelse`. Man kan gjennom slik utvidelse sørge for kompatibilitet med ulike implementasjoner da man alltid kan forholde seg til basis elementene. Disse basiselementene blir ikke tatt opp her da dette ligger utenfor avhandlingen.

Elementet *relation* utpeker seg særlig i denne sammenheng, og er foreløpig definert som eksprimentell av DC's arbeidsgruppe. Dette metadataelementet definerer noen realsjonstyper som er gjengitt i tabell **5**. Disse typene er definert utfra retning, det vil si at ved versjonsrelasjon er det definert to typer: "er versjon av" og "har versjon". Modellen for relasjoner som presenters i denne avhandlingen baserer seg på grunnstammen av disse relasjonstypene : *Version, Replaced, Require, Part, Reference*, og *Format*. Forklaring på bruk av relasjonstypene er hentet fra et working draft datert året 1997 **[DC-Relation] .** Det nyeste forslaget fra 2000 har utvidet og forandret dette settet noe slik at det forekommer som i tabell **5**.

Tabell 5: Relasjonstyper i Dublin Core (09.05.00) [DC-Relation]

Qualifiers	Forklaring
Is Version Of Has Version	"Version relations are those in which one resource is an historical state or edition of another resource by the same creator."
Is Replaced By Replaces	Ny i 2000. Forklaring utover relasjonstypen er ikke gitt. Denne er tatt inn i stedet for Is Based On/Is Basis for.
Is Required By Requires	"Dependency relations are those in which one resource requires another resource for its functioning, delivery, or content and cannot be used without the related resource being present."

Tabell 5: Relasjonstyper i Dublin Core (09.05.00) [DC-Relation]

Qualifiers	Forklaring
Is Part Of Has Part	"Part/Whole relations are those in which one resource is a physical or logical part of another."
Is Referenced By References	"Reference relations are those in which the author of one resource cites, acknowledges, disputes or otherwise refers to another resource."
Is Format Of Has Format	"Format transformation relations are those in which one resource has been derived from another by a reproduction or reformatting technology which is not fundamentally an interpretation but is intended to be a representation."

Kapittel 3

Digitalt Biblioteks prosjekt (DigLib)

Det digitale biblioteksprosjektet (DigLib) ved institutt for datateknikk og informasjonsvitenskap (IDI) inngår som en del av studieretning for informasjonsforvaltning. Målet for prosjektet er å utvikle en komponentbassert arkitektur for interoperable digitale bibliotek. Prosjektet skal tilby studenter ved IF muligheten til å forske og utvikle komponenter og arkitekturer etter eget behov og ønske. **[Aalberg]**

Den modellen for DigLib som presenteres i dette kapitlet har vært utarbeidet som en del av denne avhandlingen. Kapittelet gir en grunnleggende innføring i de ideer og komponenter som DigLib består av med hensyn til det som presenters i denne avhandlingen. For en mer inngående presentasjon refereres det til Trond Aalberg's artikkel "IDI DIGLIB: Komponentbasert arkitektur for forskning og utvikling av digitale bibliotek".

Følgende punkter blir gjennomgått her:

- Historie
- Begreper i DigLib modellen
- Komponentmodell for DigLib
- Informasjonsobjekt
- Database og transaksjons modell.
- Samlinger i DigLib
- Distribusjon i DigLib

3.1 Historie

Prosjektet ble påbegynt våren 1999 og under den påfølgende sommeren ble det satt i gang utvikling av, og forskning på, kommersielle og egenutviklede komponenter for å legge et grunnlag for videre forskning blant hovedfag og doktorgrad studenter. Mye av dette arbeidet inngår som en del av denne hovedfagsavhandlingen. I dette arbeidet var det særlig databasesystemer og ulike modeller for hvordan informasjon skal

lagres og utveksles som det ble sett nærmere på. Det ble utviklet en modell basert på semi-strukturert data, og denne ble utprøvd på noen samlinger. Denne modellen er videreført og presenters i kapittel **4**.

3.2 Oversikt over DIGLIB.

3.2.1 *Den generelle modellen for DIGLIB*

I forbindelse med DigLib-prosjektet har man vært nødt til å innføre en del begreper for å beskrive de ulike deler av et slikt system:

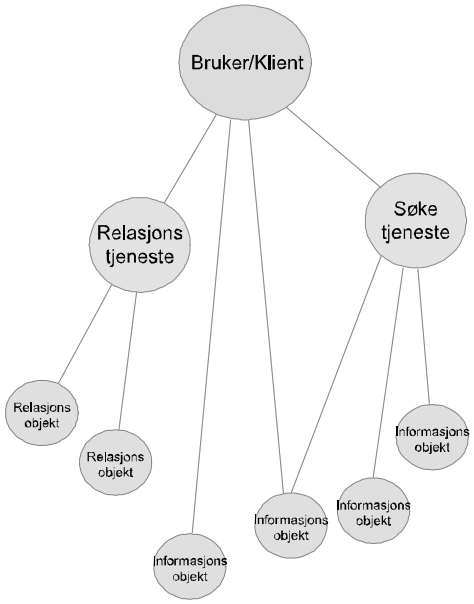
- **Klient.** Dette er den del av systemet som kommuniserer med sluttbruker. Dette kan bestå av en interaktiv web-side, et java-program eller en spesialbygd applikasjon.

- **Tjeneste. Dette** er den del av systemet som fungerer som innfalsport til ulike måter å hente frem informasjon.

- **Informasjonsobjekt.** Dette er de enhetene som representerer informasjon. Dette er råvaren i det digitale bibliotek.

I figur **3.2.1.1** er det illustrert en relasjonstjeneste ved siden av en søketjeneste som sammen gir mulighet for gjenhenting av informasjonsobjekt. Relasjonstjenesten er det system som presenteres i denne avhandlingen (kapittel **5**). Legg merke til at det her skilles mellom relasjonsobjekt og informasjonsobjekt selv om relasjonsobjekter også er informasjonsobjekter. De er gitt ulike navn for å skille mellom informasjonsobjekter i samlinger, og informasjonsobjektene som knytter relasjoner mellom dem. Modellen viser hvordan et system for relasjoner forholder seg til den interne struktur og resten av verden. Informasjons-objektene er løse ressurser som er lokalisert et sted på internett. De trenger ikke å være en del av de samlinger som finnes lokalt.

Fig 3.2.1.1 Forholdet mellom klient, tjener og informasjonsobjekt i DigLib



3.2.2 *Komponentene i DIGLIB.*

I figur **3.2.2.1** er de ulike hoved-komponentene DigLib-systemet består av, satt opp. Modellen viser hvordan DigLib-systemet er per dags dato, og tar kun hensyn-til de komponenter som utgjør kjernen av systemet, og de komponenter som har inngått i arbeidet med denne avhandlingen. Det er ikke tatt hensyn til komponenter som er under utvikling av andre studenter. Systemet er ment å være komponent-basert, dvs de ulike komponenten kommuniserer mellom lagene med standardiserte protokoller. Modellen består av syv lag som gradvis går fra menneske til maskin. De syv lagene er:

- *Bruker-grensesnitt* - Dette laget representerer de komponenter som brukeren forholder seg til. Dette kan være web-siden i web-leseren, eller et dedikert program som kjøres på brukerens maskin (klient).

- *Samlings-tjenester* - Dette laget inneholder de komponentene som formidler informasjon fra de ulike samlingene. Dvs. tjenester som gir tilgang til databasene gjennom en spesifisert protokoll.

- *Tjenste* - Dette laget inneholder de komponenter som formidler informasjon mellom klienten og det øvrige DigLib. Tjenestene som plasserer seg i dette laget er ikke tilknyttet spesifikke samlinger. F.eks.: en handle tjeneste.

- *Distribusjon* - Dette laget representerer de komponenter som tar seg av distribusjon (utveksling) av data. Herunder er det særlig CORBA **[CORBA]**

som er valgt for å formidle informasjonsobjekter.

- *Database-Integrering* - Komponentene i dette laget er selvutviklede bibliotek for databasesystemene. Disse tar seg av innleggelse og uthenting gjennom en egendefinert protokoll. Her har man standardisert på XML **[W3C-XML]** som dataformat for utveksling av tekstelig informasjon. Disse komponentene vil variere etter de ulike behov som en samling/et prosjekt har og skal hovedsakelig kommunisere gjennom en av de protokollere som er definert for distribusjon.

- *Databasesystemer* - Disse komponentene representere de databasesystemer som benyttes i DIGLIB systemet.

- *Operativsystem* - Komponentene her består av de ulike operativsystem som brukes på tjener siden.

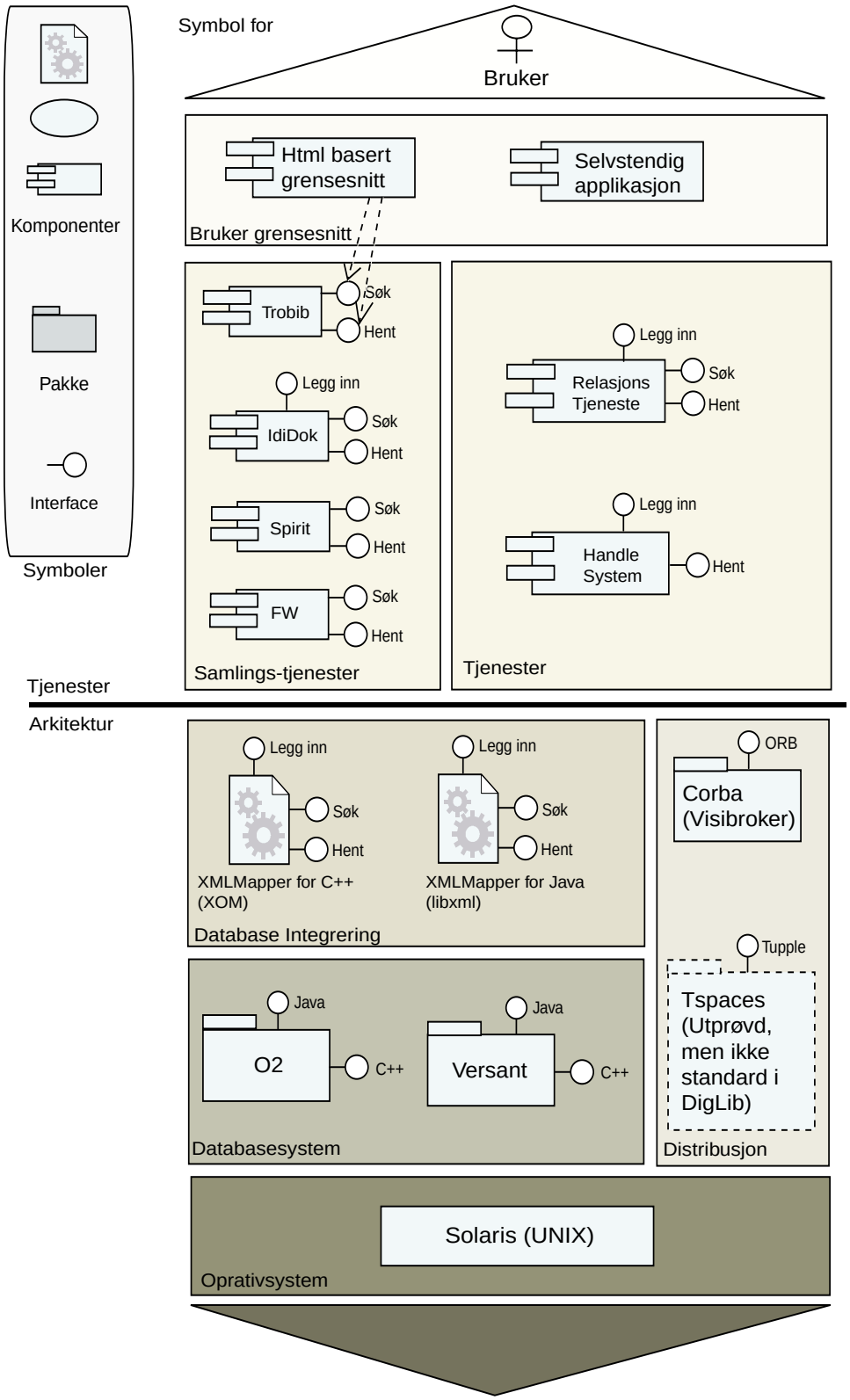
Det er viktig å presisere at dette er en "flytende" modell, dvs. den er under utvikling og vil forandre seg noe ettersom ulike komponenter blir lagt til eller forandres.

Modellen presenterer også de ulike grensesnittene som brukes mot de ulike komponentene. Når det gjelder utveksling av informasjonsobjekter har man bestemt seg for å satse på CORBA. Integreringen av denne standarden er i skrivende stund ikke ferdig utviklet, men modellen reflekterer hvordan dette er tenkt implementert. Det er også åpnet for å bruke andre typer arkitektur for distribusjon, og her har blant annet TSpace teknologi fra IBM **[Tspace]** vært sett på i forbindelse med denne hovedfagsavhandlingen.

Modellen er delvis modellert i UML, dvs. at komponentene er modellert med standardsymboler fra UML standarden.

Det er lagt inn et skille mellom tjenestelaget og databaseintegrering/distribusjon. Dette er tatt med for å vise hvordan de ulike komponentene plasserer seg. Over streken befinner de komponentene seg som er rettet mot brukeren. Komponentene under streken utgjør arkitekturen på tjenersiden.

Fig 3.2.2.1 Modell av DIGLIB systemet.



3.3 Informasjonsobjekter.

3.3.1 *Hva er et informasjonsobjekt?*

Informasjon er råvaren i det digitale biblioteket, og med informasjon menes det i denne sammenheng bilder, lyd, metadata og tekst. I DigLib prosjektet har man tatt i bruk begrepet *informasjonsobjekt* som betegnelse på denne digitale informasjonen. **[Aalberg]** . I informasjons-begrepet ligger også en forståelse av at dette er noe som kan prosesseres.

- Et informasjonsobjekt kan inneholde bilde, lyd, animasjon og tekst. Informasjonobjektet har ett innhold, det vil si at det er snakk om *ett* bilde, *en* tekst, *en* lyd, eller *en* spesialisert binærfil som inneholder alle disse elementene, f.eks.: PDF fil med tekstelig og billedlig innhold.

- Et informasjonsobjekt kan også bestå av flere informasjonsobjekter, eller være en del av et annet informasjons-objekt. Dette blir i denne avhandlingen foreslått realisert gjennom bruk av relasjoner.

Informasjonsobjekt samler de begreper som har vært brukt tidligere i oppgaven: komponent fra Dextermodellen (kapittel **2.3**) og digitalt objekt fra Arms's arkitektur (kapittel **2.4**). I DigLib prosjektet er et av målene å innføre en standardisert måte å behandle slike objekter på, dette inkluderer systemer for oppretting, organisering og behandling av slike objekter. Informasjonsobjektet skal foruten et innhold, også inneholde informasjon om hvordan objektet skal behandles.

3.3.2 *Informasjonsobjekter og persistens*

Lagring av informasjonsobjekter skjer i DigLib systemet gjennom bruk av objektorienterte database-system. Valg av objektorientert database underbygges av ønsket om å behandle informasjon som objekter. Objektorientert oppbygging av samlinger tillater å innføre bruk av funksjoner i informasjonsobjektene. Slik kan f.eks. et informasjonsobjekt med XML-innhold inneholde funksjoner for visning i HTML i tillegg til visning i XML. To typer av innhold for informasjonsobjekter er brukt: tekstelig og binært innhold.

Lagring av metadata og annen semi-strukturert data

I den modellen for lagring som er innført i DigLib (kapittel **4**) blir *tekstelig* innhold av typen metadata og annen semi-strukturert data lagret ved bruk av XML-basert lagringsmodell.

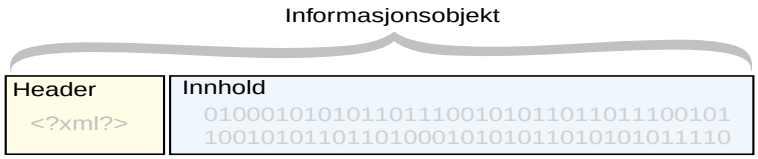
Lagring av binært innhold

Lagring av binære filer har vært et diskutert tema innenfor DigLib, og i lagringsmodellen (kapittel **4**) er det foreslått en løsning hvor man bruker egne klasser for lagring av denne type innhold.

3.3.3 *Distribusjon av Informasjonsobjekter*

Informasjonsobjekt eksisterer i en form i systemets database, og vil ved utveksling *mappes* til en annen form som forenkler og effektiviserer forsendelse. For å kunne utveksle slike informasjonsobjekter er det i DigLib-systemet foreslått en type objekt som består av en *header* og ett innhold (**Fig 3.3.3.1)**. Dette er i tråd med Arms's digitale objekt **[Arms1997]** . *Header*-delen innholder nøkkelmetadata som mimetype, adresse og andre instruksjoner for prosessering. Denne delen er strukturert i XML. Innholdsdelen av objektet inneholder selve informasjonen. Hele informasjonsobjektet er binært slik at det kan oversendes som en binær strøm.

Fig 3.3.3.1 Informasjonsobjekt for utveksling.



Et slik informasjonsobjekt skal kunne hentes frem gjennom bruk av en unik adresse som brukes mot en *object request broker* (ORB). ORB er en del av CORBA standarden.

I DigLib prosjektet har man satset på bruk av handles basert på *the handle system* fra CNRI **[CNRI]** . En handle vil da inneholde angivelse av den unike adressen som kan brukes mot ORB'en.

3.3.4 *Bruk av mimetyper*

Mimetyper angir formatet til innholdet av et informasjonsobjekt. Mimetypen består av superklasser som beskriver en gruppe formatet tilhører (f.eks.: *text, image, animation, etc*), og subbklasser som angir selve formatet (f.eks.: *jpeg, gif, bmp, etc)*. En mimetrype blir angitt slik: *Image/jpg*, dette tilsvarer et bilde av typen jpg.

3.4 Database-transaksjon i DIGLIB

3.4.1 *Fokus på metadata*

Grunnlaget for å velge en semi-strukturert modell som XML for å lagre informasjon, var at samlinger ofte består av en betydelig mengde metadata. Det er i tillegg brukt spesialverktøy for å indeksere binære dokument med tekstelig innhold (PDF, Word-doc, etc).

3.4.2 *Tidlige versjoner*

Sommeren 1999 ble det som et av de første ledd i en oppbygging av et digitalt biblioteks-system, satt i gang en eksprimentell implementering av et XML-grensesnitt mot den objekt-orienterte databasen *O2* fra *Ardent software*.

xml-o2-matic

Forfatteren av denne avhandlingen utviklet den første prototypen av et XML grensesnitt i språket *C++* mot O2. Dette biblioteket ble bassert på en XML-*parser* fra *IBM Alphaworks* **[XML4C]** . Utviklingen av denne XML-parseren er senere blitt overtatt av *Apache* og heter nå *Xerces* **[Xerces-C]** . Biblioteket som ble prototypet fikk navnet *xml-o2-matic* (XOM) for å synliggjøre den automatiske konverteringen mellom XML og den interne objektstrukturen i O2 **[Lichtenberg]** . Dette biblioteket ble i første omgang utviklet med spesielt hensyn til samlinger av metadata, og senere utvidet til å inkludere samlinger hvor man hadde både metadata og selve dokumentet metadataen refererte til.

Lagring av relasjoner

Det var i begynnelsen meningen å bruke denne modellen for lagring av relasjonene i database. Dette har vist seg vanskelig, da O2 ikke utvikles eller støttes lenger. Følgelig blir ikke Java grensesnittet mot O2 videreført til å støtte nye versjoner av SUN's Java. Valget falt da på en annen objektorientert database fra Verity.

3.4.3 *Ny modell for lagring*

Etter en del utvikling innenfor DigLib-systemet ble det klart at man trengte å skille metadata og binære filer. Binære filer ble tidligere lagret ved hjelp av spesielle utvidelser i XML-modellen hvor et XML-element tillot et binært innhold. Dette bryter med den generelle modellen for XML og en ny modell er utviklet. Denne er presentert i kapittel **4**.

Den nye modellen for lagring bruker Versants ODBMS og det tilhørende JVI (Java Versant Interface) API'et for Java. Dette systemet tillater lagring av Java-objekter med tilhørende funksjoner. Systemet er meget fleksibelt og har vist seg å være bedre egnet ved bruk av Java, enn *O2*. For en nærmere introduksjon til dette systemet anbefales web-adressen: *http://www.versant.com/*

3.5 Ulike samlingstyper i DIGLIB.

DigLib prosjektet har flere samlingstyper skaffet til veie av instituttet og studenter. Ideen med disse samlingene er å tilby studenter en rekke forskjellige formater (lyd, bilde, tekst) som kan brukes i egne prosjekt. Dette betyr at flere ulike tjenester kan oppstå til en tjeneste utifra de behov og ønsker den enkelte student har i sitt hovedfag- eller doktorgrad-prosjekt. Samlingene består foreløpig av følgende:

- *Trobib* - En samling av metadata om utgivelser i trondheim. Dette er en ren tekstelig samling, dvs. den innholder xml-strukturert metadata i ren tekst.
- *Ididok* - En dynamisk samling av publikasjoner ved IDI. Denne samlingen består hovedsakelig av binære filer (PDF), samt enkel metadata om Forfatter og tittel.
- *Bildebase* - En samling av flyfoto over gårder. Samlingen består av en metadata del og en binær del bestående av bilder i tre formater: "Full", "Medium" og "Icon".
- *Spirit of the Vikings* - En samling av lyd fra NRK's radiosendinger under andre verdenskrig. Samlingen består av metadata og lydfiler.

Grensesnitt

Med unntak av *Trobib* og *Ididok* er det ikke tatt hensyn til utvikling av brukergrensesnitt mot disse samlingene. Trobib og ididok har egne grensesnitt for søking og gjenhenting av informasjonsobjekter, som ble utviklet i forbindelse med denne avhandlingen.

Samlingene og lagringsmodellen

Alle disse samlingene ble testet mot lagringsmodellen for DigLib. Og med unntak av *Spirit of the Vikings* har systemet fungert tilfredstillende. *Spirit of the vikings* inneholder ekstra store lydfiler, og hele databasen er på over 6 gigabytes med data. Dette ga problemer med lagring relatert til databasesystemet, og overføring av store binær objekter i Java. Dette er tatt opp i kapittel **6.3.2**.

3.6 Systemer for distribuering vurdert i DigLib

3.6.1 *Hva er et distribuert system*

Slagordet til SUN: “The network is the computer”, uttrykker hva begrepet distribuert system handler om. Følgende sitat fra *Whatis.com* definerer begrepet:

> *“Computing is said to be “distributed” when the computer programming and data that computers work on are spread out over more than one computer, usually over a network”* **www.whatis.com**

Det er en vanlig oppfatning at et distribuert system er et system som består av flere datamaskiner med atskilt hukommelse og lagring, knyttet sammen på en slik måte at det for sluttbrukeren ser ut som et enhetlig system.

3.6.2 *Arkitekturer for distribusjon*

Det finnes en rekke arkitekturer som støtter distribusjon, og de som har blitt vurdert i DigLib-systemet er: Java’s RMI, CORBA og JavaSpace/TSpace.

Remote Method Invocation (RMI)

RMI er en teknologi i Java som støtter deling av objekter i et nettverk. Det fungerer nonelunde på samme måte som CORBA, men er noe enklere og er begrenset til bruk av Java teknologi, og objektdeling mellom java-program. **[RMI]**

CORBA

CORBA gjør det samme som RMI, men er en eldre og mer utprøvd standard. Denne er også likegyldig til hvilket språk klient og tjener er implementert i. CORBA er allikevel en mer komplisert løsning enn RMI. I tillegg til serveren og klienten må man ha en ORB (Object Request Broker) som tar seg av forespørsler og oversending av objekter. Dette er den løsningen som er valgt for distribusjon i DigLib systemet. Systemet er utprøvd, men er ikke implementert fullstendig i DigLib-systemet enda. **[CORBA]**

TSpace og JavaSpace

JavaSpace og TSpace bygger på den såkalte LINDA modellen for bruk i parallell prosessering. Disse to teknologiene innfører en nettverksbuffer som klientene kan kontakte og bruke til oversending og lagring av objekter. Ideene er et klientene kan bruke basis operasjoner som “Write” og “Take” for å lagre og hente ut objekter fra et felles rom. TSpace står for *tuplespace* og baserer seg på lagring av tupler hvor et tuppel består av et nøkkelelement og et eller flere data-/innholds-

element. Siden IBM står bak TSpace tar denne i bruk mye av den databaseteknologien som ligger bak *DB2*-systemet. TSpace er meget enkelt å arbeide med og tillater kraftige verktøy i forhold til søking i et tuplespace og lagringsmuligheter. Spesielt for TSpace er også støtten for XQL som er et SQL lignende spørre-språk for XML. TSpace muliggjør en rekke interessante bruksområder, blant annet støtte for håndholde enheter. **[Tspace] [Javaspaces]**

TSpace ble utprøvd i denne avhandlingen for å distribuere relasjoner, men det ble ikke funnet som en fullgod løsning. Problemet med denne teknologien er at den, som RMI er en java teknologi. Den er ikke like fleksibel og kraftig som CORBA. En kort gjennomgang av relasjonssystem basert på TSpace gjøres i kapittel **7.3.2**.

Kapittel 4

Lagringsmodell

I dette kapittelet blir det gjennomgått en modell for lagring av informasjonsobjekter i objektorientert database. Denne modellen danner grunnlaget for lagring av relasjoner som presenteres i kapittel **5**. Modellen bygger videre på den originale lagringsmodellen som ble utviklet i avhandlingen i forbindelse med DigLib-systemet. Følgende punkter blir gjennomgått:

- De ulike klassene i programbiblioteket
- Avhengigheter til andre programbibliotek
- Eksempel på hvordan lagringsmodellen fungerer

4.1 Klasser i lagringsmodellen

4.1.1 *Persistente klasser*

Lærdommene fra DigLib ble videreført i et XOM-lignende bibliotek (kapittel **3.4.2**) for Java som benyttes for å legge inn informasjon i Versant databasen **[Versant]** . Den generelle objektmodellen basert på XML består av følgende persistente klasser:

- *InformationObject* - et superklasse objekt som alle understående klasser arver. Dette objektet er formet i henhold til informasjonsobjekt-definisjonen i DigLib. Den består av en mimetype attributt og en innholdsattributt.

Videre har en tre klasser for lagring av XML strukturert data:

- *XMLDocument* - er et rotobjekt som samler elementene som inngår i det enkelte XML-dokument. Denne arver fra *InformationObject*.

- *XMLElement* - er en klasse som inneholder det enkelte

element og dets innhold. Denne arver fra *InformationObject*.

- *XMLAttribute* - er en klasse som inneholder eventuelle attributter som forekommer inne i et XML-element. Denne arver fra *InformationObject*.

Disse klassene følger malen for XML, dvs. de tar vare på informasjon som presenteres i XML-dokumentet uavhengig av intern struktur. Dette er allikevel en litt enkel modell som ikke tar hensyn til lagring av DTD'er og andre spesifiseringselementer i XML. Disse aspekter blir gjennomgått i diskusjonskapittelet (**6.3)**.

I tillegg til disse klassene har man en persistent klasse for lagring av binære filer. Denne klassen fungerer som en beholder for filer som ikke har rent tekstlig innhold, det vil si bilder, lyder og dokumenttyper som PDF, DOC, etc.:

- *BinaryDocument* - en klasse som lagrer binære filer. Dette er en eget objekt som ikke tilhører XML-strukturen. Denne arver fra *InformationObject*.

4.1.2 *Ikke-persistente transaksjons klasser.*

Disse klassene inneholder de funksjoner som tilsvarer *accessor* funksjonen i Dexter modellen (kapittel **2.3**), og RAP i Arms' arkitektur (kapittel **2.4.3**).

I lagringsmodellen for java er det implementert to klasser for innlegging og uthenting av informasjonsobjekter, og en spesialisert klasse for innlegging av hele samlinger:

- *libxml_in* - klasse med metoder for innlegging i database ved bruk av lagringsmodellen.

- *libxml_out* - klasse med metoder for uthenting fra database ved bruk av lagringsmodellen.

- *InsertXML* - klasse som tar seg av masse-innlegging av hele databaser spesifisert i XML

Systembeskrivelse av disse klassene finnes i Appendix **5.**

Ikke-persistent transaksjonsklasse i XOM

I lagringsmodellen for O2 databasen i DigLib ble det implementert et sentralt samlingsobjekt som sørger for komunikasjon mellom den ytre verden og den interne strukturen i databasen. Dette objektet er persistent i databasen.

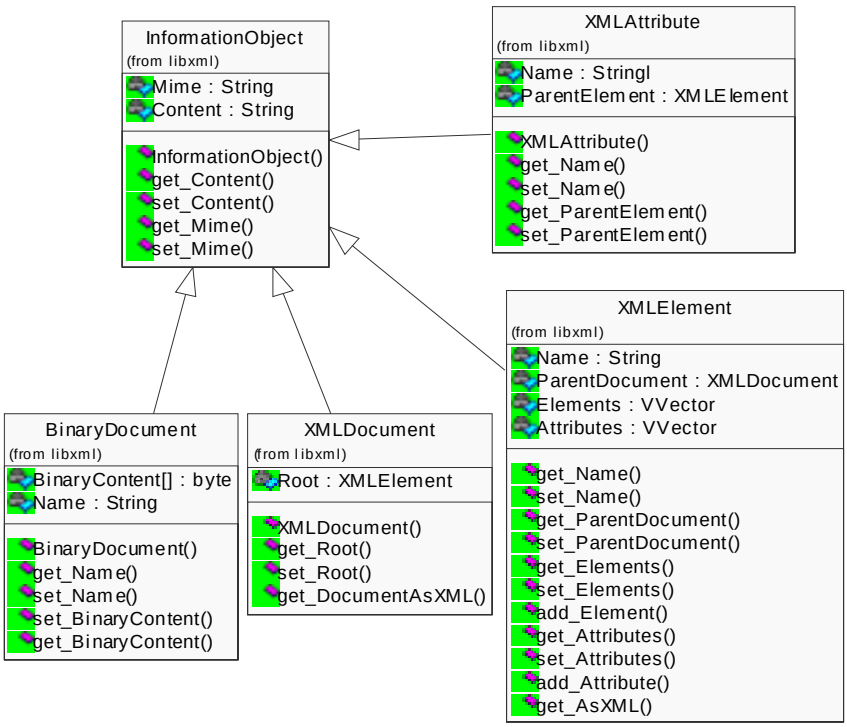
- *DLRepository* - Objekt som henter ut ønsket informasjon fra samlingen. Dette objektet fungerer som innfalssport og tilbyr de ulike tjenestene.

Et slikt *DLRepository* er ikke implementert i Java versjonen av biblioteket for å holde samlingene så enkle som mulig og uavhengig av endringer i aksesseringsklassen.

4.1.3 *Modeller av klasser i libxml.*

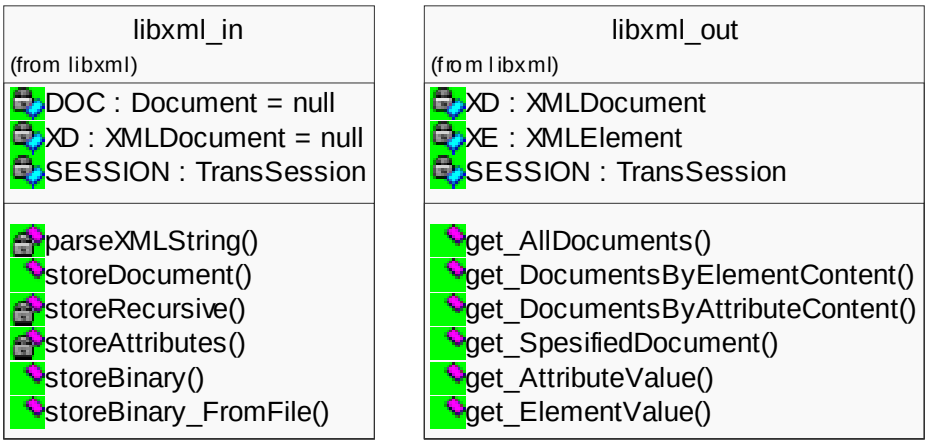
I modellen under (**Fig 4.1.3.1)** kan man se klassene som utgjør den persistente delen av lagringsmodellen. En beskrivelse av *libxml* er gjengitt i Appendix **5.1**. Legg merke til at alle disse klassene arver fra det universelle *InformationObject* og at denne klassen er en superklasse som ikke brukes til lagring.

Fig 4.1.3.1 UML modell av persistente klasser i lagringsmodellen.



I figur **4.1.3.2** ser man en modell over transaksjonsklassene i lagringsmodellen. Disse klassene inneholder funksjoner for å legge inn og hente ut lagrede verdier og XML. Det er kun mulig å aksessere attributtene i klassene gjennom disse funksjonene.

Fig 4.1.3.2 UML modell av ikke-persistente klasser for inn og uthenting.



4.2 Programbibliotek avhengighet

Javabiblioteket *libxml* er avhengig to java programbibliotek for å fungere. Disse er angitt under med det versjonsnummeret som ble brukt i prototypen:

- *XML-parser*: IBM Alphaworks xml4j versjon 3.0.1 **[XML4J]**
- Versants *ODBMS Java Bibliotek*: JVI 2.4.0 **[Versant1]**

XML-parser

XML-*parseren* leser inn XML-dokumentet og bryter det ned i et node-tre som kan traverseres. Programbiblioteket libxml benytter dette til å finne de ulike elementer og attributter som skal opprettes som objekter i databasen.

Java Versant Interface (JVI)

JVI brukes til å gjøre java-objekter persistente i databasesystemet, søke, og hente disse ut igjen.

4.3 Eksempel på bruk av lagringsmodell

4.3.1 *Lagring*

For å vise hvordan semi-strukturert data lagres i lagringsmodellen blir det nå gjen-

nomgått et enkelt eksempel.

Gitt at vi har en database hvor vi ønsker å lagre et enkelt telefonregister med navn og telefonnummer som metadataelementer. Vi ønsker også å kunne uttrykke om telefonnummeret er hjemmenummer eller mobilnummer. I eksempel **12** kan man se en enkelt post i dette telefonregisteret beskrevet i XML.

Eksempel 12: Enkel XML-post for lagring

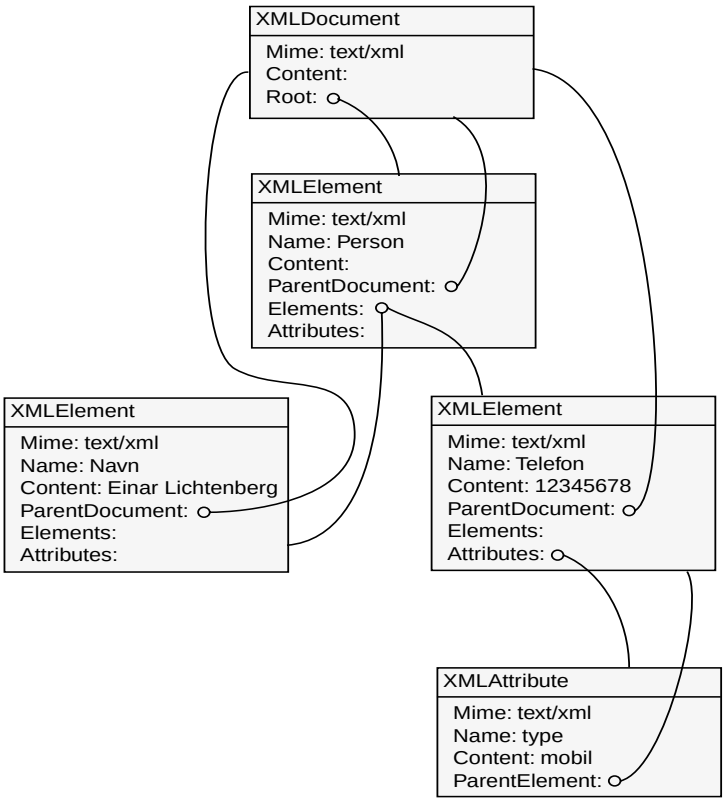
```
<Person>
   <Navn>Einar Lichtenberg</Navn>
   <Telefon type="mobil">12345678</Telefon>
</Person>
```

Når dette skal lagres i databasen, *parses* denne posten som et XML-dokument og for hvert element og attributt opprettes et objekt. Utifra eksempelet vil vi få en objekt struktur som i figur **4.3.1.1**. Først opprettes et objekt av type *XMLDocument* som samler alle objektene tilhørende denne posten under seg. XML-standarden krever at man må ha et rot element som samler alle elementer i dokumentet. Dette rot elementet er i eksempelet *Person*, og et nytt objekt av type *XMLElement* opprettes med *Name* lik “Person”. Derretter settes *XMLDocument*-objektets attributt *Root* til å peke på *XMLElement* “Person”.

Vi har opprettet selve grunnstammen i posten, og kan fylle inn metadataelementet som utgjør posten. For “Navn” og “Telefon” -elementene opprettes det to nye *XMLElement*-objekter. I begge disse settes attributtene *Name* til navnet på elementet, og *Content* til innholdet av elementet. Disse to *XMLElement*-objektene legges inn i en liste over tilhørende elementer hos rot-elementet. Vi får således en trestruktur som er lik den angitt i XML-dokumentet.

Når det gjelder elementet “Telefon” har dette en attributt som forteller om type telefonnummer. Attributtet opprettes som eget objekt av typen *XMLAttribute* og attributtene *Name* og *Content* i objektet gis verdiene til attributtet. *XMLElement*-objektet som *XMLAttribute* tilhører legger inn en peker til dette, og *XMLAttribute* objektet legger inn en peker til det *XMLElement* det tilhører.

Fig 4.3.1.1 Modell av objekter generert fra eksempel 12



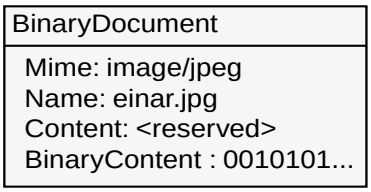
Lagring av binærfiler - masseinnlegging

Når det gjelder lagring av binærfiler skjer dette i et objekt kalt *BinaryDocument* og har en noe spesiell lagringsform. Ved masseinnleggelse av databaser gjennom bruk av programmet *InsertXML* bruker man en bestemt DTD som reserverer elementnavnet *Binary.* Dette elementet er definert til å inneholde noen attributter som angir hvor binærfilen finnes og hvordan den skal lagres, samt en angvielse av mimetypen. Denne måten å angi hva som skal lagres kan utvides til å ta i bruk relasjonssystemet, og i kapittel **5** er dette gjennomgått.

Lagring av binærfiler - innlegging av enkeltobjekt

I figur **4.3.1.2** ser man et eksempel på et slikt *BinaryDocument*-objekt. Dette er et et bilde av den personen som angis i telefonregister-posten. Dette kan legges inn ved å benytte en egen funksjon for innlegging i *libxml_in.* Binærfilen kan enten hentes fra en adresse eller gjennom en variabel av type *byte array.* Bruk av store binærfiler blir diskutert i kapittel **6.3.2**. Det vil senere i avhandlingen (kapittel **5**) vises hvordan man kan knytte dette opp mot metadataen i eksempelet over.

Fig 4.3.1.2 Eksempel på BinaryDocument objekt



4.3.2 *Gjenhenting*

Gjenhenting av lagret data skjer gjennom bruk av *libxml_out.* Denne inneholder funksjoner som kan hente ut:

- *Fullstendig XML-dokument*. Denne funksjonen traverserer objekt-treet og genererer et XML-dokument utifra navn og verdier oppgitt i *XMLElement*-objekter og *XMLAttribute*-objekter.
- Verdi for spesifisert element. Denne funksjonen henter ut verdien for et spesifisert element navn som forekommer under et spesifisert XMLDocument-objekt.
- *Verdi for spesifisert attributt*. Denne funksjonen henter ut verdien for et spesifisert attributt-navn som forekommer i et spesifisert *XMLElement* under et spesifisert *XMLDocument.*
- *Innhold av binær-objekt*. Denne funksjonen henter ut verdien for et spesifisert *BinaryDocument*-objekt.

4.4 Systembeskrivelse og kildekode

Systembeskrivelse for lagringssystemet er presentert i Appendix **5**. Kildekode for programbiblioteket finnes i Appendix **8**.

Kapittel 5

Relasjoner mellom informasjons-objekter.

I denne delen av hovedoppgaven vil det bli presentert en modell og forslag til et system for distribuerte relasjoner mellom informasjonsobjekter. Det vil bli gjennomgått en løsning som først og fremst har anvendelse i DigLib-prosjektet og benytter seg av lagringsmodellen presentert i kapittel **4**. Modellen blir presentert gjennom følgende punkter:

- Komponenter i systemet
- Måter å aksessere en relasjons-samling på
- Analyse av bruk: Hvordan et relasjons-system kan brukes i DigLib-systemet
- Begrepet relasjoner i relasjonsmodellen.
- Presentasjon av arkitektur for relasjon
- Lagring av relasjoner
- Akksesering av relasjoner
- Eksempel på bruk av relasjonsystemet.

Den modellen som presenteres her vil være med å bygge et system for såkalt *distribuert lenking*, dvs. et system hvor lenkene utgjør selvstendige relasjoner mellom informasjons-objekter. Det vil også bli gjennomgått et metadata-sett for relasjoner som er integrert i en lenkearkitektur. Denne delen utgjør kjernen i relasjons-systemet, og vil bli bygget inn i en system arkitektur som gir mulighet for behandling av relasjoner og relaterte komponenter. Oppbygging av relasjons-systemet.

5.1 Oppsumering av relasjonsmodellen

Det blir her gjennomgått de komponenter som utgjør systemet. Eksempel på bruk av relasjonsystemet blir tatt opp i kapittel **5.7**. To løsninger på en slik arkitektur vil preseneteres: En enkel klient/tjener løsning, og en distribuert løsning som ikke er realisert gjennom ferdig prototype.

5.1.1 *Komponenter i systemet*

Et relasjons system trenger en del grunnleggende komponenter som utgjør arkitekturen. Disse er identifisert i tabell **6**.

Tabell 6: Komponenter i et relasjons-system.

Komponenter	Beskrivelse
Lager for relasjoner	Et underliggende database-system som kan lagre relasjons objekter på en hensiktsmessig måte
Transaksjons-bibliotek	Et programbibliotek som tar seg av tilgang til databasen, dvs. foretar transaksjon av persistent data på en standardisert måte.
Server	Et programbibliotek som tilbyr de tjenester relasjons-systemet trenger for å behandle relasjoner.
Protokoll for kommunikasjon	En klient/tjener løsning som kommuniserer med klienten.
Klient	En klient som sender forespørsler, mottar tilbakemeldinger fra server og presenterer disse for brukeren.

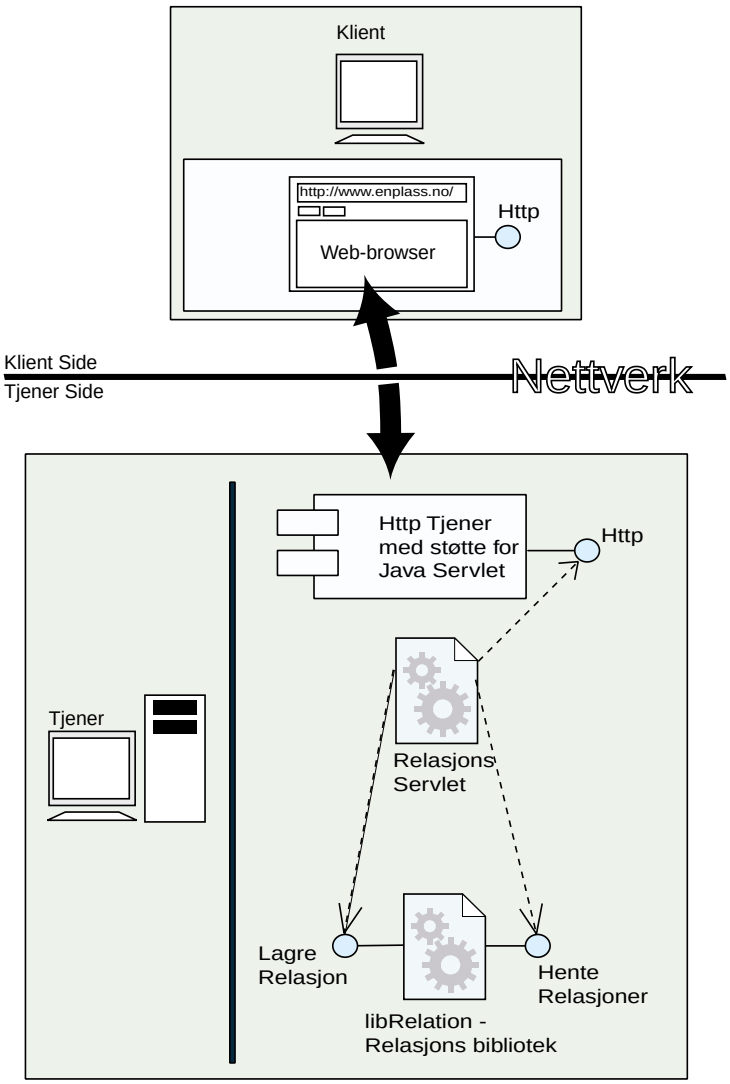
5.1.2 *To typer system for relasjoner*

I denne avhandlingen er det sett på to måter å tilfredstille komponentene nevnt i tabell **6**. En måte å bygge et relasjons-system er ved å benytte en vanlig HTTP-basert løsning hvor klienten er en web-side som kommuniserer direkte med en server. En annen måte å gjøre dette på er å bygge et distribuert system for relasjoner som samsvarer med den planlagte bruk av distribuerte tjenester ved DigLib.

Http-basert relasjons-system

I figur **5.1.2.1** kan man se hvordan en http-basert løsning av relasjons-systemet vil se ut. All behandling av relasjoner vil i en slik løsning bli gjort på tjener-siden, og resultatet av behandlingen vil bli presentert brukeren gjennom en webside. Det er her tatt i bruk Java Servlets teknologi på tjener siden.

Fig 5.1.2.1 Klient/Tjener basert relasjons-system basert på Java Servlets.



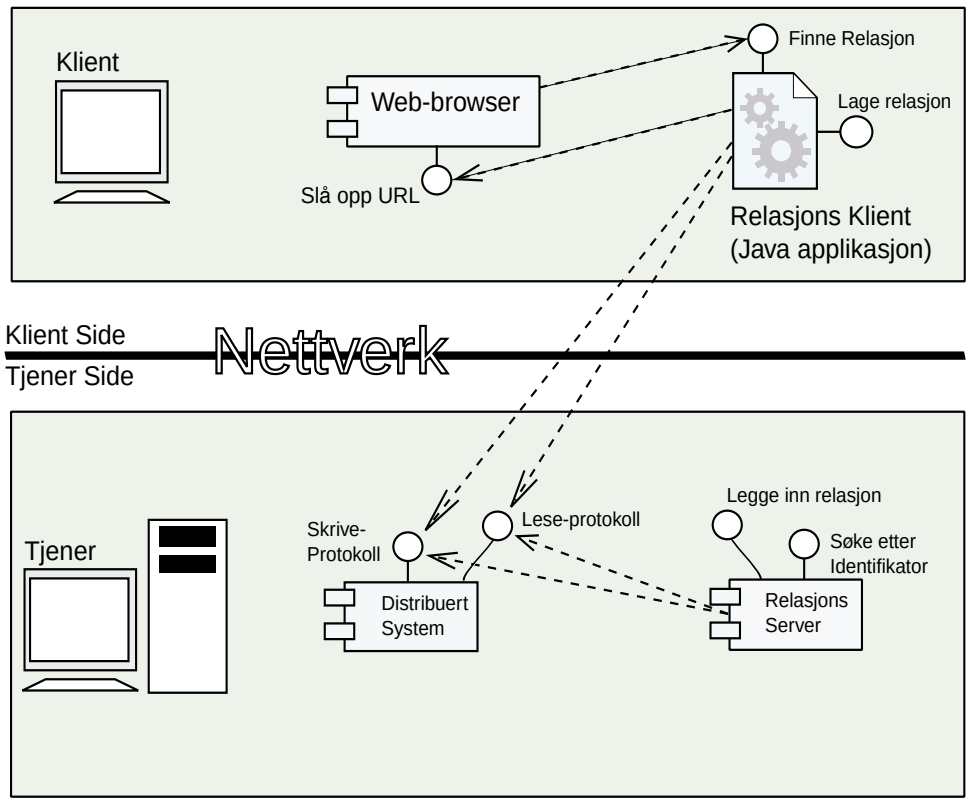
På tjener siden består relasjons-systemet av flere komponenter:

- En *http-tjener* som sørger for komunikasjon med klienter over http protokollen.
- En *java-servlet* som kommuniserer med klienten gjennom http-tjeneren. Denne genererer html sider for å presentere informasjon til bruker.
- Et underliggende *relasjons-bibliotek* som tar seg av behandling av relasjoner. Denne har tilknytning til et underliggende lagringssystem.

Distribuert relasjons-system

Under (**Fig 5.1.2.2)** er det presentert en modell som beskriver de hovedkomponenter et distribuert relasjons-systemet består av.

Fig 5.1.2.2 Hovedkomponenter i et distribuert relasjons system.



Man ser i denne figuren de tre ulike komponentene som det distribuerte relasjons systemet består av:

- *Klienten* består av et lite program som brukeren kan bruke sammen med en web-browser eller som et eget grensesnitt. Dette programmet gir mulighet til å "slå opp" ulike identifikatorer og presentere alle relasjonene denne identifikatoren har til andre ressurser på nettverket. Brukeren kan velge en relatert ressurs i klientvinduet og denne ressursen vil da presenteres i web-browseren.

På server siden har man to ulike komponenter:

- *Distribuert arkitektur.* I DigLib-systemet vil et distribuert system være basert på CORBA. Da denne hovedfagsavhandlingen ble skrevet var denne

teknologien ikke implementert i DigLib og det blir i kapittel **7.3** gjennomgått et distribuert system basert på TSpace teknologi fra IBM **[Tspace]** .

- *Relasjons-tjener* som benytter den distribuerte arkitekturen til å kommunisere med klienten. Relasjons-tjeneren benytter et underliggende lagringssystem for transaksjon med relasjons-samling.

5.2 Analyse av bruksområde

5.2.1 *Situasjoner i dagens DIGLIB*

Bruker orientert - situasjon A

Gitt følgende situasjon: En bruker av DigLib logger seg på for å gjøre søk etter informasjon om et gitt tema.

Ideelt sett vil brukeren finne et enhetlig grensesnitt for søking i alle de databaser som finnes ved DigLib. Eventuelt kan han avgrense hvilke samlinger han ønsker å se i. I noen tilfeller vil de ulike samlingene kun være søkbare gjennom egne grensenitt. Allerede her må brukeren foreta valg som han/hun kanskje ikke har fullstendig oversikt over og som kan begrense de dokument som blir oppdaget. Brukeren skriver inn sitt søke-query på søkesiden og får tilbake en mengde med dokumenter av ulik relevans. Disse presenteres med tittel, forfatter og kanskje en kort beskrivelse samt en lenke til dokumentet. For å avgjøre om dokumentene er av relevans for brukeren kan han/hun enten prøve å danne seg et bilde av dokumentet utifra den presenterte metadata, eller han/hun kan klikke på lenken for å lese/se dokumentet. Dette er en prosess som er tidkrevende og som ofte kan gi dårlige resultater. Bedre systemer for forbedring av søkepresisjon og søkerelevans kan konstrueres, men disse kan fort bli kompliserte og tunge.

Bruker orientert - situasjon B

Gitt følgende situasjon: En bruker har allerede en referanse til et dokument i DigLib som han/hun bruker, og som er av høy relevans.

Dette dokumentet inneholder noen refareanser i form av en referanseliste, samt noen lenker i selve teksten. En naturlig måte å finne andre dokument som omhandler samme tema vil da være å gå gjennom referanser og lenker som finnes i dokumentet. Disse referansene er begrenset til det tidspunkt dokumentet ble laget. Har man funnet et dokument fra 1991 vil referansen som dette inneholder også være fra 1991, med mindre forfatteren har vært inne og endret på dokumentet. Hvis forfatter har endret dokumentet for å angi nye referanser, vil det allikevel være mulig at det finnes versjoner av dokumentet på nettverket som ikke har fått

disse endringene. En slik situasjon kan skyldes at forfatter ikke har adgangsrettigheter til de systemer hvor dokumentversjonene befinner seg. Brukerens dilemma i dette kan være å finne de referansene som er interessante når det gjelder ny forskning og utvikling i forhold til dokumentets tema.

Bruker orientert - situasjon C

Gitt følgende situasjon: Flere brukere sitter med ulike lenker til informasjonsobjekter som er av relevans for et aktuelt informasjonsobjekt.

Hvis man kan hente inn de relasjoner som brukere gjør seg når de går fra en ressurs til en annen, kan man bygge et nett av relasjoner som kan være verdifullt for andre brukere. Dette er blant annet utprøvd i AntWorld **[AntWorld]**

Arkitektur orientert - situasjon D

Gitt følgende situasjon: En samling består av flere ulike typer informasjonsobjekter (bilder, lyder, metadata).

I en slik situasjon som denne finnes det flere måter å knytte sammen informasjonsobjektene. Det som blir foreslått i denne oppgaven er å bruke et relasjonssystem til å knytte disse objektene sammen. Man kan da bygge inn muligheten for å legge inn relasjoner i samlinger, eller legge disse i en egen relasjonsbase utenfor samlingen.

Arkitektur orientert - situasjon E

Gitt følgende situasjon: Man har flere samlinger som inneholder informasjonsobjekt innenfor samme tema.

I DigLib har man særlig to samlinger som begge har Trondheim som beskrivelses område: *Trobib* inneholder metadata om kunstverk og litteratur fra Trondheim. *Bildebasen* inneholder både metadata og bilder fra Trondheim. Ved å benytte seg av et relasjons-system kan man knytte relasjoner mellom disse to ulike samlingene.

5.2.2 *Bruk av system for relasjoner*

Det finnes i alle situasjonene nevnt i kapittel 5.2.1 mulighet for å legge inn et system som angir relasjoner mellom ulike dokumenter. I et slikt hjelpemiddel vil en kunne automatisk spørre en tjener om det finnes relasjoner til det dokument man har funnet fram til. Relasjons-systemet kan brukes ved siden av tradisjonelle søkesystem som en ekstra dimensjon i søkeprosessen. Man kan da tenke seg muligheten for å se om det i mengden av de gjenfunnede dokumenter finnes noen relasjoner, og om de gjenfunnede dokumenter har relasjoner til dokument som

ikke finnes i mengden. Søkessystemet vil da brukes som en inngangsportal for å finne en mengde av dokument som er relevante, og så kunne supplere resultatet med ulike relasjoner disse dokumentene har til andre dokumenter. Dette kan selvfølgelig gi en mye større mengde dokumenter og kan virke forvirrende. En slik supplerende funksjon setter krav til hvordan informasjonen filtreres. Hvis man bare går utifra de gjenfunnede dokumenter som har relasjoner innenfor dokumentmengden vil man kunne få en kraftig innsnevring av et søk. Hvis man også tar hensyn til relasjoner utenfor dokumentmengden vil det totale antall dokumenter kunne øke kraftig.

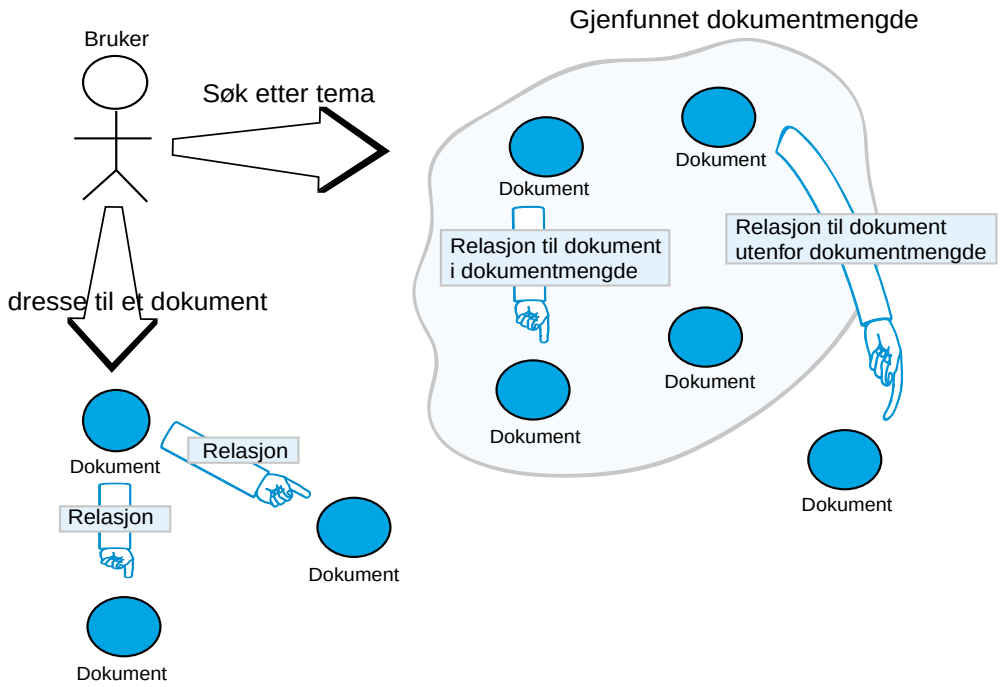
Traversering av relasjonstre

En annen måte å finne frem til relasjonene vil være å slå opp adressen til et dokument i relasjonssystemet og få tilbake de relasjoner som dokumentet har til andre dokumenter. På denne måten kan man traversere et relasjons-tre, fremfor å finne dokumenter gjennom tradisjonell søking. Forutsetningen er her at man allerede vet noe om dokumenter innenfor det gitte tema, og har en adresse til dette.

Gjenfinning av relasjoner

I figur **5.2.2.1** kan man se hvordan relasjons-systemet er tenkt som hjelpemiddel i oppdaging av nye dokument. Modellen viser både direkte oppslag og supplering til søkeresultat.

Fig 5.2.2.1 Ulike måter å finne relasjoner på.



Analyse av relasjonsnett

Andre muligheter som gjør seg gjeldende går mer i retning av eksprimentelle metoder for å trekke ut informasjon av et relasjons-nett. F.eks. kan man lett se hvilke dokumenter som henviser til det dokumentet man har funnet. En kan utifra en slik kobling gjøre antagleser om hvorvidt et dokument er en *autoritet* i forhold til et annet. Hvis et dokument ofte blir relatert til av andre dokumenter, kan man anta at dette dokumentet har en viss signifikans i forhold til disse dokumentene. Hvis et dokument har hovedsakelig relasjoner til autoriteter kan man anta at disse er utvidelser av autoriteten **[Landgraf1999]** . Dette området ligger utenfor denne avhandlingen og er vurdert i kapittel 7.4.

5.3 Gjennomgang av relasjonsbegrepet

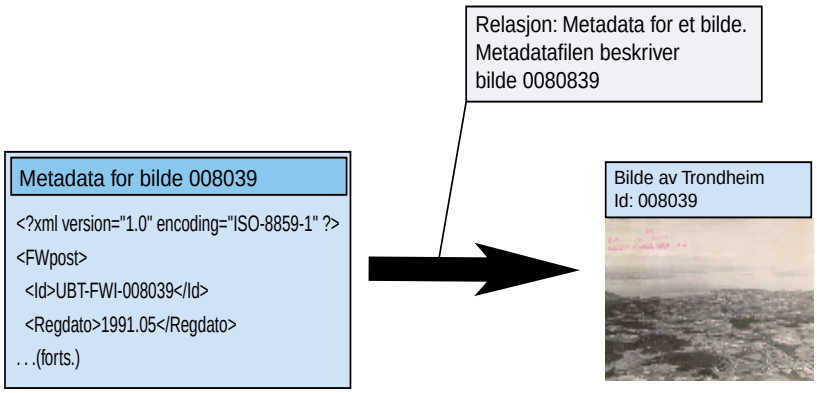
5.3.1 *Hva er en relasjon i relasjons-systemet?*

For å gi et klarere bilde av hvordan relasjoner fungerer kan man se på figur **5.3.1.1**. Figuren viser to informasjonsobjekter (IO):

- IO1: Inneholder metadata om et bilde (IO2).
- IO2:Inneholder selve bildet.

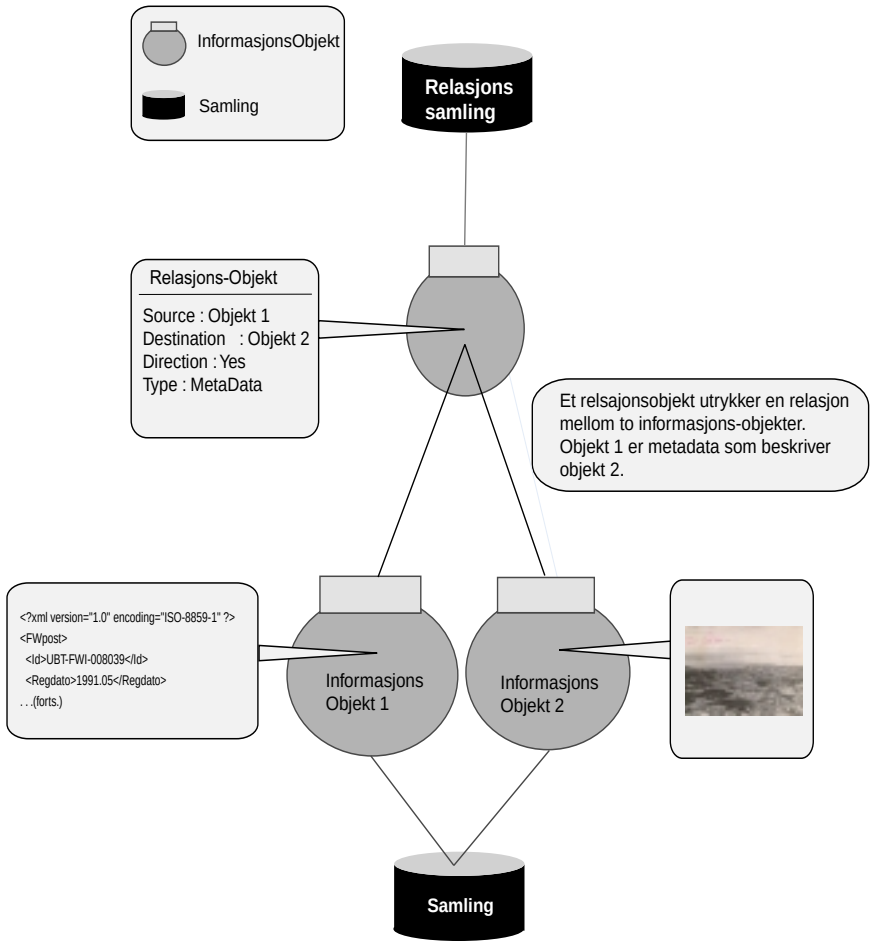
Disse to IO'ene har et forhold mellom seg som er følgende: IO1 beskriver innholds-informasjon om IO2. I et typisk biblioteks-system ville det i IO1 ha ligget en lenke i metadataen som refererte til en adresse hvor IO2 lå. I et slikt forhold vet ikke IO2 om at det har et forhold til andre objekt. Hvis man feks. søkte og fant IO2 ville det ikke fremgå at dette objektet hadde metadata knyttet til seg. For å få til en slik kjennskap trenger man et "mellom"-objekt eller mer presist et relasjons objekt. Et slikt relasjonsobjekt er et uavhengig, og frittstående objekt som angir adresse til de to objektert som har en relasjon, og samtidig angir retning og ett sett med data om relasjonen. En slik mulighet for kjennskap gjør seg særlig gjeldende der hvor man ikke har kontroll over objektene man lenker sammen. Dette kan være i situasjoner hvor man ikke har rettigheter eller tilgang på objektene man ønsker koblet sammen.

Fig 5.3.1.1 Enkelt relasjonsforhold mellom metadata og et bilde.



I dette systemet vil relasjonen selv være et informasjons-objekt som i denne sammenheng vil inneholde en samling data som forteller noe om hvordan to informasjons-objekter hører sammen. Relasjonen vil inneholde referanser (handles) til objekter gjennom å identifisere objektets URI, og samtidig ha en egen referanse. Relasjonsforholdet som er angitt i figur **5.3.1.2** modellerer det eksempelet som ble gitt over (**Fig 5.3.1.1)**. Her er det tatt i bruk angivelse av relasjonstype, som tillater angivelse av forholdet mellom informasjonsobjektene.

Fig 5.3.1.2 Relasjonsforholdet i figur 5.3.1.1 beskrevet som objekter.

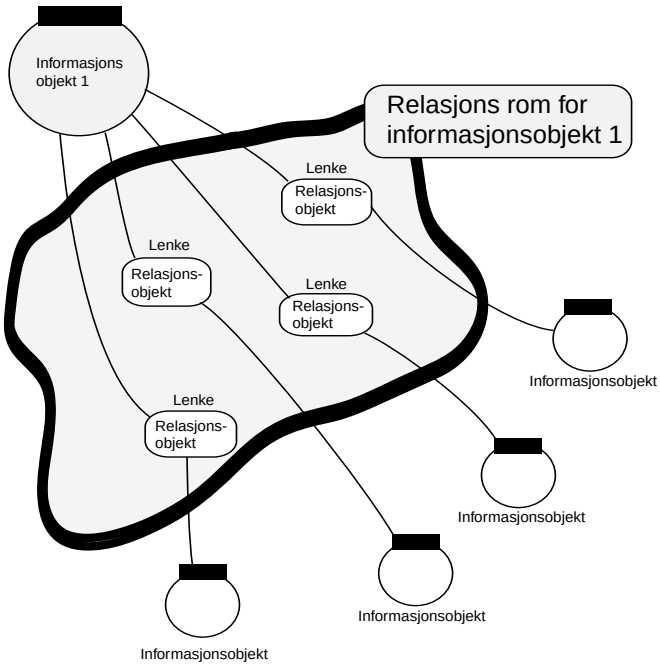


5.3.2 *Konseptet relasjonsrom*

Alle relasjoner som tilhører et informasjonsobjekt utgjør *relasjonsrommet* til dette objektet. Dette vil si at hvis et informasjonsobjekt er lenket til tre relasjonsobjekter (enten som *Source* eller *Destination*) - vil disse tre relasjonene utgjøre det digitale dokumentets relasjonsrom.

Relasjonsrommet til et informasjonsobjekt er vist i figur **5.3.2.1**. Vi kan her se at informasjonsobjekt 1 har en rekke relasjoner til andre informasjonsobjekt. Alle disse relasjonene som tilhører informasjonsobjekt 1 utgjør informasjonsobjektets relasjonsrom.

Fig 5.3.2.1 Relasjonsrom.



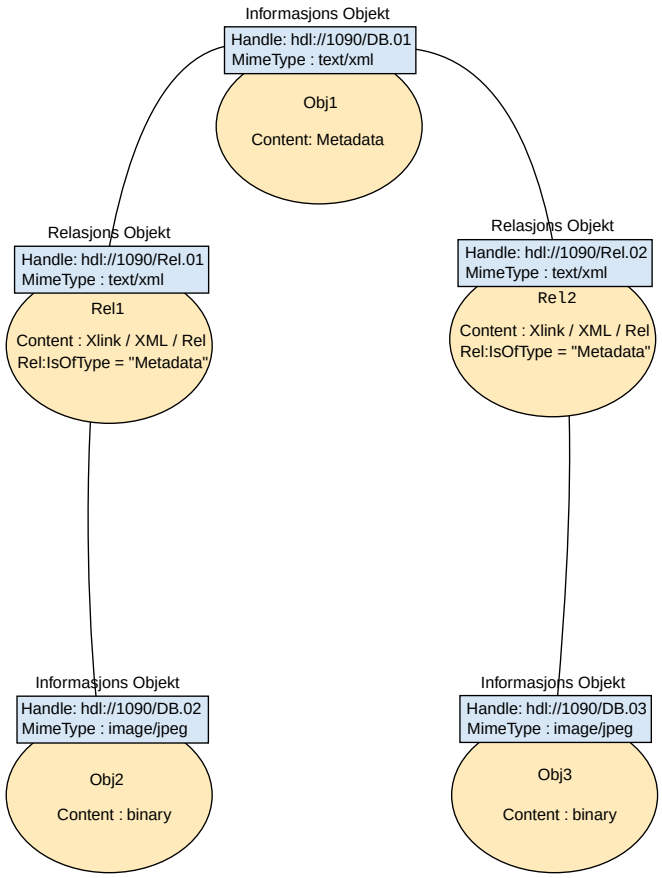
5.4 Gjennomgang av relasjons-konseptet

Det vil først bli gjennomgått en forklaring av relasjoner mellom "interne" objekter, dvs. mellom informasjonsobjekter som ligger i samme lokale samling - og som man har full kontroll over. Dette er i samsvar med alle de situasjoner som er presentert i kapittel 5.2.1

Relasjoner mellom informasjonsobjekter i en samling

Her følger en figur (**Fig 5.4.0.1)** som viser hvordan relasjoner mellom informasjonsobjekt kan se ut. Informasjonsobjektene som brukes i denne modellen bygger på de prinsipper vi har brukt i DigLib systemet. En kan se at det i denne figuren dreier seg om tre informasjonsobjekt og to relasjoner. I figuren er objekt 1 et informasjonsobjekt bestående metadata, objekt 2 og objekt 3 er bilder. Et hvert informasjonsobjekt er assosiert med en mimetype som forteller hvilket format innholdet av objektet har, samt en handle som angir adressen til objektet. Det er i dette eksemplet brukt *handles,* som er en del av den standard for identifikatorer som DigLib skal bruke.

Fig 5.4.0.1 Relasjonsforhold mellom informasjonsobjekter i en samling.



Objekt 1 har en relasjon med objekt 2 og en med objekt 3, og utifra relasjonene som her er angitt kan man lese følgende informasjon:

Objekt 1 inneholder metadata om objekt 2 og objekt 3. Dette angis av relasjonsobjekt 1 som antyder at relasjonstypen er metadata, og at kilde er objekt 1 (objekt 1 er metadata om objekt 2 og objekt 3).

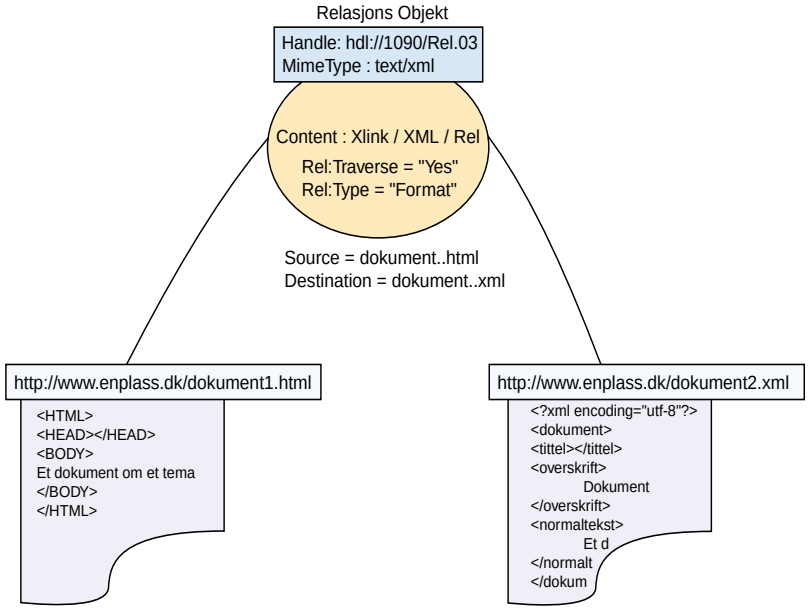
Angivelse av relasjonstype

Man kan også bruke relasjonstypen “format” og en traverserings attributt til å vise om et dokument er en original eller en sidestilt format-versjon.

I figur **5.4.0.2** illustreres en relasjon mellom to ulike versjoner av et dokument på internett. Dokument 1 er her “originalen” og dokument 2 er en XML utgave av samme dokument. I denne senarien vil dokument 1 være kilde og dokument 2 destinasjon. Hvis man så tenker seg at begge dokumentene var format-versjoner av et annet dokument ville man ikke hatt noen entydig kilde og destinasjon. Relasjonen ville være retningsløs, og bare angi at dette er to format-versjoner av

samme dokument. Man trenger derfor et parameter som kan angi om man kan følge eller traversere en lenke, eller om denne lenken angir et slik “dette-er-det-samme,-men-i-et-annet-format” forhold.

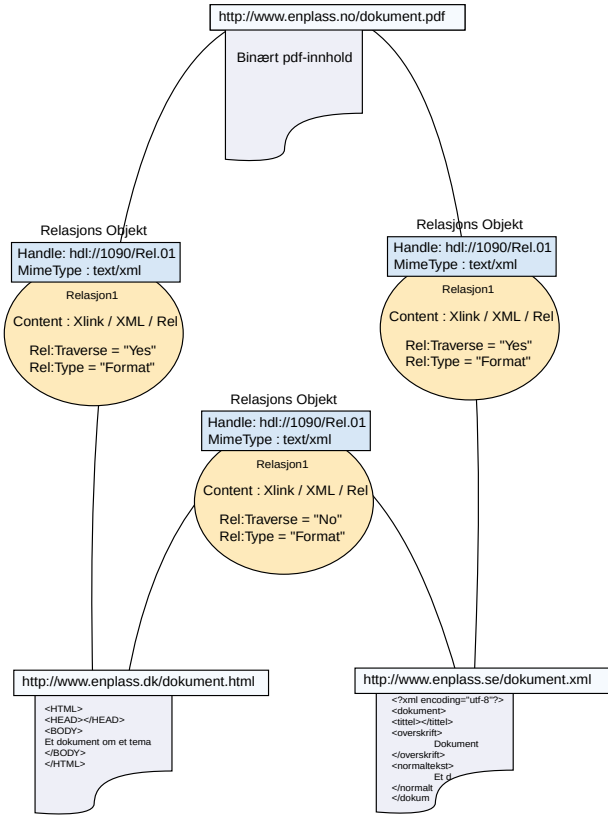
Fig 5.4.0.2 Format-relasjon mellom et Html og et XML dokument.



Originaler og versjoner

I modell **5.4.0.3** kan man se at original-dokumentet i pdf-form har to relasjoner til to andre dokument. Disse dokumentene er versjoner av original dokumentet. Dette kan en lese ved å se på relasjonstypen som er “Format” og traverserings-attributten som er satt til “Yes”. Original-dokumentet knytter en relasjon til hvert av de ulike dokumentversjonene, og angir at forholdet skal traverseres. Original-dokumentet er *source* og versjonsdokumentet er *destination*. Relasjonen mellom de ulike versjonene angir et ikke-traverserbart forhold.

Fig 5.4.0.3 Format-relasjon mellom et originalt. og to versjons-dokument.



Hvis man har funnet frem til html-dokumentet kan man ved å undersøke *relasjonsrommet* til dette dokumentet se at det er en versjon av en original (pdf-dokumentet). Man kan også se at dokumentet finnes i andre versjoner (xml dokumentet). Finnes det flere versjoner av dette dokumentet vil disse være knyttet til html-dokumentet. Man kan da raskt fastslå hva som er original-dokument og hvilke/hvor mange versjoner dette har.

En slik modell vil gi et stort antall relasjonsobjekter, men kompleksiteten i hvert objekt vil være på et minimum, og det vil være enklere å hente ut relasjoner fra systemet.

5.5 Data-sett for å uttrykke en relasjon

5.5.1 *Arkitektur for lenker*

Det er her viktig å skille mellom de to forskjellige data-sett det her er snakk om. Det ene er arkitekturen for å uttrykke lenken, det andre er metadataen for å

uttrykke relasjonen.

XML er tatt i bruk særlig fordi dette tillater utvidelse og endringer av metadata-sett og relasjons-struktur uten å måtte gjøre store endringer i kode, men også fordi dette er en god måte å synliggjøre data på. I dette hensende kan man også blande ulike klassifikasjon-systemer (f.eks.: Dublin Core og eget format).

Out-of-line

Beskrivelse av lenker er basert på XLink standarden. Denne er en kommende standard som lover en hel del muligheter i forhold til *out-of-line*-lenke. **[W3C Xlink]** XLink er en del av XML spesifikasjonen (kapittel 2.7). En slik lenke ligger i utenfor informasjonsobjektene det beskriver en lenke mellom.

Krav til oppbygning

Det har tidligere i oppgaven vært nevnt hvordan XLink er bygget opp - dette vil nå bli tatt i bruk for å beskrive lenker i relasjons-systemet. Før dette vil det bli gjennomgått en del forutsetninger:

- XLinksettet skal være enkelt og ikke ha for kompleks oppbygning.
- XLinksettet må være bygd på en slik måte at det ikke forutsetter en DTD for å fungere. Dette betyr at man må legge inn alle attributter som er nødvendig i element-tagger.
- XLinksettet må åpne for et relasjons-sett. Her vil det bli brukt namespace for å holde seg til XML standarden. Man må ha en generell modell eller rammeverk som kan representere ulike typer metadata.

Følgende punkter må være med i beskrivelsen av lenken:

- Adresse til kilde dokument. (Lenke fra)
- Adresse til destinasjons dokument. (Lenke til)
- Hvilken vei lenken går, eller om den er uten retning.
- Type lenke. Dette inngår i relasjonsbegrepet og angir relasjonsforholdet. Relasjonstyper er beskrevet i kapittel 5.5.4

Toveis lenker

Lenkene som her blir presentert er “bidirectional links”. Disse lenkene muliggjør kjennskap både for tillenkede informasjonsobjekt og det som lenker. Slike lenker løser blant annet problemer med rettigheter når man ønsker å lenke to dokumenter

i to forskjellige databaser hos forskjellige eiere, man trenger ikke adgang til eksterne databaser for å forandre lenker i dokumenter/metadata. **[Hitchcock1998]**

Span-to-span lenker

Gjennom bruk av XPointer (kapittel 2.7.2) kan man også generere span-to-span lenker som beskrevet i kapittel 2.1.3.

5.5.2 *Ad-hoc beskrivelse av realsjon*

I *raw* XML med bruk av "namespace" kan man definere en enkel relasjon som vist i tabell **13**. Denne relasjonsbeskrivelsen tar ikke med relasjons spesifikke metadata, men konsentrerer seg om å beskrive en lenke og type relasjon. En slik relasjon er basert på eget format og ikke standardisert. Dette kan ha konsekvenser for senere migrering til andre applikasjoner. Dette relasjons-settet ble forkastet til fordel for en XLink spesifisert relasjons-arkitektur.

Eksempel 13: Ad-hoc relasjons-beskrivelse.

```
<?xml version="1.0" encoding="UTF-8"?>
<Relation xmlns:Rel="http://fenris.idi.ntnu.no/namespaces/relation/">
    <Rel:Source>URI</Rel:Source>
    <Rel:Destination>URI</Rel:Destination>
    <Rel:IsOfType>Metadata, Reference, Internal, External </Rel:IsOfType>
    <Rel:IsDirectional>Yes, No</Rel:IsDirectional>
</Relation>
```

5.5.3 *Xlink basert beskrivelse av relasjon*

I tabell **7** er det presentert en XLink basert relasjon av eksempel **13** som er kalt *RelDesc*. Dette er det relasjons-sett som brukes i modellen for relasjoner. Ved å følge XLink standarden er man sikret at disse lenken vil kunne brukes i XML-tolkere ved senere utvidelse av systemet.

I relasjonsbeskrivelsen er det brukt handel (hdl://en.adresse) for å angi adresser til informasjonsobjekter. Disse kunne like godt ha vært URL'er eller URN'er. Denne definisjonen av en lenke er ikke avhengig av en DTD da den angir alle de elemen-

tene som må være med i XLink.

Tabell 7: Relasjons-sett for Relasjonsmodellen.

Relasjons beskrivelse i Xlink - *RelDesc* V1.0

```
<?xml version="1.0" encoding="UTF-8"?>
<Rel:Relation     xmlns:xlink      ="http://www.w3.org/1999/xlink/namespace/"
                  xmlns:Rel        ="http://fenris.idi.ntnu.no/namespaces/relation"
                  xlink:type       ="extended"
                  xlink:role       ="Relation"
                  xlink:title      ="Relation description">

<Rel:Source       xlink:type       ="locator"
                  xlink:role       ="source"
                  xlink:href       ="hdl://en.adresse"
                  xlink:title      ="en_tittel"/>

<Rel:Destination xlink:type        ="locator"
                  xlink:role       ="destination"
                  xlink:href       ="hdl://en.adresse"
                  xlink:title      ="en_tittel"/>
<Rel:Traversal
                  xlink:from       ="source"
                  xlink:to         ="destination"
                  Rel:direction    ="directionDescription"/>

<Rel:IsOfType>  typedescription   </Rel:IsOfType>

</Rel:Relation>
```

Rel:Relation

Elementet *Rel:Relation* er rot-elementet i relasjonen. Dette angir de *namespaces* som brukes og typeangivelse for XLink. (Tabell **8**)

Tabell 8: Attributtsett for Rel:Relation.

Attributter	Beskrivelse
xmlns:xlink	Angir navnerommet til XLink.(*fastsatt*)
xmlns:Rel	Angir navnerommet til Relation (*fastsatt*)
xlink:type	Angir at dette er en XLink lenke av typen *extended*. (*fastsatt*)
xlink:role	Angir en rolle. Denne er brukt av systemet som tolker lenken. Denne er satt til *Relation* for å angi at det er en relajon.(*fastsatt*)

Tabell 8: Attributtsett for Rel:Relation.

Attributter	Beskrivelse
xlink:title	Angir tittel på lenken. Denne er variabel og i prototypen av systemet reservert for senere bruk.

Rel:Source

Elementet *Rel:Source* beskriver kilden i lenken. Elementet angir blant annet en adresse til et informasjonsobjekt eller annen ressurs i et nettverk.(Tabell **9**).

Tabell 9: Attributtsett for Rel:Source.

Attributter	Beskrivelse
xlink:type	Angir type av XLink element. Denne er satt til *locator. (fastsatt)*
xlink:role	Angir en rolle. Denne er satt til *Source* for å angi at dette elementet beskriver kilden. (*fastsatt*)
xlink:href	Angir adressen til kilden. Dette er en unik adresse til informasjonsobjektet.
xlink:title	Angir en tittel for kilden. Denne er reservert for senere bruk.

Rel:Destination

Elementet *Rel:Destination* beskriver destinasjonen i lenken.Elementet angir blant annet en adresse til et informasjonsobjekt eller annen ressurs i et nettverk. Angivelsen av destinasjon skiller seg fra *Rel:Source* ved at rollen er satt til “Destination”. (Tabell **10**)

Tabell 10: Attributtsett for Rel:Destination.

Attributter	Beskrivelse
xlink:type	Angir type av XLink element. Denne er satt til *locator. (fastsatt)*
xlink:role	Angir en rolle. Denne er satt til *Destination* for å angi at dette elementet beskriver destinasjonen. (*fastsatt*)
xlink:href	Angir adressen til destinasjonen. Dette er en unik adresse til informasjonsobjektet.
xlink:title	Angir en tittel for kilden. Denne er reservert for senere bruk

Rel:Traversal

Elementet *Rel:Traversal* utgjør det som i XLink kalles en *arc*. Dette elementet angir hvordan en lenke traverseres ved å angi to attributter: *from* og *to*. I dette elementet er det lagt til en attributt kalt *Rel:traverse* som angir informasjon utover traverserings informasjonen som en *arc* gir. Denne kan antyde at lenken skal gå begge veier (*Bidirectional*), ikke skal traverseres (*No*), eller følge traverserings retningen (*Yes*). En fullstendig oversikt over attributtene brukt i Rel:Traversal er gjengitt i tabell **11**. Verdiene for attributten Rel:traverse er gjengitt i tabell **12**.

Tabell 11: Attributtsett for Rel:Traversal.

Attributt	Beskrivelse
xlink:from	Angir element med attributt xlink:role=”Source”
xlink:to	Angir element med attributt xlink:role=”Destination”
Rel:traverse	Angir ekstra retnings informasjon (se tabell **12**)

Tabell 12: Verdier for attributt Rel:traverse.

Verdi	Beskrivelse
NO	Angir at lenken ikke har et traverserings forhold
YES	Angir at traversering skal følges som spesifisert av Rel:Traversal
BIDIRECT	Angir at lenken kan traverseres begge veier (likeverdighet)

Rel:IsOfType

Dette elementet angir relasjonstypen for relasjonen. Det er her valgt å ikke bruke attributter, men å angi verdier for XML-elementet. Dette er gjort fordi denne skal kunne utvides ved videre arbeid med relasjonsmodellen. Relasjonstypene som er brukt i modellen gjennomgås i kapittel **5.5.4**. Utvidelse av denne diskuteres i kapittel **7.1**

5.5.4 *Relasjonstyper i relasjons-modellen*

For å kunne uttrykke relasjoner mellom to objekter må man ha ett sett av metadata som uttrykker relasjonen, dvs. vi må ha en strukturert beskrivelse som tilkjennegir relasjonens natur. Dette er en angivelse av type relasjon vi har med å gjøre. Er det en referanse, en relasjon mellom dokument og dets metadata, etc.? Det er her viktig å finne frem til et representativt utvalg for hva man vil gi relasjoner mulighet for å uttrykke.

De data-sett som er sett på i denne avhandlingen (kapittel 2.10) er spesialisert for definerte typer data, og må utvides noe for å passe inn i relasjons-modellen. Disse løsningen har enten vært eksprimentelle eller altfor omfattende. Noen har brukt prinsippene fra Hytime standarden, men ikke direkte implementert den. IFLA-modellen har satt opp en rekke relasjons-typer, og Dublin Core har definert noen under *DC.Relation.* Bruk av *DC.Relation*-elementet ansees som eksprimentelt og det er gjort noen forskjellige forslag på dette området. Det er likevel kommet frem til en anbefalt og foreløpig beskrivelse som delvis er brukt i relasjonsmodellen. Fordelen med Dublin Core er at settet kan utvides og allikevel være kompatibelt med grunnstammen i elementsettet.

Relasjons-typer i relasjonsmodellen

Relasjonstypene som er brukt i prototypen av relasjons-systemet er beskrevet i tabell **13**. Relasjons-modellen bruker elementet *Rel:IsOfType* for å uttrykke dette. Det er meningen at dette elementet skal kunne utvides med under-elementer for å uttrykke mer komplekse forhold.

Tabell 13: Relasjonstyper i relasjonsmodellen.

Type	Beskrivelse
Version	Relasjonen lenker sammen versjoner. Dette gjelder for versjoner av samme format. *Brukes som i Dublin Core*
Part	Relasjonen lenker til en del. *Brukes som i Dublin Core*
Reference	Relasjonen er en referanse. *Brukes som i Dublin Core*
Replace	En kilde erstatter destinasjonen. *Brukes som i Dublin Core*
Format	Angir versjoner av ulike format. F.eks.: Bilde i to format - gif og jpeg. *Brukes som i Dublin Core*
Extension	Angir en utvidelse av en kilde. F.eks.: Ved forbedring av en del av en teknologi. *Egendefinert*
Creator	Angir en relasjon til en konstruktør. Dette kan være et bilde, en tekst, metadata osv om konstruktøren. *Innholdsbasert. Egendefinert*
Metadata	Angir relasjon til metadata. Dette er i de tilfeller hvor man har informasjonsobjekter med metadata og informasjonsobjekter med det metadataen beskriver. *Innholdsbasert. Egendefinert*
Similarity	Angir at man har en kilde som ligner på destinasjonen. *Innholdsbasert. Egendefinert*

5.6 Lagring og gjenhenting av relasjonsobjekter

Lagring av relasjoner foregår gjennom bruk av lagringsmodellen beskrevet i kapittel **4**. gjenhenting av relasjoner benytter seg av de funksjoner som angis i den samme modellen.

5.6.1 *Relasjonsobjekter og persistens*

I relasjons-systemet er det i tillegg definert egne funksjoner som tar seg av relasjons-transaksjon. Dette er delt i to program-klasser. Systembeskrivelse for disse er gjengitt i Appendix **4**:

- *relationStore*. En gruppe funksjoner som tar seg av innlegging av relasjoner.
- *relationRetrive*. En gruppe funksjoner som tar seg av gjenhenting av relasjoner.

Disse to bibliotekene er avhengige av å kommunisere med *libxml_in* og *libxml_out* som er definert i lagringsmodellen (kapittel **4**).

Klassen relationStore

Denne delen av relasjons-systemet tar seg av innlegging av relasjoner og har to hovedfunksjoner.

- Lagring av relasjoner gjennom bruk av ferdig XML-relasjon
- Lagring av relasjon gjennom angivelse av verdier.

Klassen relationRetrive

Denne delen av relasjons-systemet tar seg av gjenhenting av relasjoner og har to hovedfunksjoner. Gjenhenting foregår ved angivelse av unik identifikator som en handel eller URL.

- Tilbakesending av XML-dokument med de relasjoner som tilhører identifikatoren.
- Tilbakesending av ønsket verdi basert på spørring etter attributt eller element.

5.6.2 *Bruk av Handles som adresse til Informasjonsobjekt*

En av grunnstenene i DigLib systemet er Handles. Handels er en unik adresse som identifiserer en informasjonsobjekt. Dette er rett og slett en mapping mellom en unik id og en URL eller URN.

Driften av Handel systemet fra CNRI er i skrivende stund ikke implementert i DigLib-systemet. Dette betyr at implementasjonen har blitt basert på bruk av URL, men vil i fremtiden automatisk kunne migreres til å bruke Handles som adresse, da Handler blir slått opp i et eget register. Det er implementert et program som henter ut infomasjonsobjekter, og det kan slik angis en unik URL som i eksempel **14**

Eksempel 14: Bruk av programmet *get* for å hente frem informasjonsobjekter.

URL: http://fenris.idi.ntnu.no/get/fw/18.0.2361

5.7 Eksempel på relasjons-systemet

Det blir her presentert den prototype som er implementert for relasjons-systemet. Siden relasjons-systemet er tenkt som et tilleg til DigLib systemet, og ikke som en egen søkemotor ble det bygget en grensesnitt for å synliggjøre hvordan systemet kan brukes. Dette blir presentert her, med en beskrivelse av hva som foregår bak kulissene når en bruker gjør en innlegging og spørring på en unik identifikator.

5.7.1 *Innlegging av en relasjon*

Dette eksempelet tar utgangspunkt i en manuell innleggelse av relasjoner mellom metadata og bilde i bildebasen ved DigLib. Dette er en prosess som skal skje automatisk ved slike typer relasjoner. (Denne type innlegging kunne like godt vært gjort med bruk av URL'er til ulike ressurser på internett.)

I figur **5.7.1.1** vises det grensesnittet som er laget for manuell innlegging av relasjoner. Her har vi lagt inn to unike adresser:

- Source: http://fenris.idi.ntnu.no/get/fw/18.0.2136 (metadata)
- Destination: http://fenris.idi.ntnu.no/get/fw/18.0.2106 (bilde)

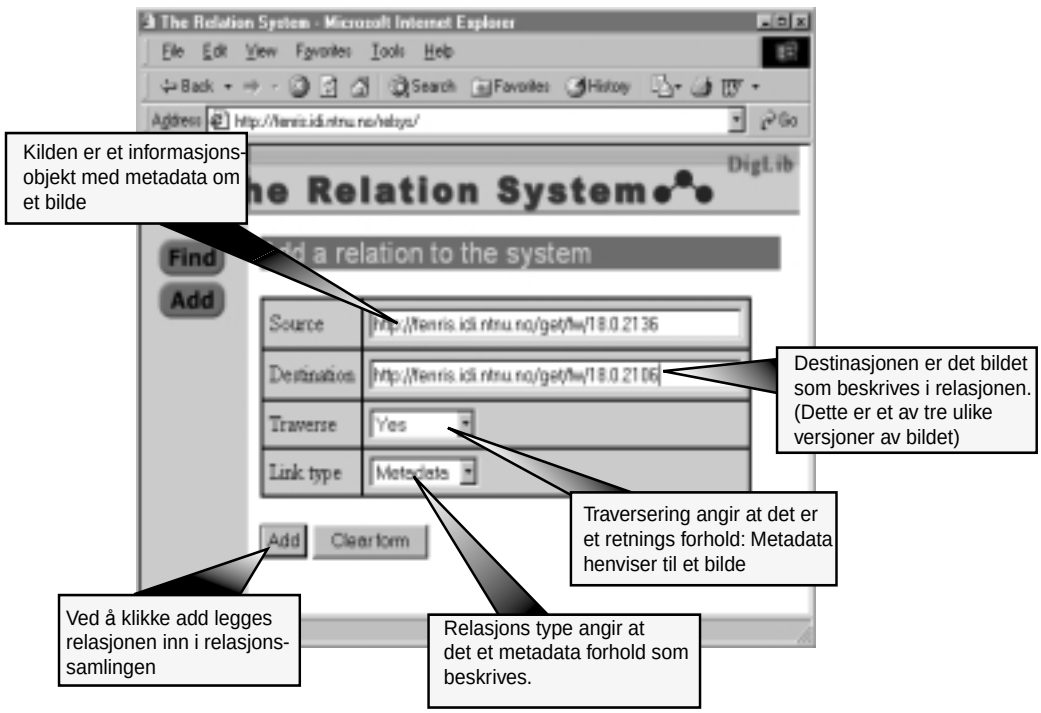
Det er angitt at man skal følge retningen som oppgis (fra *Source* til *Destination),*

og man angir relasjonstypen til “metadata”. Når dette legges inn skjer følgende transaksjon:

- Relasjonstjeneren *relAdd* som er en Java Servlet, kontaktes med opplysningene fra websiden. Det sjekkes at alt er fylt ut, og attributtene Source, Destination, Traverse, og Link Type som legges inn ved bruk av klassen *relationStore*

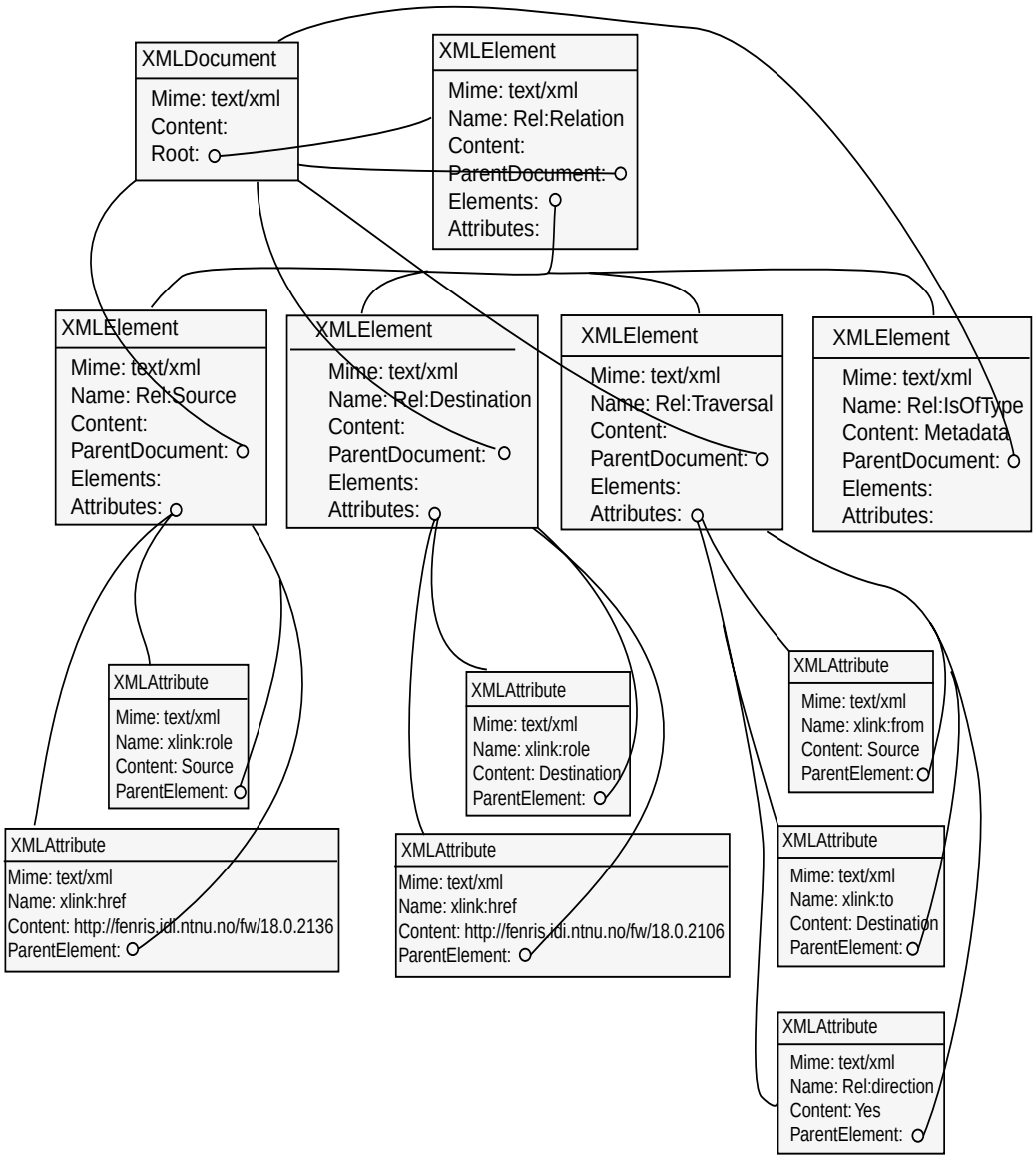
- relationStore bygger opp et xml-dokument som beskrevet i RelDesc (kapittel **5.5.3**) og benytter seg så av *libxml_in* fra lagringsmodellen for å legge inn relasjonen.

Fig 5.7.1.1 Relasjons-system - innlegging av en relasjon.



Den resulterende objektstrukturen blir da seende ut som i figur **5.7.1.2**. Denne figuren viser bare de objekter som er viktige for gjenhenting av relasjoner. Det er ikke tatt med objekter for attributtene til *Rel:Relation*-elementet. Også attributtene for *xlink:type* og *xlink:title* for elementene *Rel:Source* og *Rel:Destination* er utelatt. Denne modellen kan forekomme noe kaotisk, men er tatt med for å vise den faktiske struktur som lagres. Trådene mellom objektene viser pekere til objekter.

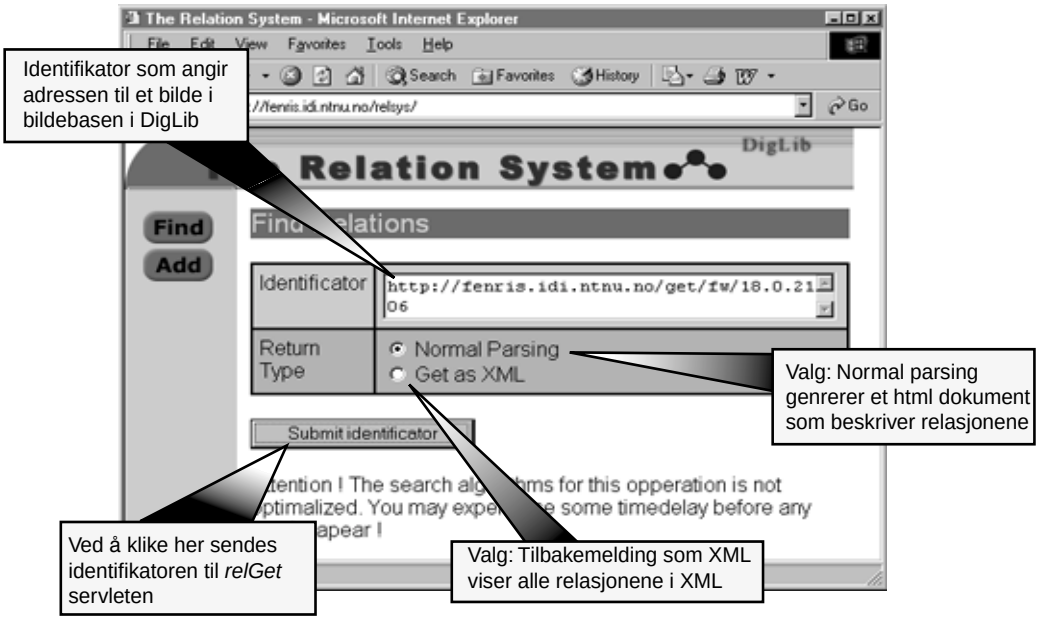
Fig 5.7.1.2 Objekter generert i lagringsmodell.



5.7.2 *Uthenting av relasjoner*

Ved uthenting av relasjoner kan man benytte det grensesnitt som presenteres i figur **5.7.2.1**. Her ser man at det er lagt inn en identifikator som korresponderer med den identifikator som tilhører informasjonsobjektet (bildet) brukt i figur **5.7.1.1**.

Fig 5.7.2.1 Relasjons-system - grensesnitt for gjenfinning av relasjoner.



Ved uthenting foregår følgende transaksjoner med det underliggende relasjons-systemet:

- Identifikatoren og type gjenhentings-visning sendes til *relGet* som er en Java Servlet. Identifikatoren sendes til *relationRetrive* som bruker *libxml_out* til å søke opp objekter som inneholder den gitte identifikator.

- Hvis visning er satt til *Get as XML* blir *relationRetrive* benyttet til å hente ut alle relasjonene som XML og genererer et stort XML-dokument som sendes tilbake til klienten.Dette er vist i figur **5.7.2.2**

- Hvis visning er satt til *Normal Parsing blir relationRetrive* benyttet til å hente ut enkelt elementer og attributter med navn og verdi. Dette sendes tilbake til *relGet* som bygger opp en HTML-side basert på disse opplysningene. Dette er vist i figur **5.7.2.3**.

Ved html visning får man muligheten til å fore de gjenhentede relasjonene tilbake til systemet slik at man kan traversere gjennom det nettverk av relasjoner som tilhører identifikatorene. Man har også muligheten til å se på informasjonsobjektet ved å klikke på identifikatoren.

Fig 5.7.2.2 Gjenhenting som xml.

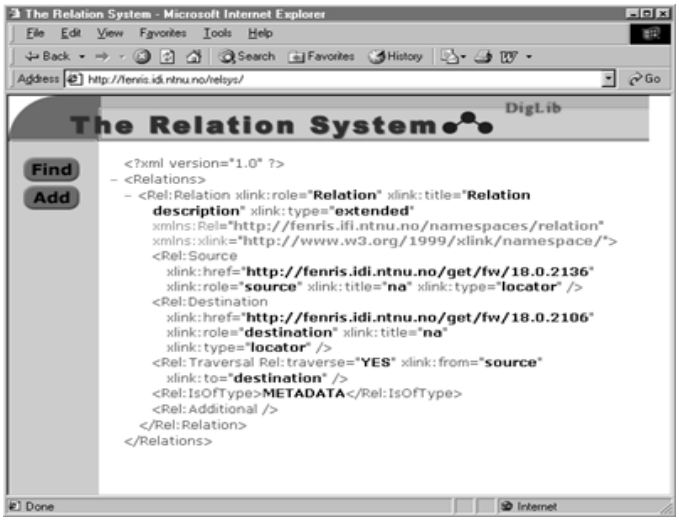
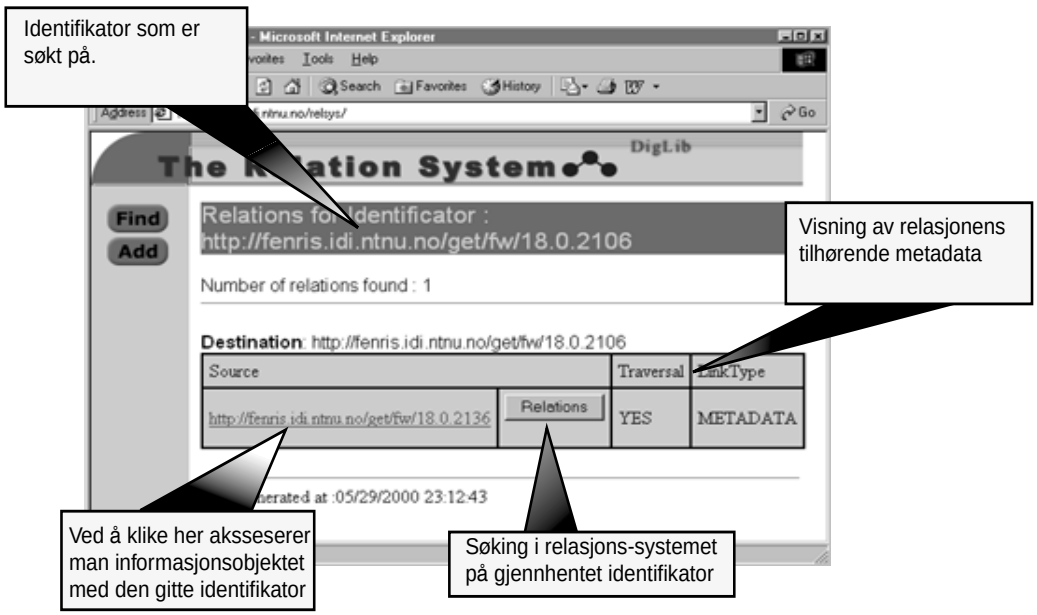


Fig 5.7.2.3 Gjenhenting som HTML.



Kapittel 6

Evaluering og erfaringer

I dette kapittelet blir det presentert de erfaringer og problemer som oppstod under arbeidet med hovedfagsavhandlingen, og forslag til videre arbeid. Følgende tema vil bli sette nærmere på her:

- Sammenligning av Dextermodellen og Arms's arkitektur for informasjon i det digitale bibliotek
- Sammenligning av modellen foreslått i denne avhandlingen og de to modellene nevnt over.
- Evaluering og erfaringer med systemet for lagring.
- Evaluering og erfaringer med modellen for relasjoner.

6.1 Oppsumering av relasjonsmodellen

I avhandlingen presenteres to modeller: en modell for relasjoner mellom informasjonsobjekter, og en modell for lagring i objektorientert databasesystem. I implementeringen av disse modellene er det ikke lagt vekt på effektivitet av algoritmer og søketid, da målet har vært å legge grunnlaget for et system basert på disse modellene.

- Relasjonsmodellen gir mulighet for relasjoner mellom ulike informasjonstyper basert på unike identifikatorer.

- Relasjonsmodellen er først og fremst tenkt som et hjelpemiddel i DigLib for lenking mellom informasjonsobjekter, men kan brukes til å knytte sammen informasjon fra ulike steder i et nettverk.

- Relasjonsmodellen består av et enkelt attributtsett implementert i XML og XLink.

- Relasjoner angir typer for å beskrive et forhold.

6.2 Kobling mellom hypermedia og digital biblioteks arkitektur

6.2.1 *En sammenligning av Dexter-modellen og Arms's arkitektur for digitale bibliotek*

Vi har i forbindelse med Dexter-modellen sett at ideen om hypermedia systemet ikke ligger langt fra ideen om det digitale bibliotekssystem beskrevet av Arms. Digitale objekt i Arms modellen og komponenter i Dextermodellen har en slående lik oppbygning. Det er interessant å se at de prinsipper som ble utviklet for hypermedia på slutten av 1980-tallet (Dexter Hypermedia Model) har blitt tatt opp i noen av de modeller som man finner innenfor det digitale bibliotek (Arms's Arkitektur).

Komponenter og digitale objekt

I Dexter modellen har man valgt å bruke begrepet komponent om den enheten som informasjon lagres i. En komponent har en nøkkeldel som består av informasjon om komponenten, blant annet dens UID (unike identifikator). En komponent har tre ulike typer: *Atom*, *link* og *composite components*. Dette er i grunnen to ulike typer komponenter da et *composite component* egentlig er et *Atom* med lenker til andre *atoms*.

I Arms's arkitektur har man brukt begrepet digitalt objekt om den enheten som lagrer informasjon. I et digitalt objekt har man også en nøkkeldel hvor en handel lagres. Et slik objekt kan inneholde et element og/eller en pakke, og en slik pakke er egentlig et digitalt objekt. Denne oppbygningen av objekter er godt sammenfallende med *components* i Dexter-modellen.

En komponent og et digitalt objekt uttrykker den samme type oppbygning. Begge bruker unike identifikatorer og inneholder nøkkelmetadata om innholdet. Innholdet kan i begge bestå av et innhold og/eller en samling med objekter/komponenter. Det som skiller disse modellene er hvordan objektene/komponentene blir lenket sammen.

Lenking

Måten man bruker lenking på i de to modellene skiller seg fra hverandre. I Dexter modellen eksisterer lenker som egne komponenter, mens Arms's arkitektur bygger lenkene inn i elementene i de digitale objektene. Dette angir igjen den forskjellen som ligger i informasjons-tenkningen for de to modellene.

Dexter-modellen er først og fremst et hypertext-system hvor bruken av lenker har en sentral plass. Dette gir et behov for å angi lenker som er mer avanserte en enkel

sammenkobling av komponenter. I Arms's arkitektur ligger hovedtyngden på oppbevaring og behandling av informasjonen, og ikke hvordan disse informasjonsbitene kan lenkes sammen gjennom relasjoner.

Lagring og behandling av informasjon

Dexter modellen ble til på et tidspunkt hvor internett og store nettverk ikke var vanlig, og derfor laget med tanke på at man hadde kontroll over hele systemet. I HTML ligger lagring utenfor systemet, det eneste krav som stilles er at aksesering av informasjon skjer gjennom HTTP-protokollen.

Dexter-modellen har foreslått en lagringsmodell som også sammenfaller godt med den måten lagring gjøres i Arms's arkitektur. Begge modellene angir et eget system/lag for lagring av informasjon med en definert akksesering og behandlings modul. I Dexter modellen har man definert et *storage layer* med en *resolver* og en *accessor* modul. I Arms's arkitektur har man et *repository* med en *repository access protocol* (RAP).

Adressering av informasjon

Unike adresser er viktig for lenking og gjenhenting av informasjon i en samling. Begge modellene har tatt høyde for dette og i Dexter modellen presenteres en *universal identificator* (UID) uten at et system for generering og tildeling av slike defineres nærmere. I Arms's arkitektur presenteres et helt system for slike unike adresser, kalt *handle system*. Disse to måtene å adressere på angir den samme tankegang: for å bruke informasjon trenger vi unike adresser.

6.2.2 *Forskjellen mellom et hypermedia system og et digitalt bibliotek*

Den mest påfallende forskjellen mellom de to presenterte modellene er bruk av lenker. Et digitalt biblioteks-system som presentert av Arms har som hovedmål å lagre og distribuere informasjon. Et hypertext system som i Dexter modellen har som hovedmål å bygge opp ressurser av flere ulike informasjons komponenter. Det er altså et skille mellom struktur og persistens, da hypermedia først og fremst er en strukturell arkitektur, mens et digitalt biblioteks-system tar for seg lagring og tilgjengelig-gjøring av informasjon. I tillegg gjør et digitalt biblioteks-system bruk av metadata om informasjonen på en standardisert måte.

Det er likevel viktig å se sammenhengen mellom disse to områdene, særlig fordi hypermedia har vært diskutert i en årrekke og slik kan bidra med verdifull informasjon ved oppbygging av nye digitale biblioteks-systemer.

Som vi har sett i denne avhandlingen kan man med fordel anvende de teknikker hypermedia angir i forbindelse med lenking og relasjoner mellom informasjon.

6.2.3 *Sammenligning med lagringsmodell for DigLib*

Modellen som er presentert i denne avhandlingen bruker prinsippene fra både Dexter modellen og Arms's arkitektur.

Fordelene med den lagringsmodell som avhandlingen presenterer er bruk av objektorienterte oppbygging. Også bruk av ny teknologi som objektorienterte databasesystem og java-objekter gir mulighet for behandling av store mengder data og transaksjoner. I en semi-strukturert objektmodell kan man også legge inn funksjoner i enkeltobjektene slik at man får intelligente objekter. Dette til forskjell Dexter modellen og Arms' arkitektur, hvor intelligens ligger i aksseseringsmoduler. Arms' modell for et repository ligger allikevel ikke langt fra det som er presentert i lagringsmodellen.

Informasjonsobjekter

Informasjonsobjekter i DigLib modellen bygger på digitale objekt i Arms's arkitektur. Disse har et hode og et innhold, hvor hodet inneholder informasjon om objektet og innholdet består av informasjon. Forskjellen er at begrepet om pakker og elementer ikke er tatt i bruk. Således er objektene atomiske, og det er opp til relasjons-systemet og ta seg av oppbygging av det Arms's kaller pakker, og Dexter modellen kaller *composite components*.

Bruk av relasjoner i DigLib

Bruken av relasjoner i DigLib-systemet samsvarer mer med et hypermedia system enn det gjør med Arms's arkitektur. Som definert i IFLA-modellen er relasjoner ikke et ukjent begrep i den digitale biblioteks-verden. Denne type relasjoner inneholder likevel ikke den type lenking som man finner i nye hypermedia arkitekturer som Xlink (kapittel 2.7.1).

Lagring av informasjon i DigLib

Storage layer i Dexter modellen og Arms's *repository access protocol* har begge gitt viktige elementer til modellen for lagring som er presentert i denne avhandlingen. Det som skiller er den bakenforliggende arkitekturen. Her er det foreslått en standardisering på semi-strukturert lagring med ad-hoc tillegg av binærlagring. I Dexter modellen har man blant annet brukt SGML for å uttrykke nøkkelmetadata og lenke informasjon. I lagringsmodellen for DigLib brukes XML for å lagre metadata og enkel tekst. Egne binærobjekter benyttes for lagring av andre mediekomponenter, men til forskjell fra de to presenterte modellene benyttes en objektorientert tankegang fra bunnen av. Blant annet arver alle objekter et superobjekt som tilsvarer den definisjon som er gitt av et informasjonsobjekt.

Behandling av informasjon i DigLib

Bruk av egne moduler for behandling av informasjon er implementert gjennom funksjoner for aksessering, men også her finner vi forskjeller fra de to øvrige modellene. I DigLib modellen er informasjon lagret i objektorientert struktur definert i java. Dette tillater objektene å inneholde funksjoner for behandling av eget innhold. Slik har vi en modell som ikke bare gir funksjonalitet til aksesseringsmodulene, men også til informasjonsobjektene. Denne funksjonaliteten ligger i samlingen, og blir foreløpig ikke eksportert ut. Dette kan allikevel gjøres gjennom den foreslåtte komponenten for distribuering i DigLib.

Bruk av unike identifikatorer i DigLib

Bruken av handles er identifisert som en del av DigLib-systemet, og da er det særlig sett på “the Handle System” fra CNRI. Dette er det samme som Arms har beskrevet i sin arkitektur.

6.3 Evaluering og erfaringer fra lagrings-modellen

Arbeidet med lagringsmodellen og den påfølgende implementering var en tidkrevende prosess, og flere utenforliggende tekniske problem oppstod. De erfaringer som man fikk fra utviklingen av modellen er gjennomgått under.

6.3.1 *Parsing av XML strukturert data*

Konstruksjon av modell for lagring ble først gjort med utgangspunkt i lagring av XML-strukturert metadata. Metadata som utgjør Trobib-samlingen i DigLib ble omstrukturert til en stor XML-fil hvor de ulike metadata-elementene utgjorde XML-elementer. Det ble konstruert et program som bruker lagrings-modellen for å legge inne hele samlinger med metadata kalt *InsertXML*. Det gikk mye tid med til å samkjøre XML-*parser* og databaselagring, og gjennom dette arbeidet ble det identifisert problemer som ble løst gjennom revidering av lagringsmodellen.

Innlegging av XML-*parser* objekter vs. strukturering i egne klasser

Det ble identifisert muligheten for å legge inn de objekter som XML-parseren genererer utifra et XML-dokument. Denne løsningen gir et vell av objekter som mangedobles den informasjonsmengde som legges inn. I tillegg er en slik innlegelse av system-spesifikke objekter versjonsavhengig. Ved å benytte denne type løsning binder man seg til den versjon XML-parseren er i, når dokumentet bli lagret.

Ved å velge en enklere og egendefinert løsning blir informasjonsmengden holdt på et minimum, og man er ikke avhengig av spesifikk versjon eller program. Denne

løsningen er ikke så fleksibel når det gjelder mulighet for innleggelse av alle typer XML-tillegg som DTD'er og XSL.

Lenking til *Parent document*

Det ble nødvendig å legge inn et ekstra attributt i *XMLElement* klassen som lenket til det *XMLDocument*-objektet som elementet tilhørte. Selv om et *XMLElement* tilhørende et *XMLDocument*-objekt kan finnes ved å traversere fra *XMLDocument->Root*, var det nødvendig med en lenke andre veien fordi man ved VQL-*queries* for å finne spesifiserte elementer ikke hadde muligheten til å vite hvilket *XMLDocument*-objekt, elementet tilhørte.

Objekt lenking vs. lister for ulike objekt-typer

I den første prototypen ble det brukt overordnede lister som lenket til de ulike objekt-typene. Dette ble gjort for å raskt kunne finne frem til *XMLElement*-objekter med binært innhold. Slike lister ble utelatt når prototypen ble implementert for Versant database-systemet, da dette tillater VQL-spørringer på objekt-typer.

6.3.2 *Lagring av binærfiler*

Lagring av binære filer som bilder, lyder og pdf-filer skjer gjennom bruk av klassen *BinaryDocument* i lagringsmodellen. Koblingen mellom denne og semi-strukturert lagring av metadata skjer gjennom bruk av relasjonsmodellen.

Modellen som først ble implementert tok i bruk angivelse av lenke fra metadata til *BInaryDocument*-objektet som var angitt i metadataen. Denne løsningen krevde at XML-dataen brukte ett reservert ord satt til "binary" som i eksempel **15**. Det ble utviklet en DTD som ble brukt i forbindelse med denne. Binærfilen ble internt i databasen lagret i et objekt av typen *XMLElement* hvor *content* attributtet ble satt til binærfilen. Dette skapte store problemer ved uthenting av informasjon fra databasen. Det måtte skrives et spesial program som kunne skille vanlige *XMLElement*-objekter fra elementer med binært innhold. Denne måten å lagre på ble forkastet til fordel for å ta i bruk objekter av typen *BinaryDocument* til lagring av binærfiler, og relasjonsmodellen ble brukt for å knytte den XML-strukturerte data sammen med binær-objektet. Dette er beskrevet i gjennomgang av lagringsmodell

kapittel **4**, og relasjonsmodell kapittel **5**.

Eksempel 15: Bruk av "binary" som reservert ord i XML-strukturert metadata.

```
<Post>
    <Navn>Einar Lichtenberg</Navn>
    <binary href="bilder/einar.jpg" internal="yes">
        <mime>image/jpeg</mime>
    </binary>
</Post>
```

Lagring av store binærfiler.

I forbindelse med testing av lagringsmodellen mot samlingen *Spirit of the Vikings* oppstod to problemer. De ene problemet var begrensninger i XML-lagring i O2 databasen. Her brukes en objekt-cache som var satt til 1 gigabyte, dette ble for lite ved innleggelse av flere store binærfiler, og følgelig krasjet databasen med store konsekvenser for andre samlinger som var lagt inn. Dette problemet skyldes i all sannsynlighet konfigureringen av database-systemet. Dette ble ikke utforsket videre da man gikk over til bruk av Versant ODBMS.

Ved innleggelse i Versant ODBMS opplevde man problemer med overføring av store binærdata fra Java til databasen. Dette problemet skyldes at man ikke kan oversendte hele binærdataen i et stykke, men må dele det opp gjennom å bruke en *vector* med *binary arrays*. Dette blir tatt opp i forslag til utvidelse (kapittel **7.5**).

6.3.3 *Bruk av XML-baserte databaser*

En rekke nye produkter innenfor XML-baserte databaser har blitt lansert mot slutten arbeidet med denne avhandlingen. Slike databaser er ofte basert på relasjonsdatabaser med XML som lagringsmodell. Man kunne valgt å bruke en slik database, men det ville være begrensende i forhold til kontroll over lagring av binærfiler, og innpassing av distribuerte tjenester. Slike databaser har ingen muligheten for innlegging av funksjoner i informasjonsobjektene.

6.4 Evaluering og erfaringer fra relasjonsmodellen

De erfaringer som presenteres her er gjort utifra enkel bruk av systemet, og det gjenstår å teste dette med ett større antall samlinger. Dette har ikke vært mulig under arbeidet med avhandlingen, da disse samlingene er del av andre prosjekter, og følgelig ikke var lagt inn på dette tidspunkt. Når disse legges inn, kan man allikevel knytte relasjoner ved å benytte den modell og det system som er beskrevet i

denne handlingen.

6.4.1 *Bruk av relasjoner mellom informasjonsobjekter*

Adressering av informasjonsobjekter i relasjoner

Innføringen av relasjons-systemet har ført til en annen lagrings metode enn den som først ble implementert (se kapittel **6.3.2**). Det er et krav at man kjenner til identifikatoren til et informasjonsobjekt før man knytter en relasjon. Dette krever at man har et handel system på plass som kan generere nye handler når et informasjonsobjekt legges inn. Dette var ikke på plass når denne avhandlingen ble gjort, og følgelig ble identifikatorene bassert på URL'er. En unik identifikator ble muliggjort ved implementering av et program som tar i mot databasenavn og en objektid innbakt i en URL (eksempel **16**), og sender tilbake informasjonsobjektet.

Eksempel 16: Unike identifikatorer ved bruk av URL og *get*-program.

```
http://fenris.idi.ntnu.no/get/databasenavn/objektid
```

Administrering av relasjoner

Det har vist seg at innføring av relasjoner gjør det enklere å administrere knyttingen mellom de ulike informasjons-typene som en samling består av. Man slipper å endre informasjon i metadata-dokumenter, og til eksempel kan man i *bildebasen* hvor man har en metadata-post og tre ulike versjoner av et bilde enkelt administrere relasjonene mellom disse som tidligere lå innbakt i metadataen.

6.4.2 *Nytteverdien av relasjoner.*

Relasjoner for lenking mellom informasjonsobjekter i en samling

Nytteverdien av denne type bruk av relasjoner er klar: dette letter arbeidet med sammenknytting av ulike versjoner og informasjonstyper. Dette kommer særlig godt frem ved benyttelese i bildesamlingen ved DigLib hvor man kan angi relasjoner mellom metadata-dokumentet og de ulike versjonene av bildene.

Oppdaging av informasjon

Nytteverdien av relasjons-systemet ved oppdaging av informasjon avhenger av at man har en stor mengde relasjoner som kan brukes. For å gi mest mulig verdi for sluttbruker bør dette bestå av relasjoner ikke bare konstruert av maskinprosessering, men også gjennom brukergenerering. Denne delen av relasjons-systemet er ikke særlig utprøvd og blir tatt opp i kapittel **7.2.2**.

6.4.3 *Metadatasett for relasjoner*

I relasjonsmodellen er det fokusert på to metadata-elementer i relasjonen: beskrivelse av retning og type. Utvidelser til dette settet er diskutert i kapittel **7.1**.

Retningsbeskrivelse

Angivelse av retning gir mulighet for å uttrykke komplekse lenke forhold. Det er imidlertid ikke klart om dette er nødvendig og forståelig i alle tilfeller. Blant annet har det vist seg at ved angivelse av enkelte typebeskrivelser blir attributtet for retning noe meningsløst. Retningsbeskrivelsen har hatt mest nytte i beskrivelse av relasjoner mellom informasjonsobjekter med hensyn på versjoner. Her kan man angi om versjonene er likestilte (ingen retning) eller om det er snakk om et original/versjon forhold.

Typebeskrivelse

Beskrivelse av type kan med fordel utvides med flere elementer, men det er viktig at disse elementener er entydige og klart avgrensede. Det har blant annet vært gjennomgått relasjonstypene angitt i Dublin Core (kapittel **2.10.3**). I relasjonsmodellen har det også blitt brukt typer som går på innholdet av informasjonsobjektet, til eksempel “metadata” som angir at et informasjonsobjekt er en metadata relasjon.

Kapittel 7

Utvidelser og videre arbeid

Under presenters noen av de mulige utvidelser og forbedringer som kan gjøres i lagrings- og relasjons-systemet. Dette inkluderer også momenter som er identifisert i evaluering og erfaringsdelen. Det blir her gjennomgått følgende punkter:

- Metadata
- Generering av relasjoner
- Bruk av distribusjon
- Analyse av relasjoner
- Andre mulige utvidelser

7.1 Utvidelse av metadatasett for relasjoner

Metadatasettet for relasjoner i denne avhandlingen er blitt begrenset for å holde systemet enklest mulig, og avhandlingen har fokusert på de mest nødvendige elementer: angivelse av retning og relasjons-type. Andre mulige metadata-elementer har vært sett på, men ikke utprøvd.

7.1.1 *Tidsmerking*

En mulig utvidelse til dette ville være å innføre metadata-elementer som *tidsmerking* slik man har gjort i blant annet DLS **[Hitchcock1998]** beskrevet i kapittel **2.8.3**. Bruk av tidsmerking vil blant annet lette administrering av relasjoner slik at man ikke får konflikterende relasjoner ved brukergenrering. Det vil også være mulig å angi alder, og trekke slutninger om hvor relevant lenken er.

7.1.2 *Definering av konstruktør*

Angivelse av en konstruktør *(Creator)* ville kunne gi kontroll

med hvem som har rettigheter og er ansvarlig for relasjonen. Dette er særlig viktig i systemer hvor man tillater brukergenererte relasjoner.

7.1.3 *Valg av metadataformat*

Det kunne også vært ønskelig å innført en angivelse av hvilket metadatformat som er brukt, f.eks. *Dublin Core* eller *Ad-Hoc* etc.

7.1.4 *Annet*

Andre mulige metadata utvidelser kunne også vært sett på, men det er viktig at et slikt metadatasett for relasjoner ikke blir for omfattende. Hvis man har for mange elementer blir det vanskeligere å prosessere relasjonene, og innlegging av nye relasjoner blir en tidkrevende og komplisert prosess. Problemer med dette har man hatt i blant annet Antworld **[AntWorld]** hvor innlegging av nye "maur-spor" er så omfattende at brukere ikke benytter systemet.

7.2 Generering av relasjoner

I prototypen for relasjoner legges relasjoner inn gjennom menneske genererte relasjonsposter. Tanken er at systemet etter hvert skal brukes i systemer for samlinger hvor slik generering kanskje automatisk, og ved brukergenerering.

7.2.1 *Automatisk generering*

I tilfeller hvor det er snakk om rene XML eller HTML dokument kan man tenke seg at dokumentet blir *parset* for lenker og disse lenkene automatisk legges inn som relasjoner. Man kan også velge hvor mange nivåer man skal traversere i en slik lenkestruktur, f.eks. angi at man skal stoppe hvis en lenke går mer enn et dokument utenfor samlingen.

Innholdsbasert generering

Generering av relasjoner utfra innhold kan gjøres ved gjennomgang av metadata og relatering på nøkkelord som blir sett på i kapittel **2.1.4**. En slik mulighet må allikevel begrenses da dette kan gi en så stor mengde relasjoner at systemet blir mettet. Parsing av binærfiler er også mulig, men krever mer avanserte programmer og funksjoner.

Parsing av hele databaser

Ved masse-innleggelse av hele samlinger kan slike relasjoner bygges inn i den XML-filen som representerer samlings-metadata. I tilfeller hvor man bare har binærobjekter kan man tenke seg en egen XML-fil som forteller om hvilke relasjoner som finnes.

7.2.2 *Brukergenerering*

Brukergenererte relasjoner kan gi verdifull utvidelse av et administrert relasjonssystem. Hvis man lagrer brukernavn i relasjonen vil de relasjoner som er brukergenerert lett kunne skilles ut slik at de ikke konflikter med system genererte relasjoner. Bruk av slike relasjoner kan gi informasjon utover de relasjonstyper som forekommer ved automatisk genrering. Man kan også tenke seg at brukere kan opprette egne relasjons-samlinger for sin forskning, og dele de relasjoner som de ønsker å dele med andre. **[Hitchcock1998]**

Brukergenerering er et interessant område som bør sees nærmere på i en utvidelse av systemet presentert i denne avhandlingen.

7.2.3 *Generering av identifikatorer for relasjoner*

Generering av unike identifikatorer som kan brukes mot relasjoner gir en ny dimensjon i relasjons-systemet. Man kan da bygge relasjoner som lenker mellom andre relasjoner. Man kan også tenke seg at det kan henvises til relasjoner fra andre dokumenter. Dette er et område som må studeres nøyere for å finne konsekvensene av slik bruk.

7.3 Bruk av distribusjon for relasjoner

I modellen for DigLib som presenteres i denne avhandlingen er det foreslått bruk av distribusjon som et viktig element i utveksling av informasjon. Distribuering av relasjoner på samme måte kan være en aktuell mulighet, særlig fordi man har mulighet til å sende “intelligente” objekter over et nettverk. Med intelligente objekter menes her at objektet inneholder funksjoner som kan brukes på innholdet av objektet.

7.3.1 *Valg av distribuert arkitektur*

DigLib-prosjektet legger opp til at CORBA skal brukes som grunnlegende arkitektur for distribusjon av informasjonsobjekter. For at distribusjon skal kunne standardiseres i DigLib må det utvikles en protokoll for hvordan de ulike kompo-

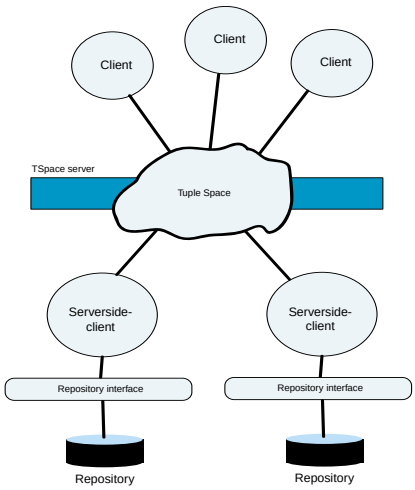
nentene bruker CORBA. Utvikling av en slik protokoll inngår ikke i denne avhandlingen.

7.3.2 *Distribusjon ved hjelp av Tspace*

Det ble i denne avhandlingen utprøvd en enkel implementering av en distribuert applikasjon bygd på Tspace teknologi fra IBM **[Tspace]** . Dette tilbød flere gode muligheter når det gjelder *caching* av relasjons objekter, men viste seg praktisk å være mindre hensiktsmessig grunnet krav til klienten. Løsningen som ble utprøvd krever at man har en egen applikasjon på klinetsiden som kan kommunisere med en server for Tspace.

Modell **7.3.2.1** er viser hvordan Tspace teknologien kan implementeres i relasjons-systemet. Modellen viser flere klienter som gjennom et felles *tuplespace* (tuppelrom) kan koble seg mot ulike relasjons-servere. Hver relasjons-server kommuniserer med tuppelrommet og formidler forespørsler mellom dette og det underliggende databasegrensesnittet. På denne måten kan ulik databaseteknologi benyttes, så lenge det som oversendes gjennom tuppelrommet er standardisert i form av XML eller annen protokoll.

Fig 7.3.2.1 Distribusjon av relasjoner ved bruk av Tspace.



Et tuppelrom er en slags database (*cache)* som ligger i hovedhukommelsen til serveren. Det man lagrer i denne databasen kalles tupler. En tuppel vil typisk se slik ut: (“Nøkkel element”, “data-element”, “data-element”), og har følgende egenskaper:

- Tuppelet kan bestå av opptil flere data-element og det er ikke nødvendig at det første elementet er nøkkel-element, eller at noen av elementene er nøkkelelement.

- Elementene kan ha navn som identifiserer feltposisjon.

- Når man skal få tak i et tuppel kan man lage en *template* som består av antall ukjente element og antall kjente element (f.eks et nøkkel-element) TSpace serveren vil da lete opp aktuelle tupler som passer templaten og sende disse tilbake til Klienten. Hvis disse ikke finnes ennå kan TSpace serveren sende tilbakemelding (*callback*) til klienten når disse tupplene blir skrevet inn.

- Man kan også benytte seg av søking i tuppelrommet gjennom et SQL lignende spørrespråk. Et hvert tuppel som skrives inn kan også gis en livslengde (*expiry time*), dvs. man kan angi hvor lenge tuppelet skal eksistere før det slettes av serveren.

Relasjons-tjeneren vil være knyttet til et tuppelrom som er fellesrommet for alle tilknyttede servere og klienter. Dette rommet har som hovedoppgave å lagre forespørsler (*requests*) fra klienter som relasjons-tjeneren kan lese. Slike forespørsler ser ut som angitt i eksempel **17**.

Eksempel 17: Tuppel for request.

```
("request", ClientID, request_type, content)
```

I dette eksempelet har man følgende data-element:

- *Request* er et nøkkel-element som er likt for alle forespørsler fra en klient.

- *ClientID* er et unikt navn som identifiserer klienten. Dette navnet opprettes i det klienten kobler seg til tuppelrommet.

- *Request_type* har to mulige verdier: “get” eller “add” som angir om klienten ønsker å hente ut en relasjon eller legge til en relasjon.

- *Content* er innholdet eller pakken som klienten sender til serveren. I tilfeller hvor forespørselen er “get” vil den inneholde en URI. I tilfeller hvor forespørselen er “add” vil den inneholde et XML-fragment med en fullstendig relasjon.

Når relasjons-tjeneren leser en forespørsel av typen “get” bruker den inneholdet av *content* i en spørring mot databasen. Det resultatet som den får samles så i en

XML fil og skrives til et nytt tuppelrom som har samme navn som klientnavnet. Dette tuppelrommet kan bare leses av tjeneren og av klienten som gjorde forespørselen. Gjennom bruk av en spesiell XML lagring i tuppelrommet kan klienten kjøre XQL spørringer mot resultatet fra tjeneren, og således filtrere ut informasjon som man er på jakt etter (f.eks. bare relasjoner inn til dokumentet, eller bare relasjoner ut fra dokumentet)

I et slikt systemet spiller det liten rolle hvilket system som ligg under for persistent lagring av relasjoner.

Forslag til protokoll for denne utvidelsen finnes i Appendix **7**

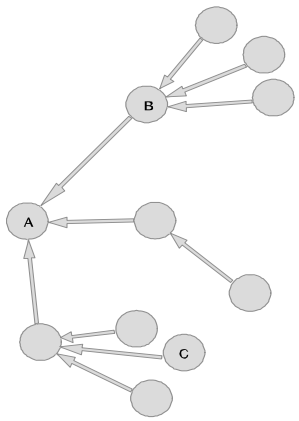
7.4 Analyse av relasjoner

I denne delen blir det presentert forslag til utvidelse som omhandler hvordan en kan analysere relasjoner for å trekke ut viktig informasjon. Dette dreier seg i så måte om hvordan en kan gi intelligent tilbakemelding til en bruker, for å gi bedre presisjon ved gjenfinning. Datamining er et begrep som ofte blir brukt om slik analyse. De mulighetene som blir sett på her er ikke implementert, og fremdeles på et konseptuelt nivå.

Informasjonshenting fra analyse av relasjons nett

Et relasjonsnett er det nettverket av lenker som oppstår når man har et sett av relasjoner mellom informasjonsobjekter i et objekt-univers. Dette relasjons nettet lenker sammen både direkte og indirekte, dvs. gjennom flere ledd.

Fig 7.4.0.1 Relasjonsnett.



Figur **7.4.0.1** angir et relasjonsett for informasjonsobjekt A. Dette nettverket av relasjoner kan gi oss en del viktig informasjon. Dette er noe av det som er blitt gjort i blant annet BibRelEx systemet. **[Landgraf1999]**

- Relasjon A er en autoritet i dette dokument universet
- Relasjonen mellom A og B er sterk (direkte relasjon mellom A og B)
- Relasjon mellom B og C er svakere (1. mellomledd mellom A og C)

Man kan også angi relevans til et informasjonsobjekt ved å se på antall relasjoner som lenker til informasjonsobjektet. En lignende relevans gjennom antall referanser er gjort i søkesystemet *Google* **[Google]** , hvor antall lenker til en side avgjør hvor relevant en side er. Hvis ingen lenker til siden kan man anta at den ikke ar så viktig som en side med mange lenker til seg.

7.5 Andre utvidelser

Flere mulige utvidelser finnes, og både lagringsmodellen og relasjonsmodellen er prototyper som gjennom videre forskning kan forbedres. Den arkitektur som er presentert i denne oppgaven er ment å kunne brukes i videre arbeid med DigLib-prosjektet. Det er forfatterens håp at relasjoner blir en integrert del i det fremtidige biblioteks-system som er under utvikling.

Kapittel 8

Referanser

8.1 Mal for referanser.

Malen for referanser er hentet fra artikler publisert hos D-Lib Magazine.

[ReferanseNøkkel]
Navn på forfattere, "Tittel på verk", *Hvor artikkelen forekommer* Dato.
< http:// eventuell adresse på web >

8.2 Internett-ressurser brukt i avhandlingen.

[AntWorld]
"Antworld"
< http://aplab.rutgers.edu/ant/ >
(Sjekket 20.mai 2000)

[Ardent]
"Ardent Software"
< http://www.ardent.com >
Notis : Database systemet O2 er ute av produksjon, og Ardent har fjernet produktsidene for dette database systemet.

[CORBA]
"The OMG's site for Corba"
< http://www.corba.org/ >
(Sjekket 20.mai 2000)

[CNRI]
"The handle system"
< http://www.handle.net/ >
(Sjekket 20.mai 2000)

[Google]
"Google"
< http://www.google.com/ >
(Sjekket 30.mai 2000)

[Javaspaces]
"Javaspaces(TM) Technology"
< http://java.sun.com/products/javaspaces/index.html >
(Sjekket 20.mai 2000)

[Multicosm]
"Multicosm"
< http://www.multicosm.com/welcome.htm >
(Sjekket 30.mai 2000)

[Oasis]
"The XML cover pages"
http://www.oasis-open.org/cover/sgml-xml.html
(Sjekket 30.mai 2000)

[RMI]
"Remote Method Invocation"
< http://java.sun.com/products/jdk/rmi/index.html >
(Sjekket 20.mai 2000)

[Tspace]
"Tspaces"
< http://www.almaden.ibm.com/cs/TSpaces/ >
(Sjekket 20.mai 2000)

[Versant]
"Versant Corporation"
< http://www.versant.com/ >
(Sjekket 20.mai 2000)

[Xerces-C]
"Xecres C++ Parser"
< http://xml.apache.org/xerces-c/index.html >
(Sjekket 20.mai 2000)

[XML4C]
"XML for C++ Parser"
< http://www.alphaworks.ibm.com/tech/xml4c/ >
(Sjekket 20.mai 2000)

[XML4J]
"XML for Java"
< http://www.alphaworks.ibm.com/tech/xml4j/ >
(Sjekket 20.mai 2000)

8.3 Støtte-litteratur for implementasjon.

[Alhir]
Sinan Si Alhir, "UML in a nutshell". *O'Reilly & Associates,* September 1998.
ISBN: 1-56592-448-7

[Flanagan]
David Flanagan, "JAVA in a nutshell". *O'Reilly & Associate,* 1997.

[Hunter]
Jason Hunter, William Crawford, "Java Servlet Programming", *O'Reilly & Associate,* October 1998.
ISBN: 1-56592-391-X

[Versant1]
"Java Versant Interface 2.4.0 Usage Manual". *Versant Object Technology Corporation* 1999.

[Versant2]
"Versant Database Administration Manual". *Versant Object Technology Corporation* 1999.

8.4 Artikkel referanser brukt i hovedfagsavhandlingen.

[Aalberg]
Trond Aalberg, "IDI DIGLIB : Komponentbasert arkitektur for forskning og utvikling av digitale biliotek", *IF-IDI* 2000

[Arms1997]
William Y. Arms, C. Blanchi, and E.A. Overly "An architecture for Information in Digital Libraries". *Dlib Magazine*, February 1997.
< http://www.dlib.org/dlib/february97/cnri/02arms1.html >
ISSN: 1082-9873
(Sjekket 20.mai 2000)

[Abiteboul2000]
Serge Abiteboul, Peter Buneman, Dan Suciu, "Data on the Web: From Relations to Semistructured Data and XML", *Morgan Kaufmann Publishers*, 2000
ISBN: 1-55860-622-X

[Berners-Lee1]
Tim Berners-Lee "Semantic Web Road Map"
< http://www.w3.org/DesignIssues/Semantic.html >
(Sjekket 30.mai 2000)

[Berners-Lee2]
Tim Berners-Lee, "The Semantic Web"
< http://www.w3.org/DesignIssues/ >
(Sjekket 20.mai 2000)

[Bosak1997]
Jon Bosak, "XML, Java and the future of the web", *Sun microsystems* 1997
< http://metalab.unc.edu/pub/sun-info/standards/xml/why/xmlapps.html >
(Sjekket 20.mai 2000)

[Bush1945]
Vanevar Bush, "As we may think", *The Atlantic Monthly*, July Issue 1945
< http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm >
(Sjekket 20.mai 2000)

[Caplan1]
Priscilla Caplan and William Y. Arms “Reference Linking for Journal Articles”, *D-Lib Magazine Volume 5 Number 7/8*, July/August 1999.
< http://www.dlib.org/dlib/july99/caplan/07caplan.html >
(Sjekket 20.mai 2000)

[Carr1]
L.Carr, H.C. davis, D. De Roure and W. Hall, "Aplication-Independent Link Processing"
< http://www.mmrg.ecs.soton.ac.uk/publications/archive/carr1998c/ >
(Sjekket 24.mai 2000)

[Carr2]
L. A. Carr, D. De Roure, W. Hall & G. Hill, "Implementing an Open Link Service for the World-Wide Web" . *World Wide Web Journal, 1*
< http://www.bib.ecs.soton.ac.uk/data/disk0/00/00/14/40/pdf/carr1998b.pdf >
(Sjekket 24.mai 2000)

[Clark]
James Clark, "XML Namespaces"
< http:// www.jclark.com/xml/xmlns.html >
(Sjekket 20.mai 2000)

[Cover99]
R. Cover, " XML linking and Adressing Languages (XPath, XPointer, Xlink)".
< http://www.oasis-open.org/cover/xll.html > *(10.mai 2000)*
(Sjekket 20.mai 2000)

[Davis92]
H.Davis, W. Hall I. Heath G. Hill and R. Wilkins, "Microcosm: An Open Hypermedia Environment for Information Integration."
< http://www.mmrg.ecs.soton.ac.uk/publications/archive/htichcock1992a/html/ >
(Sjekket 20.mai 2000)

[Davis1999]
Klaus Marius Hansen, Christian Yndigegn, Kaj Grønbæk, "Dynamic use of Digital Library Material - Supporting Users with Typed Links in Open hypermedia", *Springer, LNCS 1696, ECDL'99 Proceedings* 1999, s. 254-273.

[DC-Element_Set]
"Dublin Core Metadata Element Set, Version 1.1: Reference Description"
< http://purl.oclc.org/dc/documents/rec-dces-19990702.htm >
(Sjekket 20.mai 2000)

[DC-Qualifiers]
"Approval of initial Dublin Core Interoperabiity Qualifiers"
<http://www.mailbase.ac.uk/lists/dc-general/2000-04/0010.html> *(24.mai 2000)*
(Sjekket 20.mai 2000)

[DC-Relation]
"Relation Element Working Draft 1997-12-19"
< http://purl.org/dc/documents/wd-relation-current.htm > *(24.mai 2000)*
(Sjekket 20.mai 2000)

[DLS1]
Leslie Carr, David De Roure, Wendy Hill, Gary Hill, “The Distributed Link Service : A Tool for Publishers, Authors and Readers”
< http://www.w3.org/Conferences/WWW4/Papers/178/ >
(Sjekket 20.mai 2000)

[DLS2]
David De Roure, Samhaa El-Beltagy, Les Carr, Wendy Hall, “A Distributed Link Service using Query Routing”
< http://www.ecs.soton.ac.uk/~dder/qdls/ >
(Sjekket 20.mai 2000)

[Halasz1994]
Frank Halasz, Mayer Schwartz, "The Dexter Hypertext Reference Model", *Communications of the ACM* , February 1994/Vol.37, No.2

[Hall95]
W. Hall, L Carr, D. De Roure, "Linking the World Wide Web and Microcosm"
< http://www.mmrg.ecs.soton.ac.uk/publications/archive/hall1995/html/ >
(Sjekket 20.mai 2000)

[Hardman1994]
Lynda Hardman, Dick C.A. Bulterman, Guido Van Rossum, "The Amsterdam hypermedia model : Adding time and context to the dexter model.", *Communications of the ACM*, February 1994/Vol.37, No.2

[Harold1998]
Elliotte Rusty Harold, “XML: Extensible Markup Language”, *IDG Books Worldwide, Inc.*, 1998
ISBN: 0-7645-3199-9

[Hitchcock 1997a]
S. Hitchcock et. al., “Linking Everything to Everything : Journal Publishing Myth or Reality ?”
< http://www.mmrg.ecs.soton.ac.uk/publications/archive/hitchcock1997a/html/ >
(Sjekket 24.mai 2000)

[Hitchcock1997b]
S. Hitchcock, L. Carr, S. Harris, J.M.N. Hey and W. Hall, "Citation Linking : Improving Access to Online Journals"
< http://www.mmrg.ecs.soton.ac.uk/publications/archive/htichcock1997b/html/ >
(Sjekket 24.mai 2000)

[Hitchcock1998]
Steve Hitchcock, Les Carr, Wendy Hall, Stephen Harris, S.Probets, D.Evans, D.Brailsford, "Linking electronic journals: Lessons from the Open Journal project.", *D-Lib Magazine*, December. 1998.
< http://www.mmrg.ecs.soton.ac.uk/publications/archive/hitchcock1998/html/ >
(24.mai 2000)

[Hitchcock1998a]
S. Hitchcock, F. Quek, L. Carr, W. Hall, A. Witbrock and I. Tarr, "Towards Universal Linking for Electronic Journals", *Serials Review, 24, 1*, 1998, s. 21-33
< http://www.mmrg.ecs.soton.ac.uk/publications/archive/htichcock1998a/html/ >
(Sjekket 24.mai 2000)

[Hitchcock1998b]
S. Hitchcock, R. Kimberley, S. harris, L. Carr and W. Hall, "Web of research : putting the user in control"
< http://www.mmrg.ecs.soton.ac.uk/publications/archive/hitchcock1998b/html/ >
(Sjekket 24.mai 2000)

[HTML4]
"HTML 4.01 Spesification"
< http://www.w3.org/TR/html4/ >
(Sjekket 30.mai 2000)

[HTML4_Links]
"HTML 4.01 Spesification - Links"
< http://www.w3.org/TR/html4/struct/links.html >
(Sjekket 30.mai 2000)

[HyTime]
"HyTime: ISO 10744 Hypermedia/Time-based Structuring Language"
< http://dmsl.cs.uml.edu/standards/hytime.html >
(Sjekket 30.mai 2000)

[IFLA98]
IFLA Study Group on the Functional Requirements for Bibliographic Records, "Functional Requirements for Bibliographic Records", *UBCIM Publications* – New Series Vol 19.
< http://www.ifla.org/VII/s13/frbr/frbr.pdf >
(Sjekket 20.mai 2000)

[Landgraf1999]
Anne Brüggermann-Klein, Rolf Klein, Britta Landgraf, "BibRelEx - Exploring Bibliographic Databases by Visualization of Annotated Content-Based Relations", *D-Lib Magazine Volume 5 Number 11*, November 1999, ISSN 1082-9873
< http://www.dlib.org/dlib/november99/landgraf/11landgraf.html >
(Sjekket 20.mai 2000)

[Lichtenberg]
Einar W. M. Lichtenberg, "Tilrettelegging av XML-modell for bruk i objektorientert database. XML-o2-matic(XOM)" - , *IF-IDI*, 1999

[Moore1]
Regan Moore, Chaitan Baru, Arcot Rajasekar, Bertram Ludaescher, Richard Marciano, Michael Wan, Wayne Schroeder, and Amarnath Gupta, "Collection-Based Persistent Digital Archives -Part 1", *D-lib Magazine Volume 6 Number 3*, March 2000.
< http://www.dlib.org/dlib/march00/moore/03moore-pt1.html >
(Sjekket 30.mai 2000)

[Moore2]
Regan Moore, Chaitan Baru, Arcot Rajasekar, Bertram Ludaescher, Richard Marciano, Michael Wan, Wayne Schroeder, and Amarnath Gupta, "Collection-Based Persistent Digital Archives -Part 2", *D-lib Magazine Volume 6 Number 4*, April 2000.
< http://www.dlib.org/dlib/april00/moore/04moore-pt2.html >
(Sjekket 30.mai 2000)

[OpenJournal]
"The Open Journal Project"
< http://journals.ecs.soton.ac.uk/ >
(Sjekket 30.mai 2000)

[OpenJournal98]
“Open Journal Project: final report to eLib”
< http://journals.ecs.soton.ac.uk/yr3/3rd-year-open.htm >
(Sjekket 20.mai 2000)

[Risher]
Helen Atkins, Catherine Lyons, Howard Ratner, Carol Risher, Chris Shillum, David Sidman, Andrew Stevens, "Reference Linking with DOI’s : A case study", *D-lib Magazine Volume 6 Number 2*, February 2000
< http://www.dlib.org/dlib/february00/02risher.html >
(Sjekket 30.mai 2000)

[VanDeSompel1]
Herbert Van de Sompel and Patrick Hochstenbach, "Reference Linking in a Hybrid Library Environment Part 1: Frameworks for Linking" *D-Lib Magazine Volume 5 Issue 4*, April 1999
< http://www.dlib.org/dlib/april99/van_de_sompel/04van_de_sompel-pt1.html >
(Sjekket 30.mai 2000)

[VanDeSompel2]
Herbert Van de Sompel and Patrick Hochstenbach, "Reference Linking in a Hybrid Library Environment Part 2: SFX, a Generic Linking Solution", *D-lib Magazine Volume 5 Issue 4*, April 1999.
< http://www.dlib.org/dlib/april99/van_de_sompel/04van_de_sompel-pt2.html >
(Sjekket 30.mai 2000)

[VanDeSompel3]
Herbert Van de Sompel and Patrick Hochstenbach, "Reference Linking in a Hybrid Library Environment Part 3: Generalizing the SFX solution in the "SFX@Ghent & SFX@LANL" experiment", *D-Lib Magazine,Volume 5 Number 10*, October 1999
< http://www.dlib.org/dlib/october99/van_de_sompel/10van_de_sompel.html >
(Sjekket 30.mai 2000)

[XML.com1]
Bob DuCharm, "Links That Are More Valuable Than the Information They Link?"
< http://www.xml.com/pub/98/07/xlink/index.html >
(Sjekket 30.mai 2000)

[XMLMyths]
"Four myths about XML"
< http://metalab.unc.edu/pub/sun-info/xml/why/4myths.htm >
(Siden er fjernet)

[W3C Xlink]
“XML Linking Language (Xlink)”, *W3C* working draft 20-december-1999
< http://www.w3.org/TR/xlink/ >
(Sjekket 30.mai 2000)

[W3C Xpointer]
“XML Pointer Language (Xlink)”, W3C working draft 06-december-1999
< http://www.w3.org/TR/xptr >
(Sjekket 30.mai)

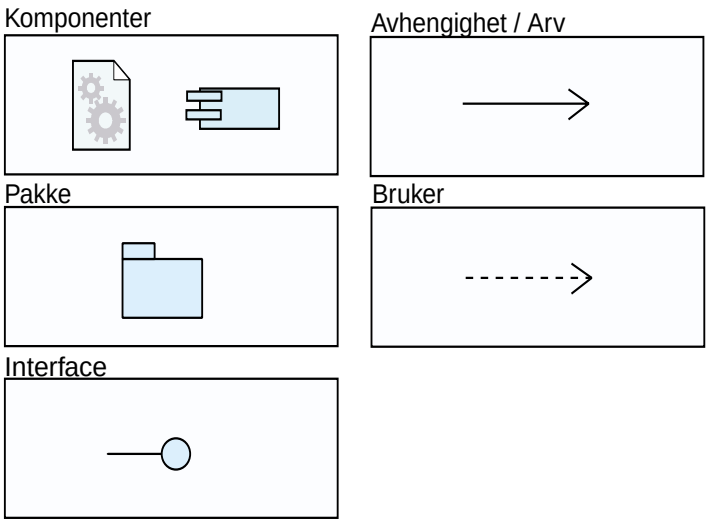
[W3C-Namespace]
"Namespaces in XML", World Wide Web Consortium 14-January-1999
< http://www.w3.org/TR/1999/REC-xml-names-19990114/ >
(Sjekket 30.mai)

[W3C-XML]
"Extensible Markup Language (XML) 1.0", *W3C* Recommendation 10-February-1998
<http://www.w3.org/TR/REC-xml >
(Sjekket 30.mai)

Appendix

1 UML symboler brukt i modeller.

UML symboler brukt i hovedoppgaven.



2 Komponenter i systemet

2.1 *Komponenter i implementasjon.*

Tabell 1: “Komersielle” Komponenter

Database	Versant ODBMS
Java Interface mot database	JVI for Versant
XMLParser	XML4J/Xerces
Utviklingsverktøy	JBuilder3.5 Enterprise TextPad 4.1 JDK 1.2.2
Webserver	Apache
Java Servlet	Apache JServ

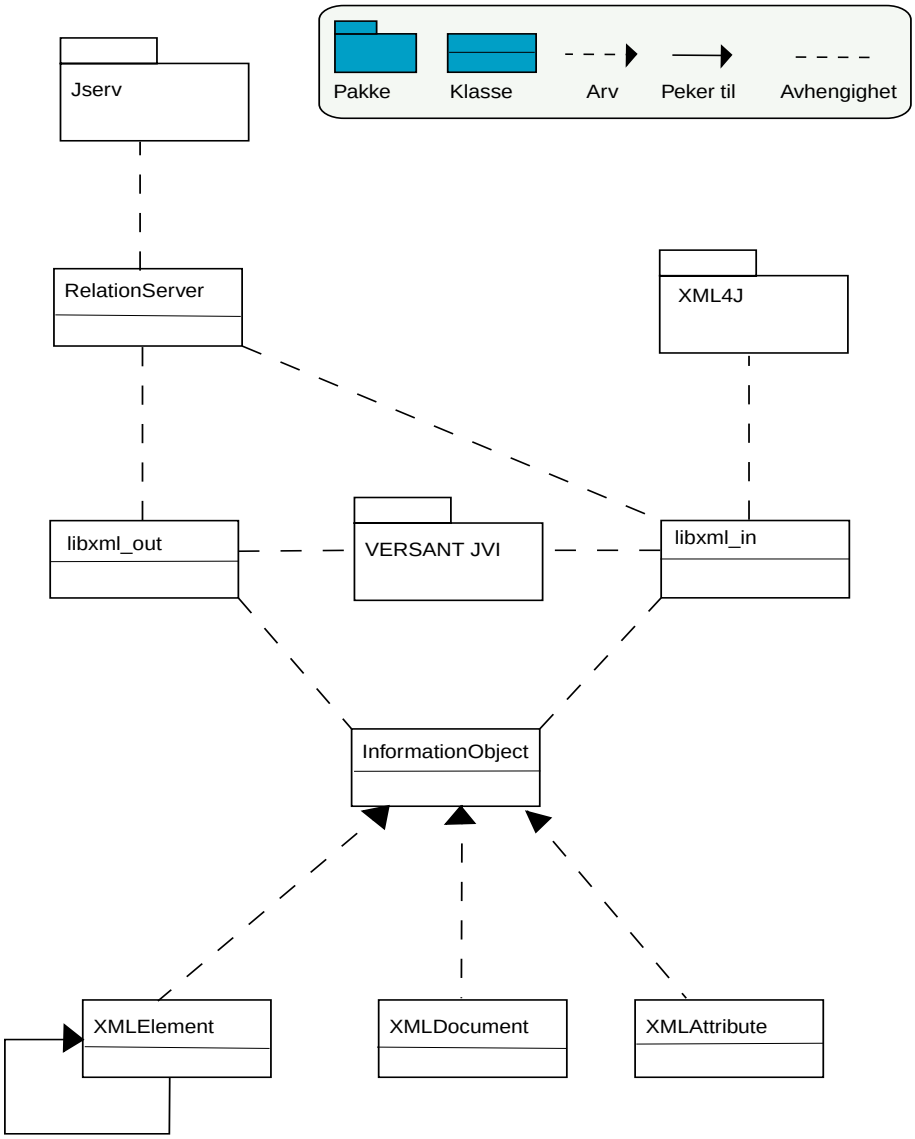
2.2 *Komponenter brukt i DIGLIB*

Tabell 2: DIGLIB Komponenter

Samlinger	Metadat, Bilder (FW, FW2) Metadata (Trobib)
Felles server grensesnitt	Standarisert grensesnitt for innlegging og uthenting av samling. Egenutviklet javabibliotek for databasetransaksjon basert på XML: xmllib

3 Klassehierarki for relasjons-systemet.

Relasjons-Server Klasser



4 Systembeskrivelse

4.1 *Bruk av klassene for lagring - libxml*

Java-biblioteket libxml er ikke pakket som jar-fil, men er en mappe med navn libxml. Dette er gjort for å ha bedre tilgang på koden.

Oppsett av CLASSPATH

Alle klasser som deltar i lagringssystemet er samlet under java-biblioteket `libxml`. På serveren for DigLib (fenris) ligger disse under `/usr/flocal/java/classes/`. For å kunne bruke dette må biblioteket ligge i `CLASSPATH` variabelen. Ved bruk av mappen gjengitt over blir dette:

```
export CLASSPATH=$CLASSPATH;/usr/flocal/java/classes/
```

Det er viktig at man ikke inkluderer navnet til mappen `libxml` i `classpath` variabelen.

Bruk av biblioteket i egne Java-program

Når man har satt opp `CLASSPATH` for `libxml` kan man inkludere dette i sitt eget Java program ved å bruke følgende deklarasjon:

```
import libxml.*;
```

Man vil da kunne ta i bruk klassene gjennom å instansiere dem som objekt gjennom bruk av `new`.

```
libxml.libxml_in minInstans = new libxml.libxml_in();
```

4.2 *Bruk av klassene for relasjoner - libRelation*

Java-biblioteket libxml er ikke pakket som jar-fil, men er en mappe med navn libxml. Dette er gjort for å ha bedre tilgang på koden.

Oppsett av CLASSPATH

For å bruke klassene i `libRelation` må man angi `CLASSPATH` til biblioteket. Denne er på fenris lagt under `/usr/flocal/java/classes/` slik at `CLASSPATH` angitt i 4.1 også inkluderer dette biblioteket.
`!` `libRelation` bruker `libxml`, og det er derfor viktig at `libxml` ligger i `CLASS-`

```
PATH !
```

Bruk av biblioteket for relasjoner i egne program

For å bruke libRelation i egne java-programmer må man importere biblioteket:

```
import libRelation.*;
```

Oppretting av nye instanser av klasser i biblioteket gjøres slik:

```
libRelation.relationStore minInstans = new libRelation.relati-
onStore(“minDatabase”);
```

5 Systembeskrivelse for *libxml*

5.1 *Innhold av libxml.*

```
Package : libxml

Classes : libxml_in
          libxml_out
          InsertXML
          XMLDocument
          XMLElement
          XMLAttribute
          BinaryDocument
```

5.2 *libxml_in*

Klasse for lagring av XML i Versant ODBMS. Klassen har en *constructor,* og to *public methods* for oppretting av objekter.

Constructors

libxml_in();
Oppretter et nytt libxml_in objekt

Public Methods

storeDocument - funksjon som tar imot et XML-dokument og et databasenavn og lagrer XML-dokumentet i objektorientert struktur i spesifisert database.

Input: Innhold i XML (String), Databasenavn (String)

Returns: void

Funksjonskall:
```
void storeDocument(String Content, String Database)
```

storeBinary - funksjon som lagrer et binærobjekt basert på en byte[] variabel.

Input: Mimetype (String), Navn (String), Beskrivelse (String), Databasenavn (String)
Returns: void

Funksjonskall:
```
StoreBinary(String bMime, String bName, String bDesc,
byte[] bytes, String Database)
```

storeBinary_FromFile - funksjon som lagrer binærobjekt basert på angivelse av fil på disk.

Input: Filnavn med sti (String), Databasenavn (String)
Returns: 1 hvis suksess, -1 hvis feil

Funksjonskall:
```
StoreBinary_FromFile(String bFileName, String Database)
```

5.3 *libxml_out*

Klasse for uthenting av lagret informasjon. Klassen har en *constructor* og seks public methods.

Constructors

Funksjonskall: libxml_out();

Public Methods

get_DocumentsByElementContent - Henter en vector som inneholder alle objekt-tid'er til *XMLDocument*-objekter som passer det elementnavnet som det er søkt på.

Input: Elementnavn (String), Trunkerings-angivelse (true/false), Databasenavn (String)
Returns: vector (String)

Funksjonskall:
```
get_DocumentsByElementContent(String
```

```
parameter, boolean TRUNC, String DATABASE)
```

get_DocumentsByAttributeContent - Henter en vector som inneholder alle objektid'er til *XMLDocument*-objekter som passer det attributtnavnet som det er søkt på.

Input: Attributtnavn (String), Trunkerings-angivelse (true/false), Databasenavn (String)
Returns: vector (String)

Funksjonskall:
```
get_DocumentsByAttributeContent(String parameter,
boolean TRUNC, String DATABASE)
```

get_AttributeValue - Henter ut verdien av et spesifisert attributtnavn i et spesifisert elementnavn i et spesifisert *XMLDocument*-objekt.

Input: XMLDocument-objektId (String), Elementnavn (String), Attributtnavn (String), Databasenavn (String)
Returns: Attributtverdi (String)

Funksjonskall:
```
get_AttributeValue(String liod, String ElementName,
String AttributeName, String DATABASE)
```

get_ElementValue - Henter ut verdien av et spesifisert elementnavn i et spesifisert *XMLDocument*-objekt.

Input: XMLDocument-objektId (String), Elementnavn (String), Databasenavn (String)
Returns: Elementverdi (String)

Funksjonskall:
```
get_ElementValue(String liod, String ElementName,
String DATABASE)
```

get_SpecifiedDocument - Henter ut et spesifisert dokument basert på ojektid. Hvis objektetid'en ikke tilsvarer et *XMLDocument* blir det tilhørende *XMLDocument* funnet og et XML-dokument blir returnert. Hvis objektid'en tilsvarer et Binary-Document blir innholdet av dette returnert.

Input: XMLDocument-objektId (String), Databasenavn (String)
Returns: XMLdokument eller binærverdi (Object)

Funksjonskall:
```
get_SpecifiedDocument(String liod, String DATABASE)
```

6 Systembeskrivelse for libRelation

6.1 *Innhold av libRelation.*

```
Package : libRelation

Classes : relationStore
          relationRetrive
```

6.2 *relationStore*

Klasse for lagring av relasjoner.

Constructors

relationStore - Oppretter nytt objekt. To valg: Ingen input, eller input av Databasenavn som skal brukes.

Funksjonskall:

```
relationStore();
relationStore(String db);
```

Public Methods

set_DatabaseName - Setter nytt databasenavn som det skal jobbes mot.

Input: Databasenavn (String)
Returns: void

Funksjonskall:

```
set_DatabaseName(String db);
```

get_DatabaseName - Henter databasenavnet som brukes.

Input:
Returns: Databasenavn (String)

Funksjonskall:

```
String get_DatabaseName();
```

add_Relation - Oppretter en relasjon utifra angivelse av de verdier som inngår i beskrivelsen i den database som er angitt i relationStore objektet.

Input: KildeAdresse (String), DestinasjonsAdresse (String),

Retningsangivelse (String), Relasjonstype (String)
Returns: void

Funksjonskall:
```
add_Relation(String Source, String Destination,
String Traverse, String RType);
```

add_Relation - Oppretter en relasjon utifra en ferdig beskrivels av relasjonen i XML, i den database som er angitt i relationStore objektet.

Input: XMLdokument med beskrivelse av relasjon (String)
Returns: void

Funksjonskall:
```
add_Relation(String xmlString);
```

6.3 *relationRetrive*

Klasse for gjenhenting av relasjoner

Constructors

relationRetrive - Oppretter nytt objekt. To valg: Ingen input, eller input av Databasenavn som skal brukes.

Funksjonskall:
```
relationRetrive();
relationRetrive(String db);
```

Public Methods

set_DatabaseName - Setter nytt databasenavn som det skal jobbes mot.

Input: Databasenavn (String)
Returns: void

Funksjonskall:
```
set_DatabaseName(String db);
```

get_DatabaseName - Henter databasenavnet som brukes.

Input:
Returns: Databasenavn (String)

Funksjonskall:
```
String get_DatabaseName();
```

get_RelationsList - Henter ut en liste i XML med alle de relasjoner som passer et søk på identifikator.

Input: Identifikator som skal søkes på (String)
Returns: Vector med alle id til *XMLDocument*-objekt som inneholder relasjonene som tilsvarte søket (String)

Funksjonskall:
```
get_RelationsList(String query)
```

get_InfoVector - Spesialfunksjon for uthenting av relasjonselementer uten bruk av XML. Funksjonen lager en matrix bestående av to vektorer. I hvert element i den første vektoren legges en ny vektor som består av relasjonselementene i følgende rekkefølge : *Source*, *Destination*, *Traverse*, *Type*.

Input: Identifikator som skal søkes på (String)
Returns: vektor med vektorer som inneholder relasjonselementene til de gjenfunnede relasjoner (Vector/Vector/String)

Funksjonskall:
```
get_InfoVector(String DocID)
```

get_Relation - Henter ut en relasjon utifra id til et *XMLDocument*-object

Input: Id for *XMLDocument*-objekt (String)
Returns: XMLdokument (String)

Funksjonskall:
```
String get_Relation(String docId)
```

7 Distribuert system.

7.1 *Protokoll for tuppler i TSpace.*

Tabell 3: Klient protokoll

Navn	Funksjon	Beskrivelse
request	add	Legger til en ny relasjon
request	get	Henter ut relasjon som koresponderer med en gitt handle
client		Angir navn på innloget klient

Tabell 4: Tuppler skrevet av klient

Tuppler	Forklaring
(“request”, CLIENTNAME, “add”, CONTENT)	CONTENT = XML “kodet” relasjon CLIENTNAME = Navn på klienten
(“request”, CLIENTNAME, “add”, CONTENT)	CONTENT = Handle som skal slås opp CLIENTNAME = Navn på klienten
(“relation_client”, CLIENTNAME)	CLIENTNAME = Navn på klienten STATE = Klientens tilstand (“ready” / “busy”

Tabell 5: Server Protokoll

Navn	Funksjon	Beskrivelse
request_completed		Sender ut melding til klient om “oppdrag utført”
XML_Document		Tuppel som inneholder

Tabell 6: Tuppler skrevet av tjener

Tuppler	Forklaring
(“request_completed”, MESSAGE)	Skrives tilbake til klientens rom ! Med beskjed om hva som er utført.
(“xml_relation”, XML)	Skrives tilbake til klientens rom ! Med en Relasjon i form av XML
(“relation_server”, SERVERNAME)	Angir servernavn (Hvis det finnes felere servere som kan betjene klienter.

8 Kildekode.

8.1 *Lagringsmodell - InformationObject*

```
package libxml;
/***********************************************
*   RelationObject definition
*       Conforms to the informationObject standard
*   Creator: Einar Lichtenberg
*/
public class InformationObject {

        // Conformation with Information Object
        private java.lang.String Mime;// Mimetype of content
        private java.lang.String Content; // Contnent of object

        // Constructors
        public InformationObject() {
        }

        // Public methods
        public java.lang.String get_Content() {
                return this.Content;
        }
        public void set_Content(java.lang.String co) {
                this.Content = co;
        }
        public java.lang.String get_Mime() {
                return this.Mime;
        }
        public void set_Mime(java.lang.String mi) {
                this.Mime = mi;
        }
}
```

8.2 *Lagringsmodell - XMLDocument*

```
package libxml;
/*****************************************
* XMLDocumentObject definition
* Inherits from informationObject
* Creator : Einar Lichtenberg (C) 2000
*/
public class XMLDocument extends libxml.InformationObject {
        private XMLElement Root; //Root Element

        // Constructors
        public XMLDocument() {}

        // Get entire Vector of elements
        public libxml.XMLElement get_Root() { return this.Root; }
        // Set entire Vector of elements
        public void set_Root(libxml.XMLElement ro) { this.Root = ro; }

        //Get entire document as element
        public String get_DocumentAsXML() { return get_Root().get_AsXML(); }
}
```

8.3 *Lagringsmodell - XMLElement.*

```
package libxml;
/************************************************
* XMLElement definition
* Inherits from informationObject
* Creator: Einar Lichtenberg (C) 2000
*/
import java.util.*;
import com.versant.util.*;

public class XMLElement extends libxml.InformationObject {

        // Conformation with Information Object
        private java.lang.String Name;
        private libxml.XMLDocument ParentDocument;
        private VVector Elements;
        private VVector Attributes;

        // Constructor
        public XMLElement() {
               Elements   = new VVector();
               Attributes = new VVector();
        }

        // Get Name of Element
        public java.lang.String get_Name() { return this.Name; }
        // Set Name of Element
        public void set_Name(java.lang.String nam) { this.Name = nam; }
        // Get ParentDocument of Element
        public libxml.XMLDocument get_ParentDocument()
               { return this.ParentDocument; }
        // Set ParentDocument of Element
        public void set_ParentDocument(libxml.XMLDocument doc)
               { this.ParentDocument = doc; }
        // Get entire Vector of elements
        public VVector get_Elements() { return this.Elements; }
        // Set entire Vector of elements
        public void set_Elements(VVector elm) { this.Elements = elm; }
        // new XMLElement et the end of the vector
        public void add_Element(libxml.XMLElement elm)
               { this.Elements.addElement(elm);}
        // Get entire Vector of elements
        public VVector get_Attributes() { return this.Attributes; }
        // Set entire Vector of elements
        public void set_Attributes(VVector att) { this.Attributes = att; }
        // new XMLAttribute at the end of the vector
        public void add_Attribute(libxml.XMLAttribute attr)
               { this.Attributes.addElement(attr);}

        //get_AsXML returns entire substructure of this element
        //including attributes
        public String get_AsXML() {
               String xmlstring = "<" + this.get_Name();
               if ( !( this.get_Attributes() == null) ) {
                   VVector attrV = this.get_Attributes();
                   Enumeration e = attrV.elements();
            while ( e.hasMoreElements() ) {
                      libxml.XMLAttribute XA =
                       (libxml.XMLAttribute)e.nextElement();
                      xmlstring = xmlstring + " " + XA.get_Name() + "=\"" +
                      XA.get_Content() +"\"";
                   }
```

```
        }
                xmlstring = xmlstring + ">";
                if ( !( this.get_Content() == null) )
                    xmlstring = xmlstring + this.get_Content();
                if ( !( this.get_Elements() == null) ) {
                    VVector elmV = this.get_Elements();
                    Enumeration e = elmV.elements();
                    while ( e.hasMoreElements() ) {
                        libxml.XMLElement elm =
                        (libxml.XMLElement)e.nextElement();
                        xmlstring = xmlstring + elm.get_AsXML();
                    }
                }
                xmlstring = xmlstring + "</" + this.get_Name() + ">";
                return xmlstring;
        } /*end get_AsXML*/

} /*END_CLASS*/
```

8.4 *Lagringsmodell - XMLAttribute.*

```
package libxml;
/****************************************
* XMLAttribute definition
* Inherits from informationObject
* Creator : Einar Lichtenberg (C) 2000
*/
public class XMLAttribute extends libxml.InformationObject {

        private java.lang.String Name;// Name of attribute
        private libxml.XMLElement ParentElement; // XMLElement parent
        // Constructors
        public XMLAttribute() {
        }

        // Get name
        public java.lang.String get_Name() { return this.Name; }
        // Set name
        public void set_Name(java.lang.String nam) { this.Name = nam; }
        // Get parent
        public libxml.XMLElement get_ParentElement()
                { return this.ParentElement; }
        // Set parent
        public void set_ParentElement(libxml.XMLElement elm)
                { this.ParentElement = elm; }
}
```

8.5 *Lagringsmodell - BinaryDocument.*

```
package libxml;
/** BinaryDocument definition *********************************
*       Inherits from informationObject
*       Creator: Einar Lichtenberg (C) 2000
*       Uses Content as general description field
*/
public class BinaryDocument extends libxml.InformationObject {

        private java.lang.String Name;// Mimetype of content
        private byte[]   BinaryContent; //Binary content of object
```

```
        // Constructors
        public BinaryDocument() {
                this.BinaryContent = null;
        }
        //Get Name
        public java.lang.String get_Name() { return this.Name; }
        //Set Name
        public void set_Name(java.lang.String nam) { this.Name = nam; }
        //Set the binary content
        public void set_BinaryContent(byte[] bfi) { this.BinaryContent = bfi; }
        //get the binary content
        public byte[] get_BinaryContent() { return this.BinaryContent; }
}
```

8.6 *Lagringsmodell - libxml_in*

```
package libxml;
/*************************************************************
* libxml.libxml_in
* Mapps between Versant's Database and XML
* Creator : Einar Lichtenberg
*/

//XML-Parser imoports
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.*;
import org.w3c.dom.traversal.*;
import org.xml.sax.*;
//Java imoports
import java.lang.*;
import java.util.*;
import java.io.*;
import java.io.IOException;

//Versant Database imports
import com.versant.trans.*;
import com.versant.util.*;

/*****************************************************************
* CLASS libxml_in
*/
public class libxml_in {

        /*************************************************************
        * GLOBAL VARIABLES
        */
        protected static Document DOC = null; // Stores the DOM-tree
        protected static libxml.XMLDocument XD = null; // This is
                              //for reference inn parsing
        protected static TransSession SESSION; // For versant

        /*************************************************************
        * PROCEDURE PARSEXMLString
        */
        private static void parseXMLString(String parseData) {
                DOMParser parser = new DOMParser();
                InputSource is = new InputSource(new StringReader(parseData) );
                //File xmlfile = parseData;
                try {
                    parser.parse(is);
```

```
        } catch (SAXException se) {
            se.printStackTrace();
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
        DOC = parser.getDocument();
        DOC.getDocumentElement().normalize();
}

/*****************************************************************
* PROCEDURE parseDocument;
*/
public static void storeDocument(String Content, String Database)
{
        XMLElement XEP;
        SESSION  = new TransSession (Database);
        parseXMLString(Content); //Parse the xml string into DOC
        Element ROOT  = (Element)DOC.getDocumentElement();
        NodeList nodelist = ROOT.getChildNodes();

        //New XMLdocument and ROOT Element
        XD = new XMLDocument();
        XD.set_Mime("text/xml");
        XEP = new XMLElement();
        XEP.set_Mime("text/xml");
        XEP.set_Name(ROOT.getTagName() );
        XEP.set_ParentDocument(XD);
        XD.set_Root(XEP);

        //First one Has attributes ?
        NamedNodeMap attributemap = ROOT.getAttributes();
        if ( !(attributemap.item(0) == null) ) {StoreAttributes(XEP,attributemap);}
    else {XEP.set_Attributes(null);}

        //First on has children ?
        if (! (nodelist==null) ) StoreRecursive(XEP,nodelist); //Now store it !
        XD.set_Content(null); //document has no content
        SESSION.makePersistent (XEP); //Make object persistent
        SESSION.makePersistent (XD);
        SESSION.commitAndCleanCod(); //Commit transaction;
        SESSION.endSession ();
        System.out.println("Stored object....");

}/*END_FUNCTION*/


/*********************************************************************
* Recursive stor function to get elements nested in document
*/
private static void StoreRecursive(XMLElement XElm, NodeList nList) {
        String aName = "";
        NodeList rList;
        XMLElement XE;
        Element currElm;
        Node currNode;

        for (int i = 0 ; i < nList.getLength() ; i ++ ) {

            /*IF ELEMENT */
            if (nList.item(i).getNodeType() == Node.ELEMENT_NODE) {
                //If the Node is an element
                currElm = (Element)nList.item(i);
```

```
                XE = new XMLElement();//make new XMLElement
                XE.set_Mime("text/xml");
                XE.set_Name(currElm.getTagName() );
                XE.set_ParentDocument(XD);

                NamedNodeMap nnl = currElm.getAttributes();
                if ( !(nnl.item(0) == null) ) {StoreAttributes(XE,nnl);}
                else {XE.set_Attributes(null);}
                NodeList elmList = currElm.getChildNodes();
                    //check children of current element
                if ( !(elmList == null) ) {StoreRecursive(XE, elmList); }
                else {XE.set_Elements(null);}
                XElm.add_Element(XE);
                   //add this Element to child list of parent element
                SESSION.makePersistent (XE);
            }/*END_IF*/

            /*ELSE_IF TEXT*/
            else
            if (nList.item(i).getNodeType() == Node.TEXT_NODE) {
                       //if not element then store the text in element
                currNode = nList.item(i);
                String aValue = currNode.getNodeValue().trim();
                if ( !( aValue.equals("") || aValue.equals("\r") ) ){
                    aName = aName + aValue;
                    XElm.set_Content(aName);
                }
            }/*END_ELSE_IF*/

            /*ELSE OTHER TYPE (ARE THER OTHER TYPES ?)*/
            else {
                System.out.println("Found something strange !");
            }/*END_ELSE*/

        } /*END_FOR*/
        XElm.get_Elements().trimToSize();
        if (XElm.get_Elements().isEmpty() ) XElm.set_Elements(null);
}/*END_FUNCTION*/

/*****************************************************************
* StoreAttributes
*/
private static void StoreAttributes(XMLElement XElm, NamedNodeMap nList) {
        XMLAttribute XA;
        for (int i = 0; i < nList.getLength(); i++) {
            XA = new XMLAttribute();
            XA.set_Name(nList.item(i).getNodeName() );
            XA.set_Content(nList.item(i).getNodeValue() );
            XA.set_Mime("text/xml");
            XA.set_ParentElement(XElm);
            XElm.add_Attribute(XA);
            SESSION.makePersistent(XA);
        }
        XElm.get_Attributes().trimToSize();
}/*END_FUNCTION*/

/*****************************************************************
* StoreBinary
*/
public static void StoreBinary(
        String bMime, String bName, String bDesc, byte[] bytes, String Database) {

        SESSION  = new TransSession (Database);
        BinaryDocument BD = new BinaryDocument();
```

```
                BD.set_Mime(bMime);
                BD.set_BinaryContent(bytes);
                BD.set_Name(bName);
                BD.set_Content(bDesc);
                SESSION.makePersistent(BD);
                SESSION.commit();
                SESSION.endSession();
        }

        /*****************************************************************
        * StoreBinary_FromFile
        */
        public static int StoreBinary_FromFile(String bFileName, String Database) {
                byte[] bytes = null;
                File aFile = new File(bFileName);
                try {
                    FileInputStream FS = new FileInputStream(aFile);
                    bytes = new byte[(int)aFile.length()];
                    if (FS.read(bytes) == -1) {return -1;}
                } catch (IOException ioe) {return -1; }

                    SESSION  = new TransSession (Database);
                    BinaryDocument BD = new BinaryDocument();
                    BD.set_Mime("n/a");
                    BD.set_BinaryContent(bytes);
                    BD.set_Name(aFile.getName());
                    BD.set_Content("n/a");
                    SESSION.makePersistent(BD);
                    SESSION.commit();
                    SESSION.endSession();
                    return 1;
        }
/*************************************************************************/
} // END CLASS
```

8.7 *Lagringsmodell - libxml_out.*

```
package libxml;
/************************************************************
* Class    : libxml.libxml_out
* Function : Retrives information stored in database through
*                  libxml
* Creator  : Einar W. M. Lichtenberg
*/

//XML-Parser Imports
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.*;
import org.w3c.dom.traversal.*;
import org.xml.sax.*;

//Java dependent Imports
import java.lang.*;
import java.util.*;
import java.io.*;
import java.io.IOException;

//Versant Database Imports
import com.versant.trans.*;
import com.versant.util.*;
import com.versant.fund.*;

/************************************************************
* CLASS libxml_out
*/
public class libxml_out {

//GLOBAL VARIABLES ************************************
protected static libxml.XMLDocument XD = null;
// This is for reference inn parsing
protected static libxml.XMLElement  XE = null;
//protected static TransSession SESSION;

/************************************************************
* get_DocumentsByElementContent
* Input : Element name, Return: List of objectid's
*/
public static Vector get_DocumentsByElementContent
        (String parameter, boolean TRUNC, String DATABASE) {

        Vector retV = new Vector();
        VQLQuery query = null;
        //Start database session
        TransSession SESSION  = new TransSession (DATABASE);
        if (!(TRUNC) ) {
                //If no truncation
                query =new VQLQuery
                (SESSION, "select selfoid from libxml.XMLElement where Content = $1");
                query.bind(parameter);
        } else {
                //if truncation
                query = new VQLQuery
                (SESSION, "select selfoid from libxml.XMLElement where Content like $1");
                query.bind(parameter+"*");
        }
        Enumeration e = query.execute ();
        if ( !e.hasMoreElements() ) {
                SESSION.endSession();
```

```
            return null;
        } else {
                while ( e.hasMoreElements() ) {
                    libxml.XMLElement elm = (libxml.XMLElement)e.nextElement();
                    //get XMLDocument for XMLElement
                    libxml.XMLDocument doc = elm.get_ParentDocument();
                    // Add oid to Vvector as string
                    retV.addElement(SESSION.getOidAsString(doc) );
                }
        }
        SESSION.endSession ();
        retV.trimToSize(); // Trim vector to number of real elements
        if (retV.isEmpty()) retV = null; // If empty after trim return null
        return retV;
}

/************************************************************
* get_DocumentsbyAttriuteContent
* Input: Attributename Return: Vector with XMLDocument iod's
*/
public static Vector get_DocumentsByAttributeContent
        (String parameter, boolean TRUNC, String DATABASE) {

        Vector retV = new Vector();
        VQLQuery query = null;
        TransSession SESSION  = new TransSession (DATABASE);
        if (!(TRUNC) ) {
                //If no truncation
                query =new VQLQuery
                (SESSION, "select selfoid from libxml.XMLAttribute where Content = $1");
                query.bind(parameter);
        } else {
                //if truncation
                query = new VQLQuery
                (SESSION, "select selfoid from libxml.XMLAttribute where Content like $1");
                query.bind(parameter+"*");
        }
        Enumeration e = query.execute ();
        if ( !e.hasMoreElements() ) {
            SESSION.endSession();
            return null;
        } else {
                while ( e.hasMoreElements() ) {
                    libxml.XMLAttribute atr = (libxml.XMLAttribute)e.nextElement();
                    //get XMLDocument for XMLAttribute
                    libxml.XMLDocument doc = atr.get_ParentElement().get_ParentDocument();
                    // Add OID to Vvector as string
                    retV.addElement(SESSION.getOidAsString(doc) );
                }
        }
        SESSION.endSession ();
        retV.trimToSize(); // Trim vector to number of real elements
        if (retV.isEmpty()) retV = null; // If empty after trim return null
        return retV;
}
/************************************************************
* get_AttributeValue : returns a value of an attribute
*                           spesified thorugh a loid
* ! Only returns first Elements Attribute
*/
public String get_AttributeValue
        (String liod, String ElementName, String AttributeName, String DATABASE) {
        System.out.println("Inside get_AttributeValue....");
        System.out.println("DATABASE = "+ DATABASE);
```

```
        System.out.println("ElementName = "+ElementName);
        System.out.println("AttributeName = "+AttributeName);
        System.out.println("liod = "+ liod);

        TransSession SESSION = new TransSession (DATABASE);
        XMLDocument  doc  = null;
        XMLElement   elm  = null;
        boolean  terminate = false;
        Handle docHand = SESSION.newHandle(liod);
        ClassHandle classH = docHand.classObjectOf();
        if (classH.classname().equals("libxml.XMLDocument") ) {
                doc = (XMLDocument)docHand.handleToObject();
                VQLQuery query = null;
                query =new VQLQuery
                    (SESSION, "select selfoid from libxml.XMLElement where
                                ParentDocument = $1 and Name = $2");
                query.bind(docHand.handleToObject() );
                query.bind(ElementName);
                Enumeration e = query.execute ();
                if ( !e.hasMoreElements() ) {
                    SESSION.endSession();
                    return null;
                } else {
                    elm = (libxml.XMLElement)e.nextElement();
                    VVector atts = elm.get_Attributes();
                    Enumeration en = atts.elements();
                    if ( !en.hasMoreElements() ) {
                        SESSION.endSession();
                        return null;
                    } else {
                        while ( en.hasMoreElements() ) {
                            libxml.XMLAttribute att =
                            (libxml.XMLAttribute)en.nextElement();
                            if (att.get_Name().equals(AttributeName) ) {
                                String attValue = att.get_Content();
                                SESSION.endSession();
                                return attValue;
                            }
                        }
                        SESSION.endSession();
                            return null;
                    }
                }
        } else return null;
}
/*************************************************************
* get_ElementValue : returns a value of an element
*                          spesified thorugh a loid
* ! Only returns first elements Value
*/
public String get_ElementValue(String liod, String ElementName, String DATABASE) {
        TransSession SESSION = new TransSession (DATABASE);
        XMLDocument  doc  = null;
        XMLElement   elm  = null;
        boolean  terminate = false;
        Handle docHand = SESSION.newHandle(liod);
        ClassHandle classH = docHand.classObjectOf();
        if (classH.classname().equals("libxml.XMLDocument") ) {
                doc = (XMLDocument)docHand.handleToObject();
                VQLQuery query = null;
                query =new VQLQuery
                    (SESSION, "select selfoid from libxml.XMLElement where ParentDocument =
$1 and Name = $2");
                query.bind(doc);
```

```
                query.bind(ElementName);
                Enumeration e = query.execute ();
                if ( !e.hasMoreElements() ) {
                    SESSION.endSession();
                    return null;
                } else {
                    elm = (libxml.XMLElement)e.nextElement();
                    String elmValue = elm.get_Content();
                    SESSION.endSession();
                    return elmValue;
                }
        } else {
                SESSION.endSession();
                return null;
        }
}
/*************************************************************
* get_SpecifiedDocument
* Input : liod, Return: ojbject
*/
public static Object get_SpecifiedDocument(String liod, String DATABASE) {
        TransSession SESSION = new TransSession (DATABASE);
        XMLDocument doc = null;
        Handle docHand = SESSION.newHandle(liod);
        ClassHandle classH = docHand.classObjectOf();
        if (classH.classname().equals("libxml.XMLDocument") )
                doc = (XMLDocument)docHand.handleToObject();
        else
        if (classH.classname().equals("libxml.XMLElement") ) {
                XMLElement elm  = (XMLElement)docHand.handleToObject();
                doc = elm.get_ParentDocument();
        }
        else
        if (classH.classname().equals("libxml.XMLAttribute") ) {
                    XMLAttribute att = (XMLAttribute)docHand.handleToObject();
                    doc = att.get_ParentElement().get_ParentDocument();
        }
        else
        if (classH.classname().equals("libxml.BinaryDocument") ) {
                    Object O = docHand.handleToObject();
                    BinaryDocument BD = (BinaryDocument)O;
                    SESSION.endSession();
                    return BD;
        }
        else {
                SESSION.endSession();
                return null; // if unknown class
        }
        String xmlString = doc.get_DocumentAsXML();
        SESSION.endSession();
        return xmlString;

} /*END_GETSPECIFIED*/
} // END CLASS ************************************************
```

8.8 *Lagringsmodell - InsertXML*

```
package libxml;
```

```
/*************************************************************
* CLASS InsertXML
*
* Interacts with Versant's Database
*/

import org.apache.xerces.parsers.DOMParser;
//import org.apache.xerces.dom.*;
import org.w3c.dom.*;
import org.w3c.dom.traversal.*;
import org.xml.sax.*;

import java.lang.*;
import java.util.*;
import java.io.*;
import java.io.IOException;

//Versant Database
import com.versant.trans.*;
import com.versant.util.*;

/*************************************************************
* CLASS InsertXML
*/
public class InsertXML {

        protected static String _DATABASE= "";
        protected static String _POSTNAME   = null;
        protected static Document _doc; // Stores the DOM-tree
        protected static Document _doc_fragment; // Stores a fragment
        protected static XMLDocument XD; // This is for reference inn parsing

        protected static boolean binaryAware = false;

        protected static TransSession session; //For versant

        /*************************************************************
        * PROCEDURE XMLPARSER
        */
        public static void LoadXMLFile(String parseData) {
                DOMParser parser = new DOMParser();
                InputSource is = new InputSource(parseData);
                //File xmlfile = parseData;
            try {
                    parser.parse(is);

                } catch (SAXException se) {
                    se.printStackTrace();
                } catch (IOException ioe) {
                    ioe.printStackTrace();
                }
                _doc = parser.getDocument();
                _doc.getDocumentElement().normalize();
                }

        /*************************************************************
        * PROCEDURE PARSEXMLString
        */
        public static void parseXMLString(String parseData) {
                DOMParser parser = new DOMParser();
                InputSource is = new InputSource(new StringReader(parseData) );
                //File xmlfile = parseData;
                try {
                    parser.parse(is);
```

```
        } catch (SAXException se) {
            se.printStackTrace();
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
        _doc_fragment = parser.getDocument();
        _doc_fragment.getDocumentElement().normalize();
}

/*****************************************************************
* PROCEDURE main;
*/
public static void main(String argv[])
{
    //InsertXML ix = new InsertXML();
    if ( (argv.length == 0) || (argv.length > 2) ) {
            System.out.println
                ("Usage : java libxml.InsertXML <file.xml> [binary]");
        } else {

        if (argv.length == 2) {
                binaryAware = true;
            }
        LoadXMLFile(argv[0]);

        NodeList nl = _doc.getElementsByTagName("collectionName");
        System.out.println ("Length of nodelist = " +
                      Integer.toString(nl.getLength()) );
            Node node = nl.item(0);
            _DATABASE = node.getFirstChild().getNodeValue();

            if (! (_DATABASE == null) ) {
            System.out.println ("Database name = " + _DATABASE);

            nl = _doc.getElementsByTagName("collectionUnits");
            NodeList childList = nl.item(0).getChildNodes();

                // the following parses through the whitespaces an /cr's
                for (int i = 0 ; i < childList.getLength() ; i ++ ){
                   if (childList.item(i).getNodeType() == Node.ELEMENT_NODE) {
                       _POSTNAME = childList.item(i).getNodeName();
                       break;
                   }
                }
            System.out.println ("Post name = " + _POSTNAME);

                //now Lets get goin with some storin'
                nl = _doc.getElementsByTagName(_POSTNAME);
                session  = new TransSession (_DATABASE);
                Element element;
                XMLElement XEP;
                NodeList nodelist;

                //*******************************************
                //Parse elements
                for (int i = 0; i < nl.getLength(); i++) {
                    //CHECK THE ELEMENT !
                    if (nl.item(i).getNodeType() == Node.ELEMENT_NODE) {
                        element = (Element)nl.item(i);

                        System.out.println("*** New post nr : " +
                        Integer.toString(i) + " ***" );
```

```
                            //New document with nodes
                            XD = new XMLDocument();
                            XD.set_Mime("text/xml");
                            XEP = new XMLElement();
                            XEP.set_Mime("text/xml");
                            XEP.set_Name(element.getNodeName() );
                            XEP.set_ParentDocument(XD);
                            XD.set_Root(XEP);

                            System.out.println("\nelement :" + element.toString() );
                            //parseXMLString(element.toString());

                            nodelist = element.getChildNodes();
                            //Do tha magic!
                            if (! (nodelist==null) ) StoreRecursive(XEP,nodelist);
                            //Now store it !
                            XD.set_Content(null);
                            XEP.set_Content(null);
                            //XE.set_Elements(null);
                            XEP.set_Attributes(null);
                            //and store them
                            session.makePersistent (XEP);
                            session.makePersistent (XD);

                            //Commit transaction;
                            session.commitAndCleanCod();
                            //session.releaseObject(XD);
                            //session.releaseObject(XEP);
                        }/*END_IF*/
                    }
                    session.endSession ();
                } else { System.out.println("Could not read Basename"); }
            }
        }

        /*****************************************************************
        * Recursive store
        */
        public static void StoreRecursive(XMLElement XElm, NodeList nList) {
                String aName = "";
                NodeList rList;
                XMLElement XE;
                Element currElm;
                Node currNode;
                //traverse nodelist
                //if (nList.item(i).getNodeType() == Node.TEXT_NODE) {

                System.out.println("Entered Store Recursive - should get Element or con-
tent...");
                for (int i = 0 ; i < nList.getLength() ; i ++ ) {

                    /*IF ELEMENT */
                    if (nList.item(i).getNodeType() == Node.ELEMENT_NODE) {
                                //If the Node is an element
                        currElm = (Element)nList.item(i);

                        XE = new XMLElement();//make new XMLElement
                        XE.set_Mime("text/xml");
                        XE.set_Name(currElm.getTagName() );
                        XE.set_ParentDocument(XD);
                        System.out.println("New Element Name : "+ XE.get_Name() +
                            " New Element Value: "+ currElm.getNodeValue() );

                        NamedNodeMap nnl = currElm.getAttributes();
```

```
                if ( !(nnl.item(0) == null) ) {StoreAttributes(XE,nnl);}
                else {XE.set_Attributes(null);}

                NodeList elmList = currElm.getChildNodes();
                //check children of current element
                // denne null sjekkingen funker ikke ! teste annen type
                // null sjekking !
                int length = elmList.getLength();

                if ( !(elmList == null) ) {StoreRecursive(XE, elmList); }
                else {XE.set_Elements(null);}
                System.out.println("elmList length : " + Integer.toString(length));
                System.out.println("elmList content: " + elmList.toString() );

                XElm.add_Element(XE);
                //add this Element to child list of parent element
                session.makePersistent (XE);
                //session.releaseObject(XE);
            }/*END_IF*/

            /*ELSE_IF TEXT*/
            else
            if (nList.item(i).getNodeType() == Node.TEXT_NODE) {
                //if not element then store the text in element
                currNode = nList.item(i);
                String aValue = currNode.getNodeValue().trim();
                if ( !( aValue.equals("") || aValue.equals("\r") ) ){
                    aName = aName + aValue;
                    XElm.set_Content(aName);
                }
                System.out.println("New Content Name : " +
                    nList.item(i).getNodeValue() + " New Content Value: " +
                    XElm.get_Content() );
            }/*END_ELSE_IF*/

            /*ELSE OTHER TYPE (ARE THER OTHER TYPES ?)*/
            else {
                System.out.println("Found something strange !");
            }/*END_ELSE*/

        } /*END_FOR*/
        XElm.get_Elements().trimToSize();
        if (XElm.get_Elements().isEmpty() ) XElm.set_Elements(null);
}/*END_FUNCTION*/

/*****************************************************************
* Store Attributes
*/
public static void StoreAttributes(XMLElement XElm, NamedNodeMap nList) {
        XMLAttribute XA;
        System.out.println
        ("Entered StoreAttributes - Should find some attributes...");
        for (int i = 0; i < nList.getLength(); i++) {
            XA = new XMLAttribute();
            XA.set_Name(nList.item(i).getNodeName() );
            XA.set_Content(nList.item(i).getNodeValue() );
            XA.set_Mime("text/xml");
            XA.set_ParentElement(XElm);
            System.out.println("New Attribute : " + XA.get_Name() +
                        " = " + XA.get_Content());
            XElm.add_Attribute(XA);
            if ( (XA.get_Name().equals("fileName")) && (binaryAware) ) {
                        int RET = storeBinary_FromFile(XA.get_Content());
                        if (RET == -1) {
```

```
                            System.out.println
                            ("An error occured during file Prosessing !");
                            }
                    }
                    session.makePersistent(XA);
                }
                XElm.get_Attributes().trimToSize();
        }/*END_FUNCTION*/

        /*********************************************************************
        * Store BinaryFile
        */
        private static int storeBinary_FromFile(String bFileName) {
                    System.out.println("Enterd STORE BINARY FILE !");
                    byte[] bytes = null;
                    File aFile = new File(bFileName);
                    try {
                        FileInputStream fileStream = new FileInputStream(aFile);
                        bytes = new byte[(int)aFile.length()];
                        int READB = fileStream.read(bytes);
                        System.out.println("READ BINARY BYTES : " +
                            Integer.toString(READB) );
                        if (READB == -1) {return -1;}
                    } catch (IOException ioe) {return -1;}
                    BinaryDocument BD = new BinaryDocument();
                    BD.set_Mime("n/a");
                    BD.set_BinaryContent(bytes);
                    BD.set_Name(aFile.getName());
                    System.out.println("STORED BINARY NAME : " + BD.get_Name());
                    BD.set_Content("n/a");
                    session.makePersistent(BD);
                    return 1;
        }
/*************************************************************************/
} // END CLASS
```

8.9 *Relasjonsmodell - relationStore*

```
package libRelation;
/*************************************************************
* relationStore
* Dependant : libxml
* Connects to libxml and stores relation
* Creator : Einar Lichtenberg
*/
//Import statements
import java.lang.*;
import java.util.*;

//libxml import
import libxml.*;

/*************************************************************
* CLASS relationStore
*/
public class relationStore {

/*************************************************************
* GLOBAL VARIABLES
*/
```

```
protected libxml_in  _LXIN;
protected libxml_out _LXOUT;

private String DATABASE = null; // Holds the current database.

static String _source= "Rel:Source";
static String _destination  = "Rel:Destination";
static String _href= "xlink:href";

static String _database= "LinkStore";

/*****************************************************************
* CONSTRUCTOR
*/
public relationStore() {
       this.DATABASE = this._database;
}

public relationStore(String db) {
       this.DATABASE = db;
}

/*****************************************************************
* Set databaseName
*/
public void set_DatabaseName(String db) { this.DATABASE = db; }

/*****************************************************************
* Get databasename
*/
public String get_DatabaseName() { return this.DATABASE; }

/*****************************************************************
* ADD A RELATION
*/
public void add_Relation(
       String Source, String Destination, String Traverse, String RType) {

       _LXIN   = new libxml_in();

       // Should use XML4J !!!
       String XMLstr = //"<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
              "<Rel:Relation xmlns:xlink=\"http://www.w3.org/1999/xlink/namespace/\" " +
              "xmlns:Rel=\"http://fenris.ifi.ntnu.no/namespaces/relation\""+
              "xlink:type=\"extended\" " +
              "xlink:role=\"Relation\" xlink:title=\"Relation description\">";

       String source = "<Rel:Source xlink:type=\"locator\" xlink:role=\"source\" " +
                       "xlink:href=\"" + Source + "\" xlink:title=\"na\"/>";

       String dest = "<Rel:Destination xlink:type=\"locator\" xlink:role=\"destination\"+
                     +  "xlink:href=\"" + Destination + "\" xlink:title=\"na\"/>";

       String travers= "<Rel:Traversal xlink:from=\"source\" xlink:to=\"destination\" " +
                       "Rel:traverse=\""+ Traverse + "\"/>";

       String type   = "<Rel:IsOfType>" + RType + "</Rel:IsOfType>";
       String comment= "<Rel:Additional>" + "</Rel:Additional>";

       String end    = "</Rel:Relation>";
    String xmlRelation = XMLstr + source + dest +  travers + type + comment + end;

       _LXIN.storeDocument(xmlRelation, DATABASE);
}
```

```
/*****************************************************************
* ADD RELATION 2 Take a finished xml string.
*/
public void add_Relation(String xmlString) {
        _LXIN.storeDocument(xmlString, DATABASE);
}

/*****************************************************************/
} // END OF CLASS
```

8.10 *Relasjonsmodell - relationRetrive*

```
package libRelation;
/*****************************************************************
* relationRetrive
* Dependant : libxml
* Connects to database via libxml for versant
* Retrives relations from identifier.
* Creator : Einar Lichtenberg
*/
//Java Import
import java.lang.*;
import java.util.*;

//libxml import
import libxml.*;

/*****************************************************************
* CLASS relationRetrive
*/
public class relationRetrive {

/*****************************************************************
* GLOBAL VARIABLES
*/
protected libxml_out _LXOUT;

private String DATABASE = null; // Holds the current database.

static String _source= "Rel:Source";
static String _destination  = "Rel:Destination";
static String _href= "xlink:href";

static String _database= "LinkStore";

/*****************************************************************
* CONSTRUCTOR
*/
public relationRetrive () {
        this.DATABASE = this._database;
        _LXOUT = new libxml_out();
}

public relationRetrive (String db) {
        this.DATABASE = db;
        _LXOUT = new libxml_out();
}

/*****************************************************************
* Set databaseName
```

```
*/
public void set_DatabaseName(String db) { this.DATABASE = db; }

/*****************************************************************
* Get databasename
*/
public String get_DatabaseName() { return this.DATABASE; }

/*****************************************************************
* Find Relations
*/
public Vector get_RelationsList(String query) {
       return _LXOUT.get_DocumentsByAttributeContent(query, false, DATABASE);
}

/*****************************************************************
* INFO VECTOR
*/
public Vector get_InfoVector(String DocID) {
       String Source =
        _LXOUT.get_AttributeValue(DocID, "Rel:Source", "xlink:href",DATABASE);
       String Destination =
        _LXOUT.get_AttributeValue(DocID, "Rel:Destination", "xlink:href",DATABASE);
       String Traverse=
        _LXOUT.get_AttributeValue(DocID, "Rel:Traversal", "Rel:traverse",DATABASE);
       String Type =
        _LXOUT.get_ElementValue(DocID, "Rel:IsOfType",DATABASE);

       Vector retVec = new Vector();
       retVec.addElement(Source);
       retVec.addElement(Destination);
       retVec.addElement(Traverse);
       retVec.addElement(Type);

       retVec.trimToSize();
       return retVec;
}
/*****************************************************************
* Find Relation
*/
public String get_Relation(String docId) {
       return (String)_LXOUT.get_SpecifiedDocument(docId, DATABASE);
}

/*****************************************************************/
} // END OF CLASS
```

8.11 *RelasjonsGrensesnitt (Servlet)- relAdd*

```
/*********************************************
* relAdd - Demonstration frontend for libRelation
* Creator: Einar Lichtenberg
*/
//Java Import
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
```

```
//libRelation Import
import libRelation.*;

/*********************************************
* Class relAdd
*/
public class relAdd extends HttpServlet {

        public void doPost (HttpServletRequest req, HttpServletResponse res)
                throws ServletException, IOException {

                res.setContentType("text/html");
                PrintWriter out = res.getWriter();

                out.println(printHead() );
                out.println("Adding....");

                libRelation.relationStore relStore;

                String parmName = "";
                String parmVal[]  = null;

                String Source = null;
                String Destination = null;
                String Traverse = null;
                String LinkType = null;


                Enumeration e = req.getParameterNames();
                 if ( !e.hasMoreElements() ) {
                           out.println ("No parameters were found.");
                 } else {
                    while ( e.hasMoreElements() ) {
                      parmName = (String)e.nextElement();
                       if ((parmName.toUpperCase()).equals("SOURCE") ) {
                           Source = req.getParameterValues(parmName)[0];
                       }
                       if ((parmName.toUpperCase()).equals("DESTINATION") ) {
                           Destination = req.getParameterValues(parmName)[0];
                       }
                       if ((parmName.toUpperCase()).equals("TRAVERSE") ) {
                           Traverse = req.getParameterValues(parmName)[0];
                       }
                       if ((parmName.toUpperCase()).equals("LINKTYPE") ) {
                           LinkType = req.getParameterValues(parmName)[0];
                       }
                    }
        }

        if (Source.equals("") || Destination.equals("") ||
                   Traverse.equals("") || LinkType.equals("") )
        { out.println(printFailure() ); }
        else {
                relStore = new relationStore("LinkStore");
                relStore.add_Relation(Source, Destination, Traverse, LinkType);
                out.println("Relation Stored...<BR>");
                out.println("Use find to locate...");
                out.println("</BODY></HTML>");
                }
        }

        /*********************************************/
        private String printFailure() {
```

```
                String retStr = "Error in Input ! Try again !<BR>";
                return retStr;
        }

        /*********************************************/
        private String printHead() {
                String retStr = "<HTML>\n"
                +"<HEAD><TITLE>Returned from Servlet</TITLE></HEAD>\n"
                +"<BODY>\n";
                return retStr;
        }
} /*END CLASS*/
```

8.12 *Relasjonsmodell (Servlet) - relGet*

```
/*****************************************************************
* Demonstraiton - relGet servlet for libRelation
* Returns relations in demonstration front-end
* Creator : Einar Lichtenberg (C) 2000
*/

/**GENERAL*****************************/
import java.io.*;
import java.util.*;

/**SERVLET*****************************/
import javax.servlet.*;
import javax.servlet.http.*;

/**XML PARSER **************************/
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.*;
import org.w3c.dom.traversal.*;
import org.xml.sax.*;

/**RELATION LIBRARY*********************/
import libRelation.*;

/*****************************************************************
* CLASS : relGet
*/
public class relGet extends HttpServlet {

        /* ** GLOBAL VARIABLES ****************** */
        libRelation.relationRetrive relRetrive; //relation library
        PrintWriter out; //The output to webbrowser
        String parmName  = "";
        String parmVal[]   = null;
        String Identificator = null;
        String ReturnType  = null;
        String xmlRelations  = "<?xml?><Relations>";
        String st       = null; //source table
        String dt       = null; //destination table

        /*****************************************************************
        * doPost : Recives post from webbrowser
        */
        public void doPost (HttpServletRequest req, HttpServletResponse res)
                throws ServletException, IOException {

                Enumeration e = req.getParameterNames();
```

```
            if ( !e.hasMoreElements() ) {
                        res.setContentType("text/html");
                    out = res.getWriter();
                            printFailure("No parameters were found!");
            } else {
                while ( e.hasMoreElements() ) {
                   parmName = (String)e.nextElement();
                    if ((parmName.toUpperCase()).equals("IDENTIFICATOR") ) {
                        Identificator = req.getParameterValues(parmName)[0];
                     }
                     if ((parmName.toUpperCase()).equals("RETURNTYPE") ) {
                        ReturnType = req.getParameterValues(parmName)[0];
                     }
                }
        }
        if (Identificator.equals("") || ReturnType.equals(""))
        {
                    res.setContentType("text/html");
                out = res.getWriter();
                    printFailure("You must enter data in the fields !"); }
        else {
                //Set output *****************
                if (ReturnType.toUpperCase().equals("NORMAL") ) {
                    res.setContentType("text/html");
                    out = res.getWriter();
                    servNormal();
                        }
                if (ReturnType.toUpperCase().equals("XML") ) {
                            res.setContentType("text/xml");
                            out = res.getWriter();
                            servXML();
                        }
                }
        }

        /*********************************************************
        * servNormal : servs a html page
        */
        private void servNormal() {
                out.println(printHeadHTML() );

                Vector SourceVec = null;
                Vector DestVec = null;
                //Vector InfoVec   = null;
                Vector LongVec   = new Vector();

                relRetrive = new relationRetrive("LinkStore");
                Vector relVec = relRetrive.get_RelationsList(Identificator);
                out.println(printHTMLHeading("Relations for Identificator :<BR>" + Identi-
ficator ));

                if (relVec == null) { out.println(printHTML("No relations found !") ); }
                else {
                    out.println(printHTML("Number of relations found : "
                            + Integer.toString(relVec.size() ) ) + "<HR SIZE=1 NOSHADE>" );

                    Enumeration e = relVec.elements();
                    if ( !e.hasMoreElements() ) {
                        out.println(printHTML("System Error : Empty Vector !") ); }
                    else {
                        while ( e.hasMoreElements() ) {
                            String id = (String)e.nextElement();
                            Vector InfoVec = relRetrive.get_InfoVector(id);
                            LongVec.addElement(InfoVec);
```

```
                }
                parseAndPrintHTML(LongVec);
            }
        }
} /*END*/

/*********************************************************
* parseAndPrintHTML : parse the XML and make HTML page
*/
private void parseAndPrintHTML(Vector VecMatrix) {
        Vector infoVec = null;
        Enumeration e = VecMatrix.elements();
        if (! e.hasMoreElements() ) {
            out.println("Error : No elements in returned Vector matrix !");
        } else {
            while (e.hasMoreElements() ) {
                infoVec = (Vector)e.nextElement();
                /*SETUP LISTING OF RELATIONS*/
                if (!(infoVec == null) ) {
                    String source = (String)infoVec.get(0);
                    String destination= (String)infoVec.get(1);
                    String traverse= (String)infoVec.get(2);
                    String type= (String)infoVec.get(3);
                    if (source.equals(Identificator) ) {
                    sourceTableIt(destination,traverse,type);
                    } else {
                    destinationTableIt(source,traverse,type);
                    }
                }
            }
            out.println(printSourceTable(st));
            out.println(printDestinationTable(dt));
            out.println(printEndHTML() );
            st = null;
            dt = null;
        }
}

/*********************************************************
* SourceTableIt : makes a table in HTML
*/
private void sourceTableIt(String de, String tr, String ty) {
        if (st == null) st="";
        st = st + "<TR><TD><A TARGET=\"new\" HREF=\"" + de + "\">"
            + de + "</A></TD>"+
                  "<TD>" + formAction(de) + "</TD>" +
                  "<TD>" + tr + "</TD>" +
                  "<TD>" + ty + "</TD></TR>";
}

/*********************************************************
* DestinationTableIt : makes a table in HTML
*/
        private void destinationTableIt(String so, String tr, String ty) {
            if (dt == null) dt="";
            dt = dt + "<TR><TD><A TARGET=\"new\" HREF=\"" + so + "\">"
                    + so + "</A></TD>"+
                      "<TD>" + formAction(so) + "</TD>" +
                      "</TD>" +
                      "<TD>" + tr + "</TD>" +
                      "<TD>" + ty + "</TD></TR>";
}

/*********************************************************
```

```
        * printSourceTable
        */
        private String printSourceTable(String sstr) {
                if (!(sstr == null)) {
                    String retStr = "<P><SPAN CLASS=\"normal\"><B>Source</B>: "
                                + Identificator +
                                "</SPAN>\n"+
                                "<table width=\"100%\" border=\"1\" cellspacing=\"0\" " +
                                "cellpadding=\"5\" bordercolor=\"#000000\" "+
                                "bgcolor=\"#FFBD99\">" +
                                "<TR><TD COLSPAN=2>Destination</TD><TD>Traversal</TD>" +
                                "<TD>LinkType</TD></TR>"+
                                sstr+
                                "</TABLE></P>";
                    return retStr;
                } else return "";
        }

        /*********************************************************
        * printDestinationTable
        */
        private String printDestinationTable(String dstr) {
                if (!(dstr == null)) {
                    String retStr = "<P><SPAN CLASS=\"normal\"><B>Destination</B>: "
                                + Identificator +
                                "</SPAN>\n"+
                                "<table width=\"100%\" border=\"1\" "+
                                "cellspacing=\"0\" " +
                                "cellpadding=\"5\" bordercolor=\"#000000\" "+
                                "bgcolor=\"#FFBD99\">" +
                                "<TR><TD COLSPAN=2>Source</TD><TD>Traversal</TD>"
                                "<TD>LinkType</TD></TR>"+
                                dstr+
                                "</TABLE></P>";
                    return retStr;
                } else return "";
        }

        /*********************************************************
        * formAction
        */
        private String formAction(String relStr) {
        return "<form method=\"post\" action=\"/servlets/relGet\" name=\"Find\">"+
                    "<input  type=\"hidden\"  name=\"Identificator\"  value=\""+  relStr  +
"\">"+
                    "<input type=\"hidden\" name=\"returnType\" value=\"normal\">"+
                    "<input type=\"submit\" name=\"Submit\" value=\"Relations\">"+
               "</form>";
        }

        /*********************************************************
        * servXML : servs a XML page
        */
        private void servXML() {
                out.println(printHeadXML() );
        relRetrive = new relationRetrive("LinkStore");
        Vector relVec = relRetrive.get_RelationsList(Identificator);
                if (relVec == null) { out.println("No relations found !"); }
        else {
         Enumeration e = relVec.elements();
         if ( !e.hasMoreElements() ) {out.println("No relations found !"); }
         else {
                        while ( e.hasMoreElements() ) {
                            String id = (String)e.nextElement();
```

```
                    out.println(relRetrive.get_Relation(id) );
                }
            }
        }
        out.println(printEndXML() );
}

/*********************************************
* printFailure : prints out error message
*/
private void printFailure(String message) {
        out.println(printHeadHTML() + message + printEndHTML() );
}

/*************************************************************
* printHTML: outputs the HTML in stylesheet type
*/
private String printHTML(String str) {
        String retStr = "<SPAN CLASS=\"normal\">"+str+"</SPAN><BR>";
        return retStr;
}

/*************************************************************
* printHTMLHeading: outputs the HTML in stylesheet type
*/
private String printHTMLHeading(String str) {
        String retStr = "<P CLASS=\"heading\">"+str+"</P>";
        return retStr;
}

/*************************************************************
* printHeadHTML: outputs the HTML Header with title
*/
private String printHeadHTML() {
        String retStr = "<HTML>\n"+
                "<HEAD><TITLE>Relations</TITLE>\n" +
                "<style type=\"text/css\">\n"+
                "<!--\n"+
                ".heading {font-family: Arial, Helvetica, sans-serif; "+
                "font-size: 16pt;"+
                "color: #FFBD99; background-color: #167684}\n"+
                ".normal {  font-family: Arial, Helvetica, sans-serif; "+
                "font-size: 12pt;"+
                "color: #000000;}\n"+
                "-->\n"+
                "</style></HEAD>\n<BODY>\n";
        return retStr;
}
/*************************************************************
* printEndHTML: outputs the HTML ending
*/
private String printEndHTML() {
        String retStr = "<HR SIZE=1 NOSHADE>\n"+
            "Page generated at :"+
            "<script language=\"JavaScript\" class=\"topText\">\n"+
                "<!-- hide\n"+
                "var curDateTime = new Date()\n"+
                "document.write(curDateTime.toLocaleString())\n"+
                "//-->\n"+
            "</script>\n"+
            "<BODY></HTML>\n";
        return retStr;
}
```

```
        /*********************************************************
        * printHeadXML : outputs the XML Header
        */
        private String printHeadXML() {
                String retStr = "<?xml version=\"1.0\" ?>\n"
                              +"<Relations>\n";
                return retStr;
        }
        /*********************************************************
        * printEndXML : outputs the XML Header
        */
                private String printEndXML() {
                    String retStr = "</Relations>\n";
                    return retStr;
        }
} /* END CLASS ************************************************/
```