PERSONLIGE SAMLINGER I DISTRIBUERTE DIGITALE BIBLIOTEK

Av

Sverre Magnus Elvenes Joki

Veileder: Professor Ingeborg Sølvberg

Hovedfagsavhandling i Informasjonsforvaltning
ved Institutt for datateknikk og informasjonsvitenskap

NTNU

SAMMENDRAG

I denne avhandlingen presenteres en arkitektur som muliggjør personalisering av digitale bibliotektjenester ved å støtte distribuering av personlige metadata til flere digitale bibliotektjenester. Ved å tilby brukere å lagre personlige metadata eller annoteringer, kan dette danne et bredere beslutningsgrunnlag for informasjonssøkerne, og hjelpe dem til bedre å kvalitetssikre informasjonen de finner. Arkitekturen er implementert ved hjelp av en WebService-arkitektur med SOAP som overføringsprotokoll.

Utgangspunkt for arkitekturen er tre faktorer:

1. Teorier om informasjonssøkeres atferd i søkeprosessene.
2. Undersøkelser av personaliseringstjenester i digitale og digitaliserte bibliotek.
3. Erfaringer fra søkeprosesser.

Ved å forstå hvordan informasjonssøkere oppfører seg i digitale søkesystemer, kan man lettere forstå de prosessene som ligger til grunn for informasjonssøkeres behov, og de krav som stilles til et søkesystem. Dette er nødvendig for å utarbeide krav til en systemarkitektur og et søkesystem. Informasjonssøker-atferd sees i lys av teoriene til Marchionini, Ellis & Haugan, og Choo et. al. Søkeprosessen til Marchionini blir behandlet i større detalj og danner grunnlag for analysen av søkeprosesser senere i avhandlingen. Marchioninis modell forholder seg til faktorer som påvirker søkeprosessen, som f.eks. informasjonssøkeren selv og søkesystemet som brukes, og den søkeprosessen informasjonssøkeren gjennomgår. Søkeprosessen består av 8 delprosesser: Gjenkjenning og akseptering av problem, Definering av problem, Valg av søkesystem, Formulering av spørring, Utføring av søk, Utforsking av resultat, Uttrekking av informasjon, og Refleksjon/avslutning. Marchionini tar utgangspunkt i informasjonssøking som en aktiv prosess der informasjonssøkeren er pådriveren i prosessen. Ellis & Haugan ser på forskjellige informasjonssøkeprosesser i forhold til arbeidsprosesser, og informasjonsbehovene relatert til de forskjellige arbeidsoppgavene. Ellis & Haugan har analysert arbeidsoppgaver i et større prosjekt, og identifisert faser i arbeidsprosessen som har forskjellige informasjonsbehov. Basert på disse funnene har Choo et. al. lansert en modell for informasjonssøking på Internett som tar utgangspunkt i forskjellige oppgaver og forskjellige informasjonsbehov knyttet til disse oppgavene.

Personaliserte tjenester i digitale bibliotek er sentrale for denne avhandlingen, og det er naturlig å se på enkelte av de løsningene som allerede eksisterer. Disse spenner fra personlige biblioteksportaler ved enkelte bibliotek (UBiT, NCSU, VCU), via personlige informasjonsmiljøer (PIE), til hele digitale bibliotek (Greenstone). De personlige *biblioteksportalene* er nettsteder knyttet til de enkelte bibliotek ved utdanningsinstitusjonene. Disse løsningene baserer seg på filtrering av informasjonskilder basert på informasjonssøkerens faglige interesse. *PIE* er en modell som tar sikte på å tilpasse søkesystemer til den enkelte informasjonssøkeren. Denne tilpasningen skjer ved at hver søkekilde

legges til i et repository som brukeren så benytter. Dette gjør at informasjonssøkeren kun forholder seg til PIE, og slipper å søke i de enkelte systemene hver for seg. *Greenstone Digital Library* er en gratis løsning for implementasjon av et digitalt bibliotek. Med Greenstone kan enhver bruker lage sitt eget digitale bibliotek, og på denne måten selv ta vare på informasjon. Denne måten å personalisere sine digitale søkeressurser på, kan være en betydelig jobb for informasjonssøkeren.

Erfaringer fra søkeprosesser danner et naturlig grunnlag for avhandlingen, og identifiserer enkelte deler av søkeprosessen som mer tidkrevende enn andre. Dette gjelder spesielt fasene der informasjonssøkeren må sjekke hvorvidt metadataposten han ser, stemmer overens med det informasjonsbehovet han har, og så ta en beslutning om søkeprosessen har gitt de ønskede resultater. I Marchioninis modell gjelder dette fasen “Trekke ut informasjon” og “Reflektere/ Stoppe”. I de tilfeller der metadata ikke er utfyllende nok, må informasjonssøkeren gå til kilden og undersøke denne. Dette kan f.eks. innebære å lese hele eller deler av informasjonsobjektet han fant.

På dette beslutningspunktet i søkeprosessen kan informasjonssøkeren gis bedre støtte ved at han får tilgang til bredere informasjon om de informasjonsobjekter han undersøker, og denne støtten kan gis ved å åpne for personlige annoteringer av informasjonsobjekter. Annotering eller kommentering av informasjonsobjekter vil gi informasjonssøkeren et bredere beslutningsgrunnlag når han skal velge ut de objektene han ønsker å undersøke videre.

I tillegg til bedre beslutningsstøtte i søkeprosessen, muliggjør personlige annoteringer en kommunikasjon mellom brukere, der de kan utveksle meninger om informasjonsobjekter. Dette er realisert i andre tjenester som f.eks. nettbokhandler, der anmeldelser av bøkene er framtredene. Denne formen for kommunikasjon fyller også et tomrom i digitale bibliotek, der det er liten eller ingen brukerkontakt, i motsetning til i tradisjonelle bibliotek, der brukerne omgås hverandre.

For å bedre informasjonssøkerens beslutningsgrunnlag er det utviklet en arkitektur som tillater brukere å ta aktivt del i et digitalt bibliotek i større grad enn hva tradisjonelle digitale bibliotek og digitaliserte bibliotek tillater.Informasjonssøkeren gis mulighet til å legge til personlige metadata (annoteringer) på informasjonsobjekter, og han får i tillegg anledning til å gruppere informasjonsobjekter i en personlig samling. Den foreslåtte arkitekturen gir informasjonssøkere mulighet til å se andre brukeres meninger om informasjonsobjekter gjennom annoteringene, og det er i tillegg mulig å bruke andres personlige samlinger som et verktøy for å lete etter informasjon.

I utvikligsarbeidet er det laget tre arkitekturer som er evaluert. Deretter er den mest egnede valgt ut for implementasjon. Arkitekturen er implementert i en pilot med begrenset funksjonalitet, og viser at den teknologiske plattformen i arkitekturen fungerer som antatt i forhold til framhenting av personlige metadata. Arkitekturen er implementert som en WebService som bruker SOAP meldinger som kommunikasjonsprotokoll. Denne måten å implementere tjenesten på, gjør det lett å implementere støtte for personlige samlinger i allerede eksis-

terende søkesystemer. Den personlige samlingstjenesten er med andre ord ikke et selvstendig system, men snarere et påbygg til eksisterende digitale og digitaliserte bibliotek. Om personlige annoteringer er riktig måte å gi informasjonssøkere et bredere beslutninggrunnlag på, er ikke bevist, men ved å se på nettbokhandler ser man at i hvert fall de ser en nytteverdi med personlige annoteringer.

Ved å implementere en personlig samlingstjeneste som en tilleggstjeneste, står man overfor utfordringer med å få søkesystem-tilbydere til å integrere støtte for personlige samlinger samt å få brukere til å kommentere informasjonsobjekter.

Denne avhandlingen viser at det er mulig å personalisere digitale bibliotekstjenester ved å integrere personlig metadata i søkesystemene. Implementasjonen viser og at det er mulig å benytte WebServices som teknologisk plattform for distribuering personlige metadata til digitale bibliotekstjenester.

FORORD

Denne avhandlingen er blitt til av en lang og tidvis tung prosess. Jeg vil spesielt takke veilederen min, professor Ingeborg Sølvberg, for kyndig veiledning under hele arbeidet. Uten hennes stødige hånd, er det ikke godt å si hvor dette hadde endt.

En takk går også til alle andre som har deltatt i prosessen med innspill og korrekturlesning. Jeg vil også rette en stor takk til Sissel Odde Steen for støtte i hele prosessen.

Trondheim 02.02.2004

Sverre Magnus Elvenes Joki

INNHOLD

SAMMENDRAG ____ 3

FORORD ____ 7

INNHOLD ____ 9

FIGURLISTE ____ 15

TABELLISTE ____ 17

1 INNLEDNING ____ 19

AVHANDLINGENS MÅL ____ 19

INTRODUKSJON ____ 19

DIGITALE BIBLIOTEK ____ 20

BRUKERMEDVIRKNING ____ 21

PERSONLIG METADATA SOM KOMMUNIKASJON ____ 22

AVGRENSING AV OPPGAVEN ____ 23

2 TEORETISK BAKGRUNN ____ 25

PERSONALISERING ____ 25

NOEN “MITT BIBLIOTEK”-LØSNINGER ____ 26

GREENSTONE ____ 30

PERSONALIZED INFORMATION ENVIRONMENT ____ 31

OPPSUMMERING ____ 33

SØKING ____ 34

MARCHIONINIS SØKEPROSESS ____ 34

FAKTORER ____ 35

SØKEPROSESSEN ____ 36

INFORMASJONSSØKER-ATFERD ____ 37

INFORMASJONSSØKINGSMODELLEN ____ 38

INFORMASJONSSØKER-ATFERD PÅ INTERNETT ____ 40

SØKEMODELLER: OPPSUMMERING ____ 42

3 SØKESCENARIER ____ 43

FAKTORER ____ 43

INFORMASJONSSØKEREN ____ 43

OPPGAVEN ____ 44

AKTUELLE SØKESYSTEMER ____ 44

SØKEPROSESSEN ____ 45

NOEN ANDRE SYSTEMER ____ 46

NOEN SØKESYSTEMER SOM IKKE ER UNDERSØKT I DETALJ ____ 47

BIBSØK NETT ____ 48

BIBSYS ISI-SØK ____ 49

D-LIB MAGAZINE ____ 49

ACM DIGITAL LIBRARY ____ 50

Council of Library and Information Resources 50
Scenarier i Marchioninis modell for søking 51
Søkeprosessen fortalt i form av Marchioninis termer 51
Grensesnitt 52
BibSøk-Nett 53
Bibsys ISI-Søk 54
D-Lib Magazine 55
ACM Digital Library 55
Ta vare på det du finner 56
Oppsummering 57
4 Kravspesifikasjon 59
Personlige samlinger 59
Brukerne 60
Studenter 60
Forsker 60
Institusjon 61
Funksjonskrav 61
Søkefunksjonskrav 61
Bokhyllefunksjoner 62
Grensesnitt 63
Institusjonsbrukere 63
Generelle krav. 64
Systemkrav 64
Avgrensinger 65
Systemtyper 66
Lokal applikasjon eller Nettapplikasjon 67
5 Beskrivelse av samlingstjeneste 69
Arkitekturer 69
Lagdeling og komponenter 69
Systemarkitektur 1 70
Systemarkitektur 2 77
Systemarkitektur 3 81
Integrasjon med andre systemer 86
Valgt systemarkitektur 86
6 Systemdesign og dokumentasjon 89
Mål for piloten 89
Teknologisk bakgrunn 90
Beskrivelse av piloten 91
Sluttbrukertjenester 92
Samlingstjenesten 92
Databasen 98
Oppsummering av teknisk løsning 99

7 KONKLUSJON 101
ARKITEKTUR FOR EN PERSONLIG SAMLINGSTJENESTE 101
PERSONLIGE SAMLINGER I ANDRE SYSTEMER 103
PERSONLIGE SAMLINGER OG BRUKERNES BEHOV 103
EVALUERING AV ARBEIDET 105
VIDERE ARBEID 106
8 REFERANSER 107
APPENDIX A MARCHIONINIS SØKEPROSESS 113
INTRODUKSJON 113
Faktorer 113
Søkeprosessen 116
APPENDIX B SØKEPROSESSEN 121
SØKEPROSESSEN BESKREVET I MARCHIONINIS TERMER 121
APPENDIX C SKJERMBILDER 125
APPENDIX D TEKNOLOGIFORKLARINGER 129
SOAP 129
Z39.50 129
THE HANDLE SYSTEM 130
DOI 130
DUBLIN CORE 130
WEBSERVICES 131
APPENDIX E KLASSEDIAGRAMMER 133
DPC.INTERFACE 133
DPC.DB 134
DPC.XML 135
DPC.DPCSERVLET 136
DPC.DPCSOAPMESSAGE 137
DPC.INTERFACE.DATABASEINTERFACE 138
DPC.INTERFACE.SEARCHINTERFACE 139
DPC.INTERFACE.USERDBINTERFACE 140
DPC.INTERFACE.COLLECTIONDBINTERFACE 141
DPC.INTERFACE.METADATADBINTERFACE 142
DPC.XML.XMLTOUSERDATAPARSER 143
DPC.XML.XMLTOCOLLECTIONRECORDPARSER 144
DPC.XML.XMLTOLOCALMETADATARECORDPARSER 145
DPC.XML.USERDATATOXMLPARSER 146
DPC.CONF 147

APPENDIX F PROGRAM DOKUMENTASJON ____________ 149
DPC.DPCSOAPMESSAGE ____________ 149
DPC.DPCSERVLET ____________ 152
DPC.CONF ____________ 162
DPC.DB ____________ 164
DPC.DB.BOOKSHELF ____________ 164
DPC.DB.COLLECTIONRECORD ____________ 169
DB.DPC.LOCALMETADATARECORD ____________ 175
DPC.DB.PERSONALDESCRIPTION ____________ 180
DPC.DB.USERDATA ____________ 185
DPC.INTERFACE ____________ 191
DPC.INTERFACE.COLLECTIONDBINTERFACE ____________ 191
DPC.INTERFACE.COLLECTIONNOTFOUNDEXCEPTION ____________ 196
DPC.INTERFACE.DATABASEINTERFACE ____________ 197
DPC.INTERFACE.METADATADBINTERFACE ____________ 198
DPC.INTERFACE.METADATANOTFOUNDEXCEPTION ____________ 200
DPC.INTERFACE.SEARCHINTERFACE ____________ 201
DPC.INTERFACE.USERDBINTERFACE ____________ 207
DPC.INTERFACE.USERNOTFOUNDEXCEPTION ____________ 212
DPC.XML ____________ 213
DPC.XML.USERDATATOXMLPARSER ____________ 213
DPC.XML.XMLTOCOLLECTIONRECORDPARSER ____________ 217
DPC.XML.XMLTOLOCALMETADATARECORDPARSER ____________ 223
DPC.XML.XMLTOUSERDATAPARSER ____________ 229
APPENDIX G PROGRAMKODE ____________ 239
DPC.DPCSERVLET ____________ 239
DPC.DPCSOAPMESSAGE ____________ 255
DPC.CONF ____________ 260
DPC.DB.BOOKSHELF ____________ 262
DPC.DB.COLLECTIONRECORD ____________ 265
DPC.DB.LOCALMETADATARECORD ____________ 270
DPC.DB.PERSONALDESCRIPTION ____________ 276
DPC.DB.USERDATA ____________ 280
DPC.INTERFACE.COLLECTIONDBINTERFACE ____________ 287
DPC.INTERFACE.COLLECTIONNOTFOUNDEXCEPTION ____________ 293
DPC.INTERFACE.DATABASEINTERFACE ____________ 293
DPC.INTERFACE.METADATADBINTERFACE ____________ 296
DPC.INTERFACE.METADATANOTFOUNDEXCEPTION ____________ 302
DPC.INTERFACE.SEARCHINTERFACE ____________ 303

DPC.INTERFACE.USERDBINTERFACE ____________________ **312**
DPC.INTERFACE.USERNOTFOUNDEXCEPTION ______________ **318**
DPC.XML.USERDATATOXMLPARSER ____________________ **319**
DPC.XML.XMLTOCOLLECTIONRECORDPARSER ____________ **323**
DPC.XML.XMLTOLOCALMETADATARECORDPARSER __________ **328**
DPC.XML.XMLTOUSERDATAPARSER _____________________ **333**

Figurliste

2–1 GRENSESNITTET I MYLIBRARY@NCSTATE 27
2–2 BOKMERKER I MYLINKS I MYLIBRARY@CORNELL. 29
2–3 ARKITEKTUREN I PERSONALIZED INFORMATION ENVIRONMENT [17] 32
2–4 INFORMASJONSØKEPROSESSEN ETTER MARCHIONINI [36] 36
3–1 SCENARIO I MARCHIONINIS MODELL FOR INFORMASJONSØKING. 51
3–2 KOMPLETT POST FRA BIBSØK NETT 53
3–3 SØKEGRENSESNITTET I ISI-SØK 54
3–4 SØKEGRENSESNITTET I DLIB 55
3–5 SØKEGRENSESNITTET I ACM 56
5–1 SYSTEMARKITEKTUR 1 70
5–2 METADATA-STRUKTUR I SYSTEMARKITEKTUR 1 72
5–3 SCENARIO1 I SYSTEMARKITEKTUR 1: BRUKER LOGGER PÅ SYSTEMET. 73
5–4 SCENARIO 2 I SYSTEMARKITEKTUR 1: BRUKER SØKER ETTER BRUKER VED BRUKERNAVN 75
5–5 SCENARIO 3, SYSTEMARKITEKTUR 1: 76
5–6 SYSTEMARKITEKTUR 2 77
5–7 METADATAPOST FOR SAMLINGER I SYSTEMARKITEKTUR 2 78
5–8 SCENARIO 1, SYSTEMARKITEKTUR 2 79
5–9 SCENARIO 2, SYSTEMARKITEKTUR 2 80
5–10 SCENARIO 3, SYSTEMARKITEKTUR 2 81
5–11 SYSTEMARKITEKTUR 3 82
5–12 METADATAPOST I SYSTEMARKITEKTUR 3 82
5–13 SCENARIO 1, SYSTEMARKITEKTUR 3. 83
5–14 SCENARIO 2, SYSTEMARKITEKTUR 3. 84
5–15 SCENARIO 3, SYSTEMARKITEKTUR 3. 85
6–1 SYSTEMKOMPONENTER I DEN PERSONLIGE SAMLINGSTJENESTEN 92
6–2 SYSTEMDESIGN FOR DEN PERSONLIGE SAMLINGSTJENESTEN. 93
6–3 UML DIAGRAM FOR SEARCHINTERFACE KLASSEN. 94
6–4 KLASSEDIAGRAM FOR DPCSERVLET. 95
6–5 PROGRAMSYSTEMETS HÅNDTERING I FORBINDELSE MED FRAMHENTING AV EN PERSONLIGE BOKHYLLE/SAMLING. 96
6–6 PROGRAMSYSTEMETS HÅNDTERING I FORBINDELSE MED FRAMHENTING AV PERSONLIGE METADATAPOSTER FOR EN GITT OBJEKTIV METADATAPOST. 97
7–1 ANEFALINGER BASERT PÅ ANDRES KJØP. 104
A–1 INFORMASJONSØKEPROSESSEN ETTER MARCHIONINI 117
3–1 SKJERMBILDE FRA POSTVISNING I ET SØKESYSTEM MED PERSONLIGE SAMLINGER. 125
3–3 VISNING AV EN PERSONLIG SAMLING FOR EN BRUKER I ET SØKESYSTEM MED PERSONLIGE SAMLINGER INTEGRERT. 126
3–2 SKJERMBILDE FRA VISNING AV PERSONOPPLYSNINGER I ET SØKESYSTEM MED PERSONLIGE SAMLINGER INTEGRERT. 126
E–1 UMLDIAGRAMFOR PAKKEN DPC.INTERFACE 133
E–2 UML-DIAGRAM FOR PAKKEN DPC.DB 134
E–3 UML-DIAGRAM FOR PAKKEN DPC.XML 135
E–4 KLASSEDIAGRAM FOR DPCSERVLET 136
E–5 KLASSEDIAGRAM FOR DPCSOAPMESSAGE 137
E–6 KLASSEDIAGRAM FOR DATABASEINTERFACE 138
E–7 KLASSEDIAGRAM FOR SEARCHINTERFACE 139

E–8 KLASSEDIAGRAM FOR USERDBINTERACE 140
E–9 KLASSEDIAGRAM FOR COLLECTIONDBINTERFACE 141
E–10 KLASSEDIAGRAM FOR METADATADBINTERFACE 142
E–11 KLASSEDIAGRAM FOR XMLTOUSERDATAPARSER 143
E–12 KLASSEDIAGRAM FOR XMLTOCOLLECTIONRECORDPARSER 144
E–13 KLASSEDIAGRAM FOR XMLTOLOCALMETADATARECORDPARSER 145
E–14 KLASSEDIAGRAM FOR USERDATATOXMLPARSER 146
E–15 KLASSEDIAGRAM FOR CONF 147

Tabelliste

2–1 SAMMENLIGNING AV PERSONALISERINGSTJENESTER 33
2–2 SKUMMINGSMETODER FRA CHOO ET. AL.[8] 40
2–3 CHOOS MODELL FOR SØKING I NETT RESSURSER 41
3–1 OVERSIKT OVER SØKESYSTEMER 45
3–2 OVERSIKT OVER SØKETJENESTER I OMTALTE SØKESYSTEMER 58
4–1 OPPSUMMERING AV BRUKERES EVNER 61
4–2 SØKEFUNKSJONER 62
4–3 BRUKER-METADATA 63
4–4 PERSONLIG METADATA 63
4–5 OPPSUMMERING AV FUNKSJONSKRAV 64
5–1 METADATAPOST I BRUKERDATABASEN 71
A–1 BESKRIVELSE AV EN OPPGAVE 114

1 INNLEDNING

1.1 Avhandlingens mål

Denne avhandlingen har som mål å beskrive en arkitektur for personlige samlinger som kan brukes for å distribuere personaliserte metadata (annoteringer) til flere søketjenester. Arkitekturen lar informasjonssøkere kommentere informasjonsobjekter og ta vare på egne samlinger. Arkitekturen er delvis implementert som en pilot.

1.2 Introduksjon

> *Searching through the petabytes of fact, fiction, and rumor that make up the World Wide Web is no mean task. It's like wandering through a library without a filing system or a library catalogue.*
>
> *Sean Carroll, PC-Magazine (2003)*

I informasjonssøkingens tidsalder har bibliotekene stått sentralt som formidlere av kvalitetssikret informasjon. Etter hvert har Internett tatt over mye av denne formidlingsrollen[1], og informasjonssøkerne må selv overta ansvaret for kvalitetskontroll av informasjonen. Siden Internetts spede begynnelse har mengden informasjon som er tilgjengelig på Internett økt enormt, og søkemotoren AllTheWeb[1] har indeksert over 3 milliarder nettsider[2]. Et søk på "digital library" gir over 1 million treff, og det sier seg selv at man må være meget selektiv for å finne det man egentlig er på jakt etter.

1. Bruken av datamaskiner og Internett har økt kraftig siden 1997 [32], og tilgangen på raske Internettforbindelser har også økt [31]. Søkemotorer blir også brukt mye, og Googles antatte markedsverdi på 104 milliarder norske kroner viser dette (omtrent samme markedsverdi som Norsk Hydro)[40] .
2. 19.11.2003 hadde de 3.151.743.117 nettsider i indeksen i henhold til forsiden www.alltheweb.com

Som nevnt var bibliotekene sentrale i informasjonsverden. De kvalitetssikret informasjon og gjorde denne tilgjengelig for sine brukere. I dagens informasjonshverdag har Internett tatt over mye av den søkingen man tidligere gikk til bibliotekene for å gjøre. I tillegg må enhver informasjonssøker være sitt eget filter for å filtrere bort ikke-relevant informasjon. Problemene med mengde og kvalitet, skissert ovenfor, dreier seg om informasjonsoversvømmelse, og løsningene er flere.

En av løsningene er å lage bedre informasjonsgjenfinningsteknikker, slik at søkene i søkemotorer som AllTheWeb eller Google[20] gir de resultatene søkeren faktisk er på jakt etter. Problemene for denne strategien er blant annet at mye av informasjonen vi trenger, ikke er tilgjengelig for søkemotorene. Og det som søkemotorene ikke klarer å indeksere, gir heller ikke informasjonssøkerne treff når de søker. Dette inkluderer informasjon som er passordbeskyttet eller f.eks. ligger i en database.

En annen løsning på informasjonsproblemet er å tilby en informasjonssøker et grensesnitt som bare inneholder data han er interessert i. Her kommer f.eks. biblioteket inn i bildet. Ved å gå på kjemibiblioteket på et universitet vil man høyst sannsynlig klare å finne informasjon rettet mot nettopp kjemikere. Man har en garanti for at det en finner, er kvalitetssikret og relevant i forhold til brukerens informasjonsbehov. Problemet her er at dette må man til biblioteket for å få, og tjenester på Internett er lettere tilgjengelige. Her kommer begrepet digitalt bibliotek inn.

Ved å tilby bibliotektjenester digitalt, kan man tilby både den kvalitetssikringen som oppleves i et bibliotek, og tilgjengeligheten til Internettjenester. Denne løsningen er ikke ny, og man har allerede mye forskning på området. Den Europeiske Konferansen om Forskning og Avansert Teknologi for Digitale Bibliotek (ECDL) er et bevis for dette med sin popularitet[15] .

1.2.1 Digitale bibliotek

Termen “Digitale bibliotek” brukes om mange forskjellige systemer. I noen tilfeller brukes det om Internett-tilgang til bibliotekets katalogsystem; et digitalisert bibliotek. I andre tilfeller snakker man om et komplett bibliotek i digital form på Internett. Whatis.com definerer “digitalt bibliotek” som en samling dokumenter i organisert elektronisk form[59] . Aalberg og Hegna trekker fram at en digitalt bibliotek “dreier seg om mer enn digitalisering av tradisjonelle bibliotekstjenester”[61] . De trekker deretter fram 6 punkter som er karakteristiske for digitale bibliotek:

- Informasjonssentrering

  Informasjon forvaltes og gjøres tilgjengelig. Informasjonen er tilgjengelig i form av informasjonsobjekter.

- Digitale informasjonsobjekter

  Det er digitale informasjonsobjekter eller referanser til slike, som fyller et digitalt bibliotek. De påpeker at referanser til fysiske objekter ikke må utelates.

- Samling-orientering

  Et digitalt bibliotek er bygd opp av kvalitetssikrede samlinger, men man kan også ha andre typer “virtuelle samlinger”.
- Nettbasert informasjonsdeling

  Bruk av Internett og andre typer nettverk er viktig for deling av informasjon i digitale bibliotek.
- Integrerte løsninger

  For å tilfredsstille brukernes informasjonsbehov trengs det effektive løsninger tilpasset brukernes ekspertisenivå.
- Interaktivitet

  Den tilgjengelige teknologien muliggjør interaksjon mellom brukerne og informasjonen.

I dag har man tilgang til flere løsninger basert på disse kriteriene. Løsningene spenner fra enkle digitale bibliotekkataloger til mer avanserte søkesystemer med tilgang til fulltekstdokumenter og personlige tilpasninger.

De digitale bibliotekene er i konstant endring, og mye har skjedd i løpet av de siste fem årene. Tidligere hadde man ofte fokus på å lage løsninger som ga brukere tilgang til å søke i bibliotekkatalogene med et elektronisk verktøy, ofte kalt et digitalisert bibliotek. Dagens digitale bibliotek er ikke så sterkt fokusert på tradisjonelle bibliotekkataloger, og det fokuseres nå i større grad på å tilby mer enn disse. Men man kan likevel ikke si at de digitale og digitaliserte bibliotekløsningene som finnes i dag, er fullverdige substitutter for et tradisjonelt bibliotek. Flere har sett, og ser fortsatt, et behov for å tilby tjenester i de elektroniske bibliotekene som gir flere tjenester enn kun søking og gjenfinning av metadata. Dette har ført til at bibliotektjenester er blitt flyttet over på Internett, og søketjenestene har utviklet seg fra enkle bokkatalog-søk til mulighet for å søke i flere databaser samtidig, og i større tilgang til informasjon i digital form. Overgangen fra det digitaliserte bibliotek til det hel-digitale ser altså ut til virkelig å skyte fart. Dette ser man blant annet i antallet tidsskrifter fag-institusjoner abonnerer på. Antallet elektroniske tidsskrifter i BIBSYS Bibliotekbase har økt fra 8.000 i juli 2001 til over 14.800 i januar 2004[5] , og Universitetsbiblioteket i Trondheim planlegger å si opp tusener av trykte tidsskrifter[28] .

1.2.2 *Brukermedvirkning*

Dagens digitale bibliotek har ofte som mål å understøtte en brukers behov for informasjon. Dette har gjort at mange av disse systemene har fokusert på søkeverktøy og hvordan det kan gjøres enklest mulig å finne ting, og ikke i så stor grad på brukernes innvirkning på systemet.

Et eksempel på brukermedvirkning i et biblioteksystem: I en tenkt situasjon står en person ved en bokhylle på et tradisjonelt bibliotek. Ved siden av han står en annen person som kikker i samme hylle. Av denne situasjonen kan man anta at disse to brukerne i hvert fall har én sammenfallende interesse. Om den ene av disse personene lurer på om en bok er god e.l., kan han spørre personen

ved siden av, som kanskje har et svar. Dette konkrete eksempelet er ikke overførbart til et digitalt bibliotek. Brukere i digitale bibliotek har ikke kontakt med hverandre på samme måte som man kan ha om man står ved siden av hverandre ved en bokhylle.

Flere selskaper tilbyr i dag brukerforum der brukere kan stille spørsmål direkte til hverandre. Dette er ikke ulikt det som skjer i det tenkte eksemplet ovenfor. Brukere hjelper hverandre, uavhengig av systemet og selskapet. Dette er selvsagt en stor besparelse for bedriften som dermed slipper å ansette folk for å svare på disse spørsmålene. Denne typen brukerkontakt er også utbredt andre steder på Internett som f.eks. SlashDot.org.

Ved North Carolina State University (NCSU) ble det gjort brukerundersøkelser som konkluderte med at brukere likte de elektroniske tjenestene som ble tilbudt. Brukerne anså dem som praktiske, men de likte også biblioteket som et fysisk sted. Resultatet viste også at brukerne ikke var klar over alle tjenestene som biblioteket tilbød. [37]

Brukerne kan trekkes inn i digitale bibliotek på flere måter. En måte er å åpne for forumtjenester som nevnt i forrige avsnitt, men dette er kanskje ikke helt ideelt, siden det må være mange brukere til stede i forumet for at en bruker med et begrenset interessefelt skal få svar på sine spørsmål. Denne løsningen vil også kreve at brukere er "til stede" i det digitale biblioteket. Dette kan f.eks. bety at de må være innlogget og følge med på et evt. forum, og svare om de ser spørsmål som faller inn under deres fagområde. Det sier seg selv at mange brukere ikke vil ta seg tid til dette, da det er meget tid- og oppmerksomhetskrevende.

1.2.3 Personlig metadata som kommunikasjon

En annen løsning for å overføre kunnskap om et dokument, er at brukere skriver kommentarer på det, eller skriver anmeldelser av det. En slik anmeldelse kan en annen bruker senere lese uten å forstyrre forfatteren av anmeldelsen. Dette systemet er ikke ulikt et system der lesere av en bok skulle skrive kommentarer på "gul-lapper" og klistre dem på boka. Da kunne andre lesere av denne boka lese andre brukeres kommentarer uten å være avhengig av å treffe vedkommende ved bokhylla i biblioteket. Dette er et system som er i utstrakt bruk i nettbokhandlene, der bokanmeldelser er framtredende[3]. Ved å la informasjonssøkere få tilgang til personlig metadata fra andre brukere, vil søkeren få et bredere grunnlag for å ta avgjørelser. Et bredt beslutningsgrunnlag gir brukeren mer informasjon i forhold til de avgjørelsene han må ta, og brukeren kan lettere skille god informasjon fra dårlig.

3. Mao.no tilbyr sågar 20 kroner i rabatt på neste kjøp, til den første som anmelder en bok. Dette viser at nettbokhandlene verdsetter bokanmeldelser.[33]

1.3 Avgrensing av oppgaven

Som beskrevet tidligere i innledningen, er det mange aspekter som er avgjørende for hva man skal tilby av tjenester i et digitalt bibliotek, og hvordan man skal gjøre det. Denne avhandlingen tar sikte på å beskrive en arkitektur som lar brukere få tilgang til personlige metadata i et søkesystem. Det tas ikke sikte på å beskrive en komplett løsning for et digitalt bibliotek. Det er også et mål at man skal kunne distribuere disse data til flere systemer.

For å finne fram til en arkitektur som beskrevet ovenfor, drøftes først noe teori om personaliseringteorier og -tjenester. Deretter gjennomgås informasjonssøkeprosesser. Informasjonssøkeprosessene danner grunnlag for egne erfaringer i noen søkegrensesnitt. Basert på disse teoriene, og på egne erfaringer fra søking, er det laget krav til hva en arkitektur for personlige samlinger skal beskrive. Deretter forklares arkitekturen, og til slutt beskrives en pilot-implementasjon av denne arkitekturen.

2 TEORETISK BAKGRUNN

2.1 Personalisering

Siden Internett er blitt allemannseie, har økningen av tilgjengelig informasjon nærmest eksplodert. Tidligere kunne man klare seg med det lokale biblioteket og det tilhørende biblioteksystemet for å finne det man trengte av informasjon. I dag, derimot, er mer og mer av informasjonen man trenger, tilgjengelig i forskjellige, mer eller mindre proprietære søkesystemer på Internett. Dette gjør at man som informasjonssøker er nødt til å forholde seg til mange og til dels ulike informasjonskilder, og dette kan fort føre til for stor tilgang på informasjon. Dette fører ofte til en tidkrevende gjennomgang av søkeresultater og informasjon, og til tross for mye informasjon å velge i, er man ikke garantert et godt resultat av søket.

For å dempe informasjons-overbelastningen på brukere har mange foreslått å personalisere søkeomgivelsene. Brukerne ser seg ikke tjent med overflødig informasjon, og forventer bedre løsninger. Meningen med å personalisere søkeomgivelsene er at informasjonstjenestene tilpasses brukeren og tilbyr den riktige informasjonen til de riktige brukerne. En annen grunn til å personalisere biblioteksystemet er at brukere forventer den slags løsninger [9] , [27] . Dette er en trend som har spredt seg fra andre nettsteder brukerne bruker, som MyYahoo! og Amazon.com. Etter hvert som brukere bli mer sofistikerte, søker de andre nettsteder enn enorme portaler. Dette er steder som bedre passer søkernes behov[27] .

Informasjonsomgivelsene kan personaliseres på flere måter. I noen sammenhenger lager man seg sitt eget "vindu" mot biblioteket der man kan legge egne lenker o.l., noe som øker tilgjengeligheten av ønsket informasjon. Dette er løsningen som er valgt av Cornell University Library[9] , North Carolina State University[37] og Virginnia Commonwealth University[21] [19] . En annen løsning er å lage sin egen samling som foreslått i PIE-arkitekturen[17] [18] . En tredje løsning er å lage sitt eget "bibliotek", og da kan man bruke Greenstone Digital Library[4] [55] [56] [57] [58] . Selv om personalisering ikke er det mest

nærliggende temaet når man snakker om Greenstone og et eget digitalt bibliotek, kan det likevel la seg gjøre.

For å forstå bedre de ulike løsningsforslagene, følger det videre en introduksjon til "Mitt Bibliotek"-løsningene ved NCState University[37] , Cornell University[9] , Virginnia Commonwealth University[19] [21] , og Universitetsbiblioteket i Trondheim[48] . Deretter følger en liten gjennomgang av Greenstone Digital Library[4] [55] [56] [57] [58] og PIE-arkitekturen[17] [18] .

2.1.1 Noen "Mitt Bibliotek"-løsninger

"Mitt Bibliotek"-løsningene som vi har sett på her, har som mål å lette hverdagen til brukerne gjennom personalisering av bibliotekets nett-grensesnitt. En gjennomgående problemstilling for bibliotekene er den store informasjonsmengden brukere av informasjonsressurser blir oversvømt med til stadighet. Ved å begrense ressursene en bruker tilbys til de som er relevante for vedkommende, kan man redusere "drukningsfaren".

Cohen et. al. ved Cornell University Library[9] ser at man har tatt hensyn til personaliseringstrenden på flere og flere nettsteder som MyYahoo!, MyCNN etc. De ser ingen grunn til at biblioteket som informasjonstilbyder ikke skal kunne tilby de samme tjenestene. Bibliotekbrukere forventer å kunne tilpasse omgivelsene på biblioteket, på samme måte som de andre nettstedene de bruker kan tilpasses deres behov. Cohen et. al. peker også på at The Library and Information Technology Association(LITA) har definert Mitt Bibliotek-tjenester som "the number one trend to watch"[30] . Siden bibliotekene konkurrerer med Internett om å tilby riktig informasjon, ser Cohen et. al. at en personalisert løsning er en måte å sikre at bibliotekbrukere kommer tilbake til biblioteket.

Ved Virginnia Commonwealth University (VCU) begynte personaliseringarbeidet med en analyse av bibliotekets nett-grensesnitt. De fant ut at en statisk nettside ikke gjorde informasjon mer tilgjengelig for brukerne. Dette problemet forsøkte de å løse ved å la brukere personalisere grensesnittet mot bibliotektjenestene. I sitt arbeid la de stor vekt på å redusere antall museklikk en bruker måtte utføre for å få tilgang til en tjeneste. De ønsket å gjøre all informasjon som var relevant for en bruker, tilgjengelig på én side.[21] [19]

Universitetsbiblioteket i Trondheim (UBiT) har gjort et tilsvarende arbeid for å personalisere bibliotektjenestene. Det er lite som skiller resultatene fra UBiTs prosjekt fra eksemplene nevnt ovenfor. Ut fra ønsket om at brukerne i større grad skal betjene seg selv etter modell fra nettbanker etc., og influert av trender på Internett (som beskrevet for Cornell University Library), ble "Mitt Bibliotek" implementert.

Tjenestene i Mitt Bibliotek-løsningene

Den vanligste løsningen på problemene som er nevnt ovenfor, er å lage et grensesnitt som brukerne selv kan tilpasse. Bibliotekene over har valgt denne løsningen, og nedenfor blir tjenestene og grensesnittene til disse forklart i korte trekk.

NCSU, som de fleste andre, valgte å lage personaliserte grensesnitt, der brukere kunne tilpasse sine egne sider (Se Figur 2–1 på side 27). Når en bruker registre-

Figur 2–1: Grensesnittet i MyLibrary@NCState

rer seg, innebærer det å registrere informasjon om seg selv, og informasjon om f.eks. akademisk interesse. På bakgrunn av dette får brukeren en ferdig side som er tilpasset det interesseområdet han tilhører, men denne kan selvsagt endres om brukeren ønsker det. En av fordelene med å lage et utgangspunkt for brukeren, er at man kan gjøre brukeren oppmerksom på de mulighetene han har. Grensesnittet inneholder flere elementer:

- Meldinger fra biblioteket og bibliotekaren som har ansvar for brukerens fagområde (“Din bibliotekar”). Disse er ment å endres regelmessig for å holde brukerne oppdatert om nyheter etc.
- Kontaktinformasjon til “Din bibliotekar”, slik at brukeren kan henvende seg til biblioteket på en enkel måte.

- Lenker til universitetets og bibliotekets nettsider som er relevanter for brukeren. Disse kan endres av brukeren ved valg fra en liste aktuelle sider.
- Bibliografiske databaser og elektroniske tidsskrifter tilpasset brukerens fagområde. Det er meningen at utvalget skal være begrenset til det brukeren finner interessant. I tillegg er det en seksjon med lenker til faglig relevante Internett-ressurser.
- Personlige lenker fungerer som et bokmerke-system i nettleseren. Brukeren kan her legge inn hvilke som helst lenker som ikke finnes i de ovenfor nevnte seksjonene.
- Søkemuligheter som inkluderer internett-søkemotorer og bibliografiske søketjenester, er gjort tilgjengelige i en "søkeboks" kalt "Quick Search".
- Brukeren kan lage seg søkeagenter som utfører periodiske søk mot biblioteksbasen og eventuelt sender resultatene på e-post.

Cornell University Library har laget en nett-tjeneste som lar brukere registrere søkeagenter som går mot bibliotekbasen, og lagre bokmerker til ressurser på Internett. Tjenestene kalles henholdsvis MyUpdates og MyLinks. Ved jevne mellomrom utfører agenten søk i bibliotekkatalogen ved biblioteket. Hvis den finner noe nytt siden sist, sender den resultatet i e-post til brukeren. Den andre tjenesten, lenkesamlingen (Se Figur 2–2 på side 29), lar brukeren ta vare på lenker til nettsteder. Disse kan brukeren organisere i mapper.

VCU har valgt å gjøre flere tjenester tilgjengelige i ett grensesnitt på samme måte som NCSU. Databasene og bibliotektjenestene kan tilpasses av den enkelte bruker. Dette gir han kortere vei til den informasjon han ønsker, og fjerner den han ikke ønsker. VCU tilbyr videre en oversikt over populære databaser. Bokmerkefunksjonen lar en bruker lagre opptil 6 lenker til hvilke som helst nettsteder. En søkeboks er integrert på nettsiden, og kan tilpasses slik at den søker mot brukerens favoritt-søkemotor. Det finnes også en lenke til en egen tidsskriftside som tilbyr en liste over aktuelle tidsskrifter for brukeren. Videre finner man en lenke til en nyhetsbulletin, f.eks. driftsmeldinger, om MyLibrary-tjenesten.

Med få unntak er alle tjenestene som tilbys, tilgjengelige i én nettside. Unntaket er tidsskriftene som er tilgjengelige i en egen nettside, men lenket til fra hovedsiden.

Tjenestene som UBiT tilbyr, er nesten identiske med NCSUs tjenester. UBiT har en søkeboks som kan tilpasses av brukeren, en liste over elektroniske tidsskrifter som er relevante for fagfeltet til brukeren, en bibliotekar som har ansvaret for brukerens fagområde og tilsvarende "dagens melding", samt en liste over personlige lenker. Den personlige tilpasningen er ganske begrenset i forhold til NCState University Library nevnt over. Man kan kun velge fagtilhørighet fra en liste over fagområder, og denne passer godt sammen med listen over institutter og enheter ved universitetet. Dette burde gjøre det lett for brukere å velge riktig fagtilhørighet. I motsetning til NCSUs løsning er det ikke mulig for brukeren ved UBiT å velge hvilke tjenester han ønsker i sin side.

MyLibrary Home
Logout
Help

MyLibrary @Cornell

MyLinks for Guest Login

MyFolders	Add Folder | Edit All Folders
Library Services	Internet Search Engines
E Journals	University Libraries
Library Web Tutorials	Personal Links
LAW101	Mylibrary literature
Database Links	Engineering "stuff"

Library Services — Add Resource | Edit This Folder

CU Library Catalog — CU Library Gateway
New Book Request — Reference Desks Directory
Request Book from Annex — Reserve
ou

Top

Internet Search Engines — Add Resource | Edit This Folder

All the Web, All the Time--FastSearch — Alta Vista
Google — Google Groups
HotBot

Top

E Journals — Add Resource | Edit This Folder

Conference proceedings on information retrieval. — D-Lib Magazine

Figur 2–2: Bokmerker i MyLinks i MyLibrary@Cornell.

En viktig tjeneste som tilbys av bl.a. UBiT og VCU, er adgangskontroll mot eksterne ressurser. Dette blir forklart nærmere nedenfor.

Løsningene i Mitt Bibliotek-løsningene

"Mitt Bibliotek"-løsningene som er presentert ovenfor, er enkle nettsteds-implementasjoner som lagrer data om en bruker i en database og genererer grensesnittet basert på dette. UBiT, Cornell og NCSU bruker samme brukeridentifikator i MyLibrary-løsningen som ved resten av universitetet. Dette letter brukeren for å huske mange brukernavn og passord. Hvordan dette er integrert i de forskjellige systemene er ukjent.

Som sagt har VCU og UBiT en adgangskontroll mot den personaliserte bibliotektjenesten som åpner for tilgang til bibliotekkatalogene og eksterne ressurser. Dette gjøre ved hjelp av en proxy-server, og sørger for at brukere får tilgang til alle tjenestene de trenger, uten at de trenger å være på biblioteket (eller campus i UBiTs tilfelle). Ved UBiT har de vurdert flere modeller for å gjøre dette, og

kommet fram til at dette ikke er en spesielt sikker måte å løse adgansproblematikken på. Fordelen er at den er populær og enkel for brukerne. Tjenester som ivaretar adgangskontrollen, ligger i inngangen til MittBibliotek-løsningen.

VCU har også tatt i bruk systemet som et læringsverktøy, noe som har vist seg å være populært. Det opprettes egne "Mitt Bibliotek"-sider for fagene, og studentene får tilgang til informasjon tilpasset det faglige behovet. Dette åpner muligheter innen begrensning av informasjonstilgangen, ikke bare for enkeltpersoner, men også for grupper med like mål. Sidene fungerer akkurat som vanlige "Mitt Bibliotek"-sider, men personaliseringen er tilpasset en gruppe mennesker. Det er ikke mulig for den enkelte bruker å tilpasse sidene, men han slipper å gjøre dette for å ta i bruk systemet, siden det på en måte er ferdig tilpasset.

Bruken av Mitt Bibliotek-løsningene

Det er ikke funnet mange informasjonskilder om bruken av "Mitt Bibliotek"-tjenester. Kilden til statistikk om bruken kommer fra Virginnia Commonwealth University (VCU).

Erfaringene fra VCU viser at de personlige nettløsningene ikke har fått den oppslutningen fra sluttbrukerne som man hadde håpet på. 4% av kontoene i systemet sto for 61% av bruken i år 2001. Dette vil si at det er få som bruker systemet, men at disse har brukt det en hel del. Det positive er at det har vært en økning i bruken av systemet fram til 2002, og brukerne har tatt i bruk systemet i større grad enn tidligere[19] . Det virker altså som om de som bruker systemet er fornøyde, og at de bruker det oftere nå enn tidligere.

2.1.2 Greenstone

Greenstone er et digitalt biblioteksystem for lagring og presentasjon av store informasjonsmengder, og er et alternativ til de som vil legge større mengder informasjon ut på nettet. Med innebygd støtte for z39.50[60] , Dienst[10] og Standford Infobus er det er distribuert system som kan søke i mange andre søkesystemer. Greenstone er ikke et personaliserings-alternativ på samme måte som "Mitt Bibliotek"-tjenestene, men tanken må ikke utelukkes helt.

Systemet er distribuert gratis under GNU-lisens, noe som gjør at systemet kan tas i bruk av mange som ellers ikke ville hatt råd til et kommersielt digitalt biblioteksystem. Greenstone er også laget med tanke på å kunne brukes på datamaskiner som ikke er knyttet opp mot Internett, og kan kjøre fra CD-ROM på til dels gamle systemer som bl.a. Windows 3.1. Alt i alt er det mye som taler for at dette systemet skal kunne tas i bruk av de fleste, uavhengig av økonomi og operativsystempreferanser.

FN har tatt i bruk Greenstone, og bruker det til en god del humanitære samlinger. Disse er laget som samlinger i et større biblioteksystem, og ikke som egne digitale bibliotek. Det er riktignok en ulempe med disse samlingene. De er å regne som selvstendige digitale bibliotek. Hver og en samling må aksesseres separat, noe som medfører at man ikke kan søke i flere samlinger samtidig.

Søkemulighetene er likevel gode, og det er mulig å søke i både fulltekst og bibliografiske felter som forfatter, tittel etc.

Samlingene i Greenstone er definert av brukeren eller operatøren. Han kan legge til flere samlinger på en tjener, og disse samlingene tilbys publikum gjennom et ferdig grensesnitt. Arbeidet med å sette opp en slik samling er forholdsvis lett, og flere typer institusjoner kan lage sine egne samlinger, på samme måte som FN har forskjellige samlinger. Ved at flere brukere setter opp egne digitale samlinger på denne måten, får man en stor mengde delt data, uten at det kreves en stor samlet innsats. Ulempen med Greenstone er at det er et stort system som krever at brukeren har god kompetanse på informasjonsteknologi for å sette det opp for bruk. Dette begrenser klart utstrekningen det vil få for sluttbrukere uten denne kompetansen.

2.1.3 Personalized Information Environment

Bakgrunn

Personalized Information Environment (PIE) [17] [18] er en arkitektur der den enkelte bruker kan bygge sitt eget informasjonsmiljø (sine egne samlinger), fra en større samling informasjonsressurser. Utgangspunktet er her det samme som i mange av de andre tilnærmingene: et behov for å begrense informasjonsmengden til et nivå brukerne kan håndtere. French og Vile ser for seg at en informasjonssøker kan gjøre dette ved å lage en samling av søkesystemer, og at han bruker denne i stedet for de andre systemene, som gir støy i søkingen. Bakgrunnen for PIE er at dagens løsning for å søke i flere søkesystemer ikke er god nok for å begrense informasjonsoversvømmelsen. French og Vile mener også at informasjonstilgangen er tilbyder-sentrert og ikke brukersentrert. Dette gjør det vanskeligere for brukerne å finne det de er ute etter.

French og Vile definerer to typer søkesystemer blant dagens søkemotorer. Den ene har et sentralisert syn, mens den andre har det de kaller en "Metasearch view". Med den første mener de at søkesystemet prøver å gjøre all informasjonen man trenger, tilgjengelig i ett søkesystem. Eksempler på dette er de tradisjonelle søkemotorene på nettet som Google og AltaVista. Det er flere problemer knyttet til denne løsningen. Det første er bl.a. at det er for mange dokumenter tilgjengelig til at de er samlet på ett sted. Denne informasjonen er en meget heterogen masse, men av en homogen type (html). Et annet problem er at flere og flere nettsteder er dynamiske og har informasjon lagret utilgjengelig for søkemotorer, samt problemet med døde lenker og sikkerhetssystemer som hindrer adgang til informasjon. En annen av grunnene til at slike systemer er utilgjengelige kan være at de som vil ta seg betalt for informasjon, ikke har noen måter å få til dette på, og at de derfor velger å ikke dele noe informasjon i det hele tatt. Dette fører til at informasjonen ikke kan indekseres av søkemotorene.

I et "Metasearch-View" blir flere sentraliserte søkemotorer innkapslet i ett søkesystem. Dette åpner for flere problemer. For det første er det vanskelig å lage en slik tjeneste, fordi utviklerne av den må kjenne til den interne oppbygningen av de enkelte søkemotorene. Dernest får man problemer med å intera-

gere med søkeresultatene. Tilpasningsmulighetene brukerne får i slike systemer er i tillegg dårligere enn i de sentraliserte søkemotorene. Dermed ser man behovet for nytenking, som French og Vile gjør med PIE.

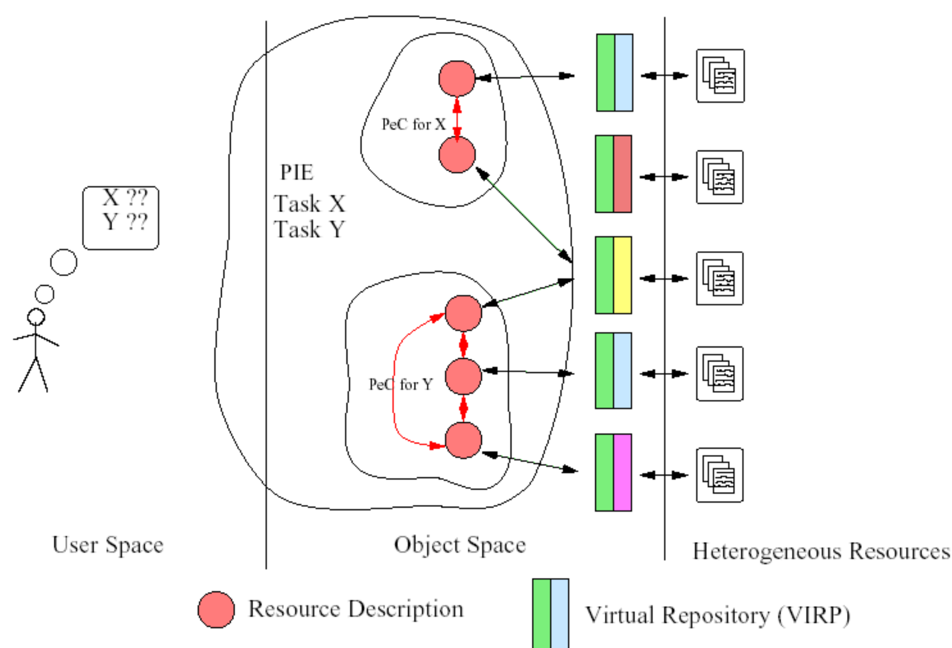
Arkitekturen

Arkitekturen i PIE er laget på bakgrunn av fire prinsipper:

- Tilpasningsevne
- Effektive og effektfulle søk
- Kontrollerbar deling
- Sikkerhet og personvern

Tilpasningsevne vil si at brukere enkelt kan lage sine egne samlinger og tilpasse dem etter hvert som de ønsker det. Brukere kan velge hva som skal være med i samlingen, og hva som ikke skal være med. Ettersom søket foregår i den personlige samlingen, må de personlige samlingene oppdateres etter hvert som endringer skjer, for at brukerne skal få gode søkemuligheter. Oppbygging av personlige samlinger kan være tidkrevende, og derfor er det ikke utenkelig at man kan ønske å dele disse med andre. En personlig samling kan bruke ressurser som krever spesielle rettigheter eller betaling for adgangen, og brukerne kan ønske å beskytte sine samlinger for andre. Da er det viktig at dette støttes av den personlige samlingen.

Byggeklossene i PIE er personaliserte samlinger, virtuelle lager, og ressursbeskrivelser. Arkitekturen kan sees i Figur 2–3 på side 32. Alle ressurser man har i



Figur 2–3: Arkitekturen i Personalized Information Environment [17]

en palett (en mengde ressurser), har et grensesnitt. Dette grensesnittet kalles VIRP (Virtual repositories).

Personaliserte samlinger (Personalized Collection, PeC) er en brukers egne samlinger av søke-ressurser. Disse samlingene er bygd opp av ressurser fra en palett der brukeren selv velger hvilke han vil ta med. Virtuelle lager (Virtual Repositories, VIRP) er grensesnittet inn mot en ressurs. Det kan kalles en innpakning, og gjør det mulig for samlingen å utføre fornuftige søk i ressursen. En ressursbeskrivelse (Resource Description, RD) er megler mellom PeC og VIRP. Den har en lenke til selve ressursen, så den fungerer også som en adgangsbillett. Når en PeC inneholder en gitt RD, kan brukeren av PeC'en bruke ressursen RD'en beskriver. Som det går fram av Ligning 2–3 på side 32, vil en PeC inneholde flere RD'er, men også "... significant information related to user manipulations. Minimally, this would include desktop organization information and user annotations ..."[11]. Dette åpner for at brukere kan legge til informasjon om en PeC, for egen eller andres del.

Med PIE-arkitekturen kan man som sagt lage seg egne ressurser i form av samlinger av ressurser. Disse kan deles med andre som har samme interesser. Dette konseptet kan godt utvides til å lage samlinger av annet enn søkemotorer. Det er ikke noe i veien for å se på digitale biblioteksystemer som ressurser.

2.1.4 Oppsummering

De personaliseringsløsningene som er skissert ovenfor, er ulike på flere punkter, men det er også likheter mellom dem. I tabell 2–1 er det listet opp forskjellige parametere som beskriver de enkelte løsningene.

Tabell 2–1: Sammenligning av personaliseringstjenester

Tjeneste	MyLibrary@ NCState	MittBibliotek ved UBiT	MyLibary@ Cornell	PIE	Greenstone
Personlig tilpasset innhold.	n	n	n	j	j
Automatisk tilpasset innhold	j	j	j	n	n
Egne lenker	j	j	j	n	n
Tilpasset utvalg av søkekilder	j	j	j	n	n
Tilpasning av søkekilder	n	n	n	j	j
Personlig metadata	n	n	n	n	j

Tabell 2–1: Sammenligning av personaliseringstjenester

Tjeneste	MyLibrary@ NCState	MittBibliotek ved UBiT	MyLibary@ Cornell	PIE	Greenstone
Egne samlin-ger	n	n	n	j	j
Integrert grensesnitt	j	j	j	n	j

Personlig tilpasset innhold er det ikke så mange som tilbyr. De tradisjonelle "Mitt bibliotek"-løsningene lar ikke brukeren selv velge innhold, men lar heller brukeren sette seg selv i en kategori, og så velger systemet innhold etter brukerens kategori. Dette er annerledes for PIE og Greenstone, der brukeren selv må velge innhold. "Mitt bibliotek"-løsningene lar brukeren legge til lenker i sitt system. Dette er de alene om å tilby.

Mitt Bibliotek-løsningene tilbyr stort sett et predefinert utvalg søkemotorer tilpasset den ovenfor nevnte brukerprofilen. Med PIE og Greenstone kan man selv definere hvilke søkemotorer/søkekilder som skal inngå i søket.

Personlig metadata kan man legge inn i Greenstone, men ikke i noen av de andre systemene, og egne samlinger kan man bare legge inn i Greenstone og PIE. Et grensesnitt for å bruke tjenesten mangler derimot i PIE i motsetning til i de andre systemene.

2.2 Søking

Målet med å personalisere bibliotekløsningene er å hindre informasjonsoversvømmelsen hos brukerne. For å klare dette er det viktig å kjenne brukernes atferd i søkesystemer. Det er gjort mye arbeid på området, og noe av arbeidet belyses videre i dette kapittelet.

Søking kan beskrives på mange nivåer. Strategier og taktikk omhandler overordnede metoder for å finne informasjon. Selve søkeprosessen omhandler mer atomær atferd fra brukerens side. Videre i dette kapittelet vil noen teorier om søkeprosesser og søker-atferd presenteres. Først presenteres Marchioninis modell for informasjonssøker-atferd i elektroniske ressurser[36] , deretter Ellis & Haugans modell for informasjonssøker-atferd[14] , og til slutt Choo et. al. modell for informasjonssøker-atferd på Internett[8] .

2.3 Marchioninis søkeprosess

Gary Marchioninis søkeprossess er en ofte referert teori[4]. Marchionini beskriver søkeprosessen som en kognitiv prosess der han fokuserer på rasjonelle valg

og begrunnelser brukeren gjør når han befinner seg i søkeprosessen. Modellen tar likevel høyde for en situasjon der man også foretar ikke-rasjonelle valg.

Bakgrunn for Marchioninis søkeprosess er at brukeren har et informasjonsbehov. Dette setter i gang selve prosessen. Et informasjonsbehov er forskjellen mellom det brukeren vil vite og det brukeren allerede vet.

Marchionini deler modellen inn i de faktorer som er av betydning for informasjonssøkeprosessen, og de delprosesser som utgjør selve søkeprosessen. Faktorene påvirker hvordan en informasjonssøker utfører prosessen og de resultater han oppnår. Først presenteres en forklaring av disse faktorene, og deretter en forklaring av søkeprosessen og de delprosessene den er satt sammen av.

2.3.1 Faktorer

Søkeprosessen som Marchionini beskriver, består av flere faktorer som spiller sammen.

Den første faktoren er **informasjonssøkeren** selv. Det er denne personen som har et informasjonsbehov og som må definere dette. Informasjonssøkerens bilde av den informasjonen han trenger, og den verden denne informasjonen finnes i, spiller en stor rolle for hvordan søkeprosessen går. Marchionini definerer fire typer informasjonsbehov, fra et følelsesmessig nivå til et konkret nivå. Marchionini definerer også to hovedtyper informasjonssøkere. Disse er domeneeksperten som har god kjennskap til hva han trenger, mens den andre, megleren, fokuserer på spørringer og informasjonsstrukturer.

Oppgaven er en realisering av informasjonssøkerens problem. Det er oppgaven som driver søkeprosessen framover, og den består av å uttrykke problemet og interagere med søkesystemet til problemet er løst. Uttrykket og artikulasjonen av problemet er vanligvis i form av et spørsmål, og interaksjonen med søkesystemet er både mental og fysisk. I tillegg må søkeren reflektere over resultatene. I utgangspunktet starter det med at brukeren har et mål for øye, men etter hvert som søkeprosessen skrider fram, kan planene endres. Mer om dette kommer senere, når selve prosessen beskrives.

Søkesystemet er kilden som representerer kunnskap, og som tilbyr verktøy og regler for tilgang og bruk av den kunnskapen [36] . Systemet har et grensesnitt som er en representasjon av kunnskapen og verktøy samt regler og mekanismer for tilgang og manipulering av denne kunnskapen. Det finnes konseptuelle og fysiske komponenter av kunnskapen og grensesnittet. Innholdets form spiller inn på hvordan man kan angripe kunnskapen. Den fysiske komponenten definerer hvordan man kan spørre etter informasjon.

Domenet representerer kunnskapsområdet eller fagfeltet (f. eks. informatikk eller psykologi). Det varierer i kompleksitet og påvirker informasjonssøkings-

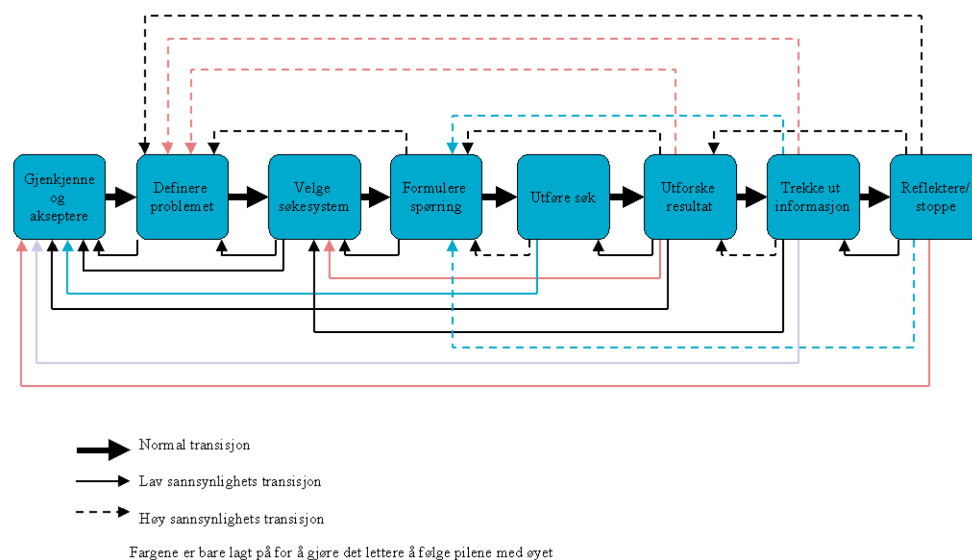
4. Sitert over 140 ganger, jfr ISI Web Of Science http://isiknowledge.com

prosessen og delprosessene. Domenet avgjør i stor grad hvilken database man må velge for å få dekket informasjonsbehovet.

Rammefaktorer angir rammene som begrenser en søkeprosess på ulike måter. De fysiske rammene legger begrensninger i form av tiden man kan bruke på et søk, og fysisk tilgjengelighet av materialet. De konseptuelle rammene handler om psykologiske og sosiale faktorer som påvirker søkeren i både positiv og negativ retning. Psykologiske faktorer kan være kognitive evner etc. Sosiale faktorer kan være sosial status og sosial interaksjon.

Søkeresultater er ikke bare selve resultatet av søk, men også prosessen som leder til resultatene. Søkeresultatene er selvsagt viktige, men selve prosessen fører også med seg resultater, og man lærer ting man tar med seg videre i prosessen. Man kan evaluere systemet man søker i, og seg selv, og dette kan man bruke for å forbedre søkeprosessen.

2.3.2 *Søkeprosessen*



Figur 2–4: Informasjonsøkeprosessen etter Marchionini [36]

Marchionini definerer en rekke prosesser. Disse beskrives i detalj i Appendix A på side 113. Her følger et sammendrag av søkeprosessen som Marchionini beskriver den[36] .

Første skritt i søkeprosessen er å innrømme og **akseptere** at man har et informasjonsproblem. Dette kan være internt eller eksternt motivert. Informasjonssøkeren finner ut at han har en mangel på informasjon.

Etter at søkeren har innsett en mangel på informasjon, må han **definere og forstå problemet**. Denne prosessen er aktiv så lenge søkeprosessen pågår, og det er en stor mulighet for å komme tilbake til denne prosessen. Denne prosessen blir ofte tatt for lett på av sluttbrukere, og dette fører til frustrasjon og feil [36] . Definisjonen av problemet må begrense det og klassifisere det, og i tillegg må man bestemme seg for en form på svaret; om man vil ha svaret som fakta eller som økt forståelse.

Når man **velger seg et søkesystem**, gjør man dette på bakgrunn av den erfaringen man tidligere har i domenet og ut fra tidligere erfaring med problemløsning generelt. Domenekunnskap setter sterke føringer for hvilket søkesystem man velger. Informasjonssøkere tilpasser søkeoppgaven til ett eller flere søkesystemer.

Prosessen med å **formulere et søk** handler om å tilpasse søkerens forståelse av problemet til det søkesystemet man valgte i forrige fase.

Utføringen av søket drives av søkerens mentale modell av søkesystemet, og er basert på formuleringen av spørringen som ble utviklet i forrige fase.

For at søkeren skal kunne måle framgang mot avslutning av søkeprosessen, må **søkeresultatene undersøkes**. Problemet og søkerens personlige informasjonsinfrastruktur bestemmer søkerens forventninger til resultatet og antallet treff, og kvaliteten på disse, som er nødvendig for å avslutte søkeprosessen. Resultatenes relevans må bedømmes i forhold til den definerte oppgaven.

Hvis søkeren finner et relevant dokument, kan han fortsette bearbeiding av det og **trekke ut mer informasjon**, eller vente med videre bearbeiding og forsette undersøkelsen av resultatsettet. Når informasjon er trukket ut, vil denne informasjonen legges til i søkerens kunnskap om domenet, og dermed endre grunnlaget for søkeprosessen. (Se «Informasjonssøker» på side 113.)

For å oppnå ønsket resultat, trenger man ofte flere iterasjoner i denne prosessen. Det første resultatsettet fungerer ofte som en form for input til formulering av videre spørringer og søk. Hvis man skal gå en runde til i "sirkelen", kreves det en vurdering av selve søkeprosessen og om denne svarte til forventningene. Spørsmålet om man skal stoppe eller ikke, kan avhenge av flere ting, som søkesystemet, rammene, oppgaven, domene-kunnskapen eller informasjonssøkingsevnen.

2.4 Informasjonssøker-atferd

David Ellis og Merethe Haugan[13] [14] har gjort undersøkelser av forskere og ingeniører ved Statoils forskningssenter i Trondheim. Basert på funnene de gjorde der, har de kommet fram til en modell for hvordan informasjonssøkere opptrer når de trenger informasjon i arbeidet sitt. Før selve modellen forklares, presenteres grunnlaget for modellen.

Grunnlaget for modellen ble lagt ved å undersøke arbeidsmetodene og organisasjonen i Statoil. Ellis og Haugan definerte tre typer prosjekter som alle hadde forskjellige informasjonsbehov.

- *Utviklingsprosjekter* ("incremental"-prosjekter) har som mål å utvikle små teknologiske framskritt, basert på eksisterende kunnskap. Disse er karakterisert ved lav risiko, lav gevinst, og kort tid (1-2 år). Informasjonsbehovet spenner fra bred generell kunnskap om området av interesse, til spesiell kunnskap om problemet som skal løses. Foretrukne informasjonskilder er personlige kontakter, egen kunnskap, og til slutt biblioteket.
- *Radikale prosjekter* baserer seg på eksisterende kunnskap som ikke regnes som god nok for å oppnå ønsket resultat. Derfor er disse karakterisert ved høyere risiko, høyere gevinst, og lengre tidsperspektiv (2-5 år). Informasjonsbehovet er generelt bredt, men faktaopplysninger trengs også. Informasjonskildene er først og fremst egen kunnskap, deretter personlige kontakter, og til slutt biblioteket. Det brukes en del publisert litteratur for å dekke informasjonsbehovet.
- *Grunnforskningsprosjekter* har som mål å øke organisasjonens kunnskap om et område, og har høy risiko og usikre gevinster for bedriften. Tidsperspektivet er langt (5-10 år).

Ellis og Haugan har også funnet ut at ulike faser av de overnevnte prosjektene har ulike informasjonsbehov. De har identifisert tre faser som er av interesse i informasjonssøkingen: evaluering av alternative løsninger, utvikling og testing, og til slutt oppsummering av erfaringer.

Evaluering av alternative løsninger har i all hovedsak to aktiviteter som utpeker seg med informasjonssøking. Den første, *ide-generering*, har et varierende behov for å sjekke mange nye ideer. Den andre aktiviteten, *undersøkelser,* tar så for seg noen av disse ideene og undersøker dem videre og dypere.

Utvikling og testingsfasen har ikke et så stort behov for ny informasjon, men den informasjonen man har, blir brukt i *utvikling* og *testing*. Av og til må man i tillegg sjekke egne funn mot nyere litteratur.

Oppsummeringsfasen tar sikte på å dele informasjonen og kunnskapen fra et prosjekt slik at det kan bli brukt i framtiden. Dette er karakterisert ved rapportskriving og retrospektive søk i bibliografisk materiale på jakt etter f.eks. andre løsninger.

2.4.1 Informasjonssøkingsmodellen

Forundersøkelser (Surveying)

Forundersøkelsen tar sikte på å danne et bilde av litteraturen innen et emne, eller å finne nøkkelpersoner innen et felt. Foretrukne informasjonskilder er bibliografiske søk og uformelle personlige nettverk. Til å begynne med er det foretrukket å bruke personlige kontakter for å finne seg et startpunkt for videre informasjonssøking.

Kjeding (Chaining)
Kjeding er prosesser der en informasjonssøker bruker referanser til å finne nytt materiale. Det vanligste her er å finne referanser i artikler og følge disse videre. Denne prosessen stopper som oftest opp fordi man ikke har tid til å lete mer. Det snakkes om to typer kjeding. Kjeding bakover foregår når man følger pekere fra et utgangspunkt, mens kjeding forover foregår når man finner ressurser som peker til utgangspunktet. Sistnevnte er ikke så mye brukt.[8]

Overvåking (Monitoring)
Overvåking er en prosess der man holder seg oppdatert på et fagfelt. Dette gjøres ved både uformelle metoder som kontakt med personer som kan ha faglig interesse, og formelle metoder som tidsskrift, proceedings og søkeagenter.

Skumming (Browsing)
Skumming er en prosess der man tar for seg dokumenter av interesse, med sikte på å identifisere viktig informasjon. Dette kan f.eks. innebære å bla gjennom tidsskrift for å se om det er noe interessant i dem. Man bruker her innholdsfortegnelser, tittellister, sammendrag etc. for lettere å finne fram til det man trenger.

Utskilling (Differentiating)
For å vurdere viktigheten av informasjonskilder, og rangere dem, brukes utskilling. Dette innebærer at informasjonssøkere bruker sin egen erfaring som utgangspunkt. Uformelle informasjonskilder blir prioritert i dette arbeidet.

Filtrering (Filtering)
Filtrering tas i bruk for å gjøre informasjonen så relevant og presis som mulig. Dette gjør også at informasjonssøkeren må være oppmerksom slik at viktig informasjon ikke oversees. Den vanligste måten å gjøre dette på, er å bruke elektroniske litteraturdatabaser. Når kolleger videreformidler noe de tror kan være interessant for andre, kalles det usymmetrisk filtrering.

Utvinning (Extracting)
Utvinning er prosessen der man systematisk gjennomgår kilder for å finne noe av interesse. Dette er en vesentlig del av informasjonssøkingen.

Avslutning (Ending)
Dette er en prosess der informasjonssøkeren utfører mindre søk for å komplettere teori til arbeidet. Dette kan være f.eks. å få svar på mindre, åpne spørsmål, eller å finne nypublisert materiale som kan være av interesse. Dette skjer ofte i slutten av et prosjekt der man kan være interessert i å finne ut om det har skjedd noe nytt siden man startet på prosjektet.

Som det går fram av modellen over, er ikke dette en helhetlig modell for atferdsbeskrivelse. Denne modellen beskriver forskjellige prosesser i forbindelse med et informasjonssøk eller en informasjonsbearbeidelsesprosess.

2.5 Informasjonssøker-atferd på Internett

Chun Wei Choo og kolleger[7] [8] har basert sitt arbeid på Marchioninis modell for browsing (skumming) og Ellis & Haugans modell for informasjonssøker-atferd nevnt i Kapittel 2.4. De definerer fire typer overordnede skummingsmetoder som de har kalt betraktninger (viewing) og søk. De fire metodene er vist i Tabell 2–2 og nevnt under:

Tabell 2–2: Skummingsmetoder fra Choo et. al.[8]

Scanning Modes	Information Need	Information Seeking	Information Use
Undirected Viewing	General areas of interest; specific need to be revealed	"Sweeping" Scan broadly a diversity of sources, taking advantage of what's easily accessible	"Browsing" Serendipitous discovery
Conditioned Viewing	Able to recognize topics of interest	"Discriminating" Browse in pre-selected sources on pre-specified topics of interest	"Learning" Increase knowledge about topics of interest
Informal Search	Able to formulate simple queries	"Satisfying" Search is focused on area or topic, but a good-enough search is satisfactory	"Selecting" Increase knowledge on area within narrow boundaries
Formal Search	Able to specify targets in detail	"Optimizing" Systematic gathering of information about an entity, following some method or procedure	"Retrieving" Formal use of information for decision-, policy-making

- Planløs betraktning (Undirected Viewing):

  I planløs betraktning har ikke søkeren noen klare informasjonsbehov. Målet er å finne endringer raskt, og man betrakter generelle informasjonsområder. Her bruker søkeren mange forskjellige kilder, noe som fører til at han finner ting han ikke søker etter, men som kan være nyttige senere.

- Betinget betraktning (Conditional Viewing):

  I betinget betraktning avgrenser søkeren interessen til et område eller til spesiell informasjon. Hovedmålet er å evaluere viktigheten av denne informasjonen i forhold til innvirkning på arbeidet.

- Uformelt søk:

  I det uformelle søket har søkeren som mål å høyne forståelsen om et tema, og ser derfor aktivt etter informasjon. Han legger likevel liten innsats i søkingen, og den er ofte ustrukturert.

- Formelt Søk

  Søkeren har ett enkelt mål, og gjør et målrettet søk etter informasjon om dette spesielle emnet. Han strukturerer søket i henhold til en metode eller lignende, og bruker det han finner som basis for en avgjørelse.

Choo et. al. har tatt utgangspunkt i de fire overnevnte skanningsmetodene og Ellis' modell for informasjonssøker-atferd. Ved å undersøke atferden til en utvalgt gruppe frivillige i to uker, har de kommet fram til et sett med atferder som så er blitt sammenlignet med Ellis' modell og de fire typene betraktning/søking som er vist i tabellen over. Dette har de samlet i en enhetlig modell for informasjonssøker-atferd på Internett som vist i Tabell 2–3. I fasen *"planløs*

Tabell 2–3: Choos modell for søking i nett ressurser

	Starting	**Chaining**	**Browsing**	**Differentiating**	**Monitoring**	**Extracting**
Undirected Viewing	Identifying, selecting, starting pages and sites	Following links on initial pages				
Conditioned Viewing			Browsing entry pages, headings, site maps	Bookmarking, printing, copying; Going directly to known site	Revisiting 'favourite' or bookmarked sites for new information	
Informal Search				Bookmarking, printing, copying; Going directly to known site	Revisiting 'favourite' or bookmarked sites for new information	Using (local) search engines to extract information
Formal Search					Revisiting 'favourite' or bookmarked sites for new information	Using search engines to extract information

betraktning" har de funnet atferd som passer overens med Ellis' "forundersøkelse" (av Choo kalt Starting) og "kjeding". Dette innebærer å identifisere start-

steder og startsider på Internett for så å følge lenker ut fra disse sidene. *"Betinget betraktning"* har elementer fra "Skumming", "Utskilling" og "Overvåking". Skummingen foregår ved at informasjonssøkerne skummer gjennom inngangssider, overskrifter og nettstedsskart. Utskillingen dreier seg om å ta vare på bokmerker, skrive ut og kopiere sider, og å bruke allerede kjente nettsider. Overvåking gjøres ved at informasjonssøkerne bruker tidligere lagrede bokmerker. *"Uformelle søk"* foregår ved at søkeren i tillegg til utskilling og overvåking bruker "utvinning". Dette gjøres ved at søkeren bruker søkemotorer for å finne informasjon. *"Formelle søk"* benytter seg av de to prosessene "Overvåking" og "Utvinning". Disse er de samme som under "Uformelle søk".

2.6 Søkemodeller: Oppsummering

I dette kapitlet er det vist flere modeller for informasjonssøking. Den første var Marchioninis modell for informasjonssøker-atferd. Marchionini definerer først noen faktorer som påvirker en informasjonssøker, og deretter beskriver han hvordan en informasjonssøkingsprosess foregår i detalj. Han trekker fram en del aspekter som kan virke innlysende, men som er viktig å tenke på når man skal forsøke å modellere informasjonssøkeres atferd, f.eks. leseferdigheter.

Modellen som Ellis og Haugan trekker fram, baserer seg på observert atferd til informasjonssøkere i en prosjektgruppe. De finner forskjellige informasjonsbehov og forskjellige informasjonskilder til forskjellige faser i arbeidet. Basert på sine undersøkelser lager de en modell for hvordan man bruker informasjonskilder i arbeidet.

Choo et. al. baserer mye av sitt arbeid på Ellis og Haugans modell, men de avgrenser modellen til informasjonssøking på Internett. De tar tak i prosessene som Ellis og Haugan finner fram til, og overfører dem til søking i nettressurser. De definerer fire typer søk som er å se på som grupper av Ellis og Haugans prosesser.

Disse modellene som er nevnt over, beskriver hver for seg forskjellige aspekter av søkeprosessen. Choo et. al. beskriver ting som foregår på et høyere kognitivt nivå enn det Marchionini beskriver. Ellis og Haugans modell beskriver søkeprosesser mer detaljert enn Choo et. al., men beskriver ikke samme type prosesser som Marchionini.

3 SØKESCENARIER

For å forstå søkeprosessene bedre og ha et grunnlag for å forbedre søkeopplevelsen, er det blitt gjort undersøkelser i forbindelse med søking. Dette kapittelet tar sikte på å beskrive noen søkeprosesser og systemene disse har foregått i.

Når man skal lage et system som tar sikte på å forbedre en søkeprosess eller endre på den, er det naturlig å sette seg inn i denne prosessen, bl.a. ved hjelp av praktisk erfaring fra søkeprosesser. Praktisk erfaring utfyller den teoretiske kunnskapen om emnet, og kan gi gode innspill og ideer når man senere skal lage kravspesifikasjoner til et nytt system.

En informasjonssøkeprosess vil nå bli undersøkt og gjennomgått i forhold til Marchioninis faktorer og søkeprosesser. Den søkeprosessen som er beskrevet nedenfor, er en del av informasjonssøkeprosessen utført av forfatteren i begynnelsen av dette arbeidet. Til å begynne med vil noen av faktorene i søkeprosessen bli beskrevet. Deretter vil selve søkeprosessen beskrives.

3.1 Faktorer

3.1.1 Informasjonssøkeren

For å kunne trekke slutninger fra beskrivelsen av søkeprosessene, er det viktig å beskrive hvem det er som søker. Med dette menes attributter som kan være av interesse for hvordan denne personen utfører søk etter informasjon. For å beskrive informasjonssøkeren er det tatt utgangspunkt i Marchioninis [36] beskrivelse av denne. Mange av attributtene Marchionini refererer til, er vanskelig å kvantifisere uten ekspertise, så det er ikke gjort i særlig stor grad.

Kort fortalt er informasjonssøkeren en student som er på jakt etter informasjon i forbindelse med et studentarbeid. Han kan ikke sies å være en megler i Marchioninis definisjon, men heller ikke fullt ut en domeneekspert. Han kan også nyttiggjøre seg en pc, og behersker flere verktøy for å skrive og lese både på pc

og uten. Informasjonsbehovet ligger nærmere det beviste enn det formaliserte, og dreier seg om å øke kunnskapen heller enn å få presise faktaopplysninger.

3.1.2 Oppgaven

Oppgaven er å finne en samling på 15 informasjonsobjekter som er relevante for denne avhandlingen. Dette angir kvantitetsmålet på oppgaven. Målet med oppgaven er å øke søkerens kunnskaper innen oppgavens tema. Denne kunnskapen skulle gi grunnlag for å starte arbeidet med denne avhandlingen. Temaet for oppgaven er: Personlige samlinger, Personlig katalogisering, Søkeprosesser, Digitale bibliotek. Temaet for søket er ikke spesifisert spesielt nøyaktig, og det kan gi mye støy i søket. Grunnen til at man ikke ønsker å spesifiserer søket mer, er at søkeren ikke er godt kjent i informasjonsdomenet, og en for snever spesifisering vil føre til at man kan miste mange nyttige søkeresultater. (Se «Informasjonssøker-atferd» på side 37.) Tidsavgrensingen i oppgaven er maksimalt 14 dager.

I tillegg til å skaffe informasjon tjente søkeprosessen også et annet mål. Dette var å bli kjent med en del søkesystemer og på bakgrunn av det kunne ta avgjørelser om utforming av slike. I tillegg er beskrivelsen av scenariene viktig for denne avhandlingen.

3.1.3 Aktuelle søkesystemer

Det er flere aktuelle søkesystemer for denne søkeoppgaven, men det blir sett bort fra andre søkesystemer enn de elektroniske. Bakgrunnen for dette er at det er de elektroniske systemene som er gjenstand for diskusjon i denne avhandlingen.

Det søkesystemet som er mest nærliggende å starte i, er BIBSYS. Dette er bibliotekkatalogen og biblioteksystemet for over 100 universitets-, høyskole- og forskningsbibliotek, og biblioteksystemet ved forfatterens institusjon. Man kan si at dette er en av de mer nærliggende kildene til “kvalitetssikret” informasjon for studenter ved NTNU. Ulempen med BIBSYS-basen er at den har lite innhold av artikler. Mye av det som publiseres i fagfeltet eller domenet som er aktuelt for dette scenariet, utgis som artikler, og dette gjør at det er fornuftig å finne journaler eller nettsteder som refererer til relevante informasjonsobjekter.

BIBSYS har[5] også en søketjeneste som tilbyr artikler fra Institute for Scientific Information (ISI) gjennom tjenesten ISI-Søk. Andre aktuelle steder for informasjon om emnet er D-Lib Magazine (www.dlib.org), Association of Computing Machinery (www.acm.org), og Council of Library and Information Resources (www.clir.org). Søkemotorer som Google (www.goggle.com) og AllTheWeb (www.alltheweb.com) kan også brukes for å finne andre publiserte og upubliserte artikler. Generelle søkemotorer kan også brukes for å finne andre nettsteder som omhandler og/eller publiserer materiale som er relevant.

5. Fra 2. april 2002 har BIBSYS sluttet å drifte ISI-basen for det norske fag og forskningsmiljøet. Disse basene er nå tilgjengelig fra http://isiknowledge.com.

3.2 Søkeprosessen

Ut fra ønsket om å undersøke selve søkeprosessen er det naturlig å stille spørsmål som:

- Hvilke typer data ser man etter?
- Hvordan oppfører man seg?
- Hva tar man vare på av informasjon underveis (typer data)?

Dette kan gi indikasjoner på hvordan søkeren "beveger" seg i søkeprosessen, og de valg som ligger til grunn for bevegelsene.

Når det er snakk om innhenting av krav til en pilot av en personlig samlingstjeneste, er det naturlig å vinkle spørsmålene litt annerledes. Da blir det mer naturlig å se på hva søkeren gjør og ser etter i detalj. Noen slike ting kan være:

- Hvor klikker man på skjermen?
- Hvilke funksjoner bruker man?
- Hvordan ser poster ut?
- Hvilken informasjon må man ta vare på, slik at man kan vite hva man har gjort?
- Hvordan kan man ta vare på informasjon om søk underveis i prosessen?

Til å begynne med i dette kapittelet vil noen nett-tjenester, som ikke er spesifikke søketjenester, bli forklart. Deretter vil noen søkesystemer presenteres, og

Tabell 3–1: Oversikt over søkesystemer

Søkesystem	Beskrivelse
BibSøk Nett	Søketjenesten i bibliotekkatalogen til bibliotekene tilknyttet BIBSYS.
BIBSYS ISI-Søk[a]	Søketjeneste for å søke i Institute for Scientific Information's databaser.
D-Lib Magazine	Søketjeneste for å søke i artiklene som har vært publisert i det elektroniske tidsskriftet D-Lib Magazine.
ACM Digital Library	Søketjeneste for å søke i alle tidsskrifter som utgis av Association for Computing Machinery.
Council of Library and Information Resources	Oversikt over artikler utgitt av CLIR.
WebSpirs	Søketjenester for å søke i tidsskriftene til OVID/SilverPlatter
NewFirstSearch	Søketjeneste for å søke i tidsskriftene som utgis/formidles av OCLC

Tabell 3–1: Oversikt over søkesystemer

Søkesystem	Beskrivelse
Aperitif.no	Nettsted for saker som omhandler kokk- og servitør-faget. Inneholder bl.a. søkbare oppskrifter.
Amazon.com	Nettbokhandel med bl.a. søketjenester for gjenfinning av bøker.

a. ISI-Søk er fra 2/4-2002 ikke lenger tilgjengelig. Samme data er tilgjengelig gjennom tjenesten WebOfScience fra ISI. Dette er likevel tatt med som en del av avhandlingen da det gir innsikt i søkeatferden.

etter dette vil noen aktuelle søkesystemer bli sett på i mer detalj. Disse vil bli forklart i forhold til Marchioninis rammeverk med et ønske om å hente inn krav til en personlig samlingstjenste. Kravene til en personlig samlingsmodul er tema for et senere kapittel.

3.2.1 *Noen andre systemer*

Da det ikke bare er bibliografiske søkesystemer som kan være til hjelp for søkere, følger her en liten presentasjon av andre systemer som kan være av interesse.

Aperitif.no

Aperitif.no er et nettsted fra Aperitif AS som bl.a. utgir et fagblad for vin-kelnere. Nettstedet er stort innenfor mat og drikke (vin, drinker). Hoveddelene av aperitif.no er pol-listen, kokebok og bartender. Disse seksjonene inneholder forskjellige oppskrifter, tips, tester og nyheter. Aperitif tilbyr for tiden tre tjenester: Min kokebok, Min vinkjeller, og Min Bartender. Dette er muligheter for å lagre bokmerker til oppskrifter, viner og drinker. En nærmere forklaring av kokebok-systemet følger.

Systemet for å lagre oppskrifter i en egen profil fungerer som forklart her. Først må man registrere seg og deretter logge inn. Når man er innlogget, kan man søke fram oppskrifter fra databasen til aperitif. Denne basen inneholder en del oppskrifter som er lagt inn av Aperitif, men også mange som er lagt inn av andre privatpersoner og institusjoner. Man kan også legge inn egne oppskrifter i sin profil. Disse oppskriftene kan enten lagres som private eller offentlige. Her er det på sin plass å navne at oppskriftene er søkbare også uten at man logger seg inn i ”Min Kokebok”.

Oppskriftene i Aperitif er organisert i ferdig definerte kategorier. Disse kategoriene er omtrent de samme som man finner i en vanlig kokebok: kjøtt, fisk, bakervare, grønnsaker, etc.

I Min Kokebok har alle oppskrifter en opphavsmann. Det vil si at man ser hvem som har lagt inn oppskriften, men det er ikke nødvendigvis denne personen som har funnet opp retten. Selv om søkemulighetene er begrenset, er de i hvert fall til stede. Dette er til stor hjelp fordi det er for mange oppskrifter i databasen

til at det er hensiktsmessing å bla i dem. Man kan ikke angi hvilke felter man vil søke i, men man kan avgrense til seksjon og tidsrom.

Det positive med dette systemet er at man ser hvem som har lagt inn oppskriften. Dette gjør at oppskriften får en helt annen autoritet. Ser man f.eks. at en mesterkokk har lagt inn en oppskrift, kan man gå ut fra at oppskriften er god og verdt å prøve selv. Oppskrifter lagt inn av Opplysningskontoret for Kjøtt kan også antas å være sikre å prøve hjemme. Informasjon om kun ingredienser gjør en ikke i stand til å ta denne typen avgjørelser, med mindre man har stor kjennskap til kokekunsten.

Amazon.com

Nettbokhandelen Amazon.com har kommet ganske langt i å personalisere brukernes handleopplevelse. De tilbyr gode søkeredskaper for brukere som vet å bruke disse, men også noen meget enkle redskaper som er lettere å bruke. Det er viktig å påpeke at de ikke har fokus på søking og informasjonsgjenfinning slik man finner i biblioteksystemene.

Brukerprofilen i Amazon.com inneholder navn og e-postadresse, samt store muligheter for å spesifisere interesser. Dette innebærer alt fra foretrukne typer butikker til favorittmerker på produkter, favorittforfattere, favorittfilm etc.

Den personlige siden viser til enhver tid produkter fra de kategoriene man har valgt. Utvalget på menyer etc. er også begrenset på bakgrunn av valgte preferanser. Dette gjør informasjonsmengden mer håndterlig.

Søkeredskapene er altså gode, men Amazon.com har ikke prioritert å vise metadata på samme måte som et bibliotek. Dette skyldes nok at brukerfokuset er ganske anderledes hos Amazon.com. De ønsker naturligvis at brukere skal handle i bokhandelen, og forteller om pris og frakt og andre tilbud knyttet til den aktuelle boka. Dette kan selvsagt gi en god pekepinn på andre ting som kan være av interesse. Anbefalinger som “Brukere som kjøpte denne, kjøpte også” kan være nyttige om man ønsker å orientere seg i et nytt domene. Finner man f.eks. fram til boka “Surely you’re joking Mr. Feynman” kan man se at Amazon.com (eller rettere sagt andre brukere gjennom deres kjøp) anbefaler bl.a. Six Easy Pieces. Dette kan være av interesse for brukere som er interessert i den første boka.

En siste finesse som er verdt å nevne med Amazon.com, er leseranmeldelsene. Dette lar en bruker registrere bokanmeldelser som deretter blir gjort tilgjengelig for andre brukere. Bokanmeldelsene er ikke søkbare, men kun tilgjengelige fra boken de gjelder. Det er også mulig å søke etter brukere, og se alle anmeldelsene en bruker har lagt inn.

3.2.2 Noen søkesystemer som ikke er undersøkt i detalj

Det er mange søkesystemer man kan bruke for å søke etter kvalitetssikret informasjon på nettet. Mange av disse tilbyr dokumenter innen en avgrenset del av et domene. Man må derfor ofte kombinere søk i flere systemer for å dekke et domene fullstendig. Det har ikke vært mulig å undersøke alle slike systemer i

denne sammenheng, men en liten oversikt over noen av de som er brukt, men som ikke er mye omtalt i denne avhandlingen, følger nedenfor.

WebSpirs
Denne tjenesten tilbyr søk i flere databaser i mange fagfelt. Den inkluderer Inspec-basene for fysikk, elektronikk, datateknikk etc., MedLine-basene fra National Library of Medicine, PsychInfo-basene om psykologi, Biological Abstracts om biologi, og mange flere.

NewFirstSearch
NewfirstSearch fra OCLC tilbyr en hel del databaser med forskjellig innhold. Disse databasene er tilgjengelig i et enhetlig grensesnitt med kobling mot beholdning for BIBSYS-bibliotekene for norske brukere.

3.3 BibSøk Nett

BibSøk Nett er søketjenesten mot bibliotekbasen til BIBSYS. Denne basen inneholder mer eller mindre all katalogisert metadata fra de over 100 bibliotekene. Denne bibliotekbasen inneholder lite artikler, men selv om dette er i ferd med å endres, er basen mindre vesentlig for informasjonssøket enn mange av de andre søkesystemene jeg har sett på. Likevel er det fullt mulig å finne noe som kunne være av interesse. Først beskrives prosessen som ble utført, deretter undersøkes de funn som ble gjort.

Bibliotekbasen i BIBSYS er tilgjengelig fra www.bibsys.no. Søkegrensesnittet er nevnt i Kapittel 3.9.1, og vil ikke bli forklart nærmere her.

Som utgangspunkt for søket ble søketermen “digital library” som fritekst valgt. Dette skulle gi et bredt anlagt søk, og en mulighet til å få en liten oversikt over domenet i denne kilden. Antallet treff ble 33, som er en høyst overkommelig mengde, og det byr ikke på problemer å lete gjennom disse med mindre man har dårlig tid. Det andre søket i dette systemet var etter ”electronic library”. Dette resulterte i 61 treff, en noe større mengde å lete gjennom, men fortsatt akseptabel.

Disse resultatene indikerer at det er flere poster i denne databasen under en annen term enn først antatt, og dette understreker viktigheten av å ikke avgrense søket for mye til å begynne med. Dette er nyttig å ta med seg i videre søk, og viser at prosessen har et verdifullt resultat utenom selve søkeresultatene. Et annet aspekt ved de to ulike søketermene er at de indikerer en forskjell i registreringen. Langt flere av postene under termen “digital library” er av nyere dato enn de under “electronic library”. Dette kan bety at informasjonsobjekter som i dag ville blitt registrert for treff med termene “digital library”, tidligere ble registrert med tanke på termene “electronic library”. Når det er sagt, må det sies at det eksisterer nyere poster under termen “electronic library” også, så det er et faktisk skille mellom disse begrepene.

En annen ulempe med BibSøk Nett er at digitale dokumenter ikke er tilgjengelige. Det finnes noen ressurser som har nettadresse, men dette gjelder de færreste. Hjelpen er at dokumenter kan bestilles, og hentes fra biblioteket.

3.4 BIBSYS ISI-Søk[6]

BIBSYS tilbyr søk i ISI-basen (Institute for Scientific Information) som inneholder artikler fra en rekke tidsskrifter, og som har en god kobling til kopibestilling og beholdningsdata fra BIBSYS-bibliotekene. Man har mulighet til å søke i artiklene eller etter bestemte tidsskrifter. I dette tilfellet ble det valgt å søke i artikler i ISI-basen da kjennskapet til hvilke tidsskrifter som inneholder rett informasjon ikke var god nok på dette tidspunktet i søkeprosessen. ISI-Søk (som BIBSYS kaller tjenesten) har ikke fulltekst-dokumenter tilgjengelig, men det er ganske enkelt å bestille kopier av artikler. Man må da hente dem på biblioteket.

For å kunne sammenligne systemene BibSøk Nett og ISI-Søk ble det søkt etter følgende uttrykk: ”digital library”. Dette ga 872 treff. Det er for mange treff å lese gjennom, så trefflisten ble avgrenset til de siste fem år, noe som resulterte i 404 treff. Da dette fortsatt ble ansett som for mye å bla gjennom, ble søket utvidet til alle år, men avgrenset ved å søke på “collection” i tillegg. Dette ga 63 treff, som er en overkommelig mengde.

Ut fra de 63 treffene ble det funnet flere artikler som ble ansett for å være av interesse.

Søkeprosessen i ISI-Søk er enkel å utføre, men søket krever en del domenekunnskap på grunn av de store datamengdene man søker i. Antallet treff for søket “digital library” forteller dette. Som søker må man selv avgrense søket, og dette krever at man har noe kunnskap om domenet for at relevante treff ikke skal gå tapt i filtreringsprosessen.

3.5 D-Lib Magazine

For å søke i D-Lib Magazine må man gå gjennom nettsiden deres på http://www.dlib.org. Der må man finne fram til søkegrensesnittet og skrive inn spørring. Søkemotoren kan søke på fraser eller boolske uttrykk. Boolske uttrykk gir flest muligheter, og blir derfor et naturlig valg. Etter å ha søkt på ”digital and collection” måtte termene not “clips and pointers” legges til for å fjerne en hel del støy. Søkeprosessen er ganske tungvint, og søkeverktøyet som ligger til grunn er ganske enkelt sammenlignet med andre bibliografiske systemer som BibSøk Nett og ISI-Søk. Det er ikke mulig å søke i metadataelementer, og man må derfor gjøre noen søk for å lære seg dette grensesnittet. Dessverre må man

6. ISI-Søk er fra 2/4-2002 ikke lenger tilgjengelig. Samme data er tilgjengelig gjennom tjenesten WebOfScience fra ISI. Dette er likevel tatt med som en del av avhandlingen da det gir innsikt i søkeatferden.

gå mange runder i Marchioninis søkeprosess for å få et søk som er tilfredsstillende. Alt i alt kan søkeprosessen i DLib oppsummeres som mange iterasjoner for å finne det riktige boolske uttrykket som igjen gir det riktige resultatsettet.

Når dette er sagt, er det mange artikler i DLib som er interessante, og i motsetning til de to overnevnte søkesystemene er de gratis tilgjengelig i fulltekst. Det er med andre ord meget lett å få tilgang til det man måtte finne.

3.6 ACM Digital Library

Association for Computing Machinery (ACM) tilbyr sitt eget grensesnitt mot alle publikasjoner de utgir. De tilbyr en stor samling artikler fra mange ulike tidsskrifter, magasiner, transactions og proceedings. Er man for slepphendt når man lager spørringer, får man fort for mange treff. Som så mange ganger tidligere i scenariene startet søket med et vidt søk for å undersøke hvor stort informasjonsgrunnlag det er snakk om i denne basen. Spørringen “digital library” gir 829 treff. For spørringen “digital AND library” blir antallet treff økt til om kring 8.797. Spørringen “electronic and library” gir nesten 5.000 treff. Det sier seg selv at spørringene må avgrenses, og man må være ganske spesifikk for å få et resultatsett man kan bla gjennom uten at det tar for lang tid.

Over en tidsperiode har det blitt søkt i ACM Digital Library, og resultatet er en hel del interessante artikler. Søkene har stort sett være frasesøk, og det er søkt på fraser som: digital library, digital library collection, electronic library, electronic library collection, digital library collection management, collection management, electronic library collection management, personal collection, etc.

Artiklene i ACM tilbys i fulltekst for de som betaler for denne tjenesten. I tillegg finnes det flere abonnementer som ikke gir full tilgang til alt, men en varierende begrenset del av samlingen. Biblioteket ved forfatterens institusjon (UBiT) har tilgang til en del artikler, og disse kan lastes ned som pdf-filer. Flere av artiklene som danner grunnlaget for denne avhandlingen, ble funnet i ACM Digital Library.

3.7 Council of Library and Information Resources

Ulikt de andre systemene har Council of Library and Information Resources (CLIR) ingen søkemuligheter mot sine publikasjoner. CLIR utgir flere publikasjoner, og tilbyr en liste over artikler de har publisert. Listen finner man fra http://www.clir.org. Denne listen kan man søke i med søkemuligheten i nettleseren man bruker. Artiklene har følgende metadata: tittel, forfatter, (publikasjons-) dato, artikkelnummer, og pris på papirkopi. Artiklene ligger også tilgjengelig i fulltekst på denne siden.

For å finne noe av interesse på disse sidene, er man nødt til å finne en tittel man synes høres interessant ut og klikke på den så man kan lese oppsummeringen

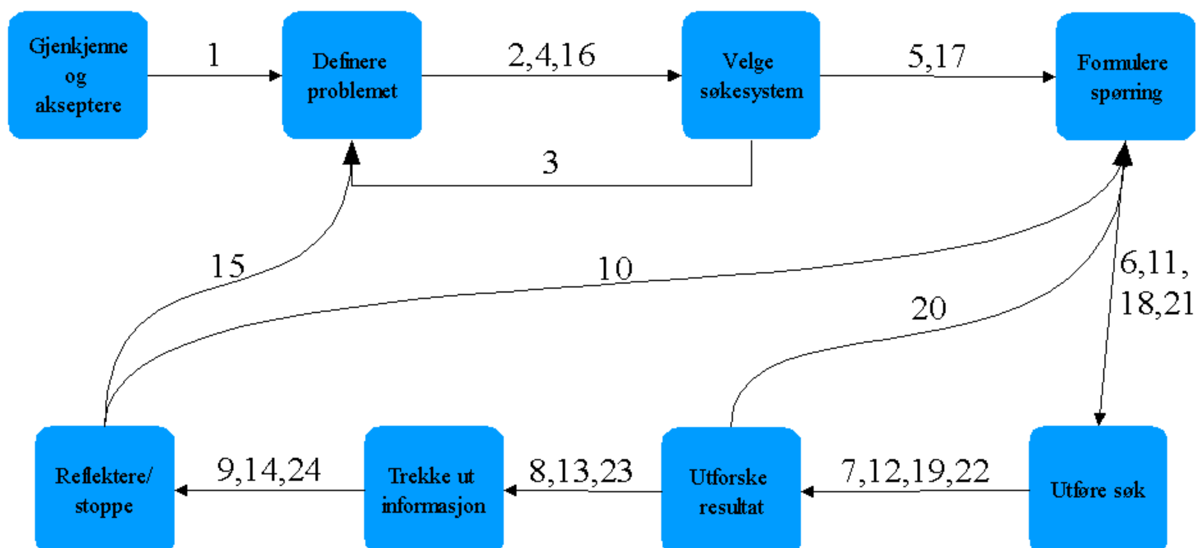
av artikkelen. Dette gjør at man går relativt fort gjennom selve søkeprosessen og dveler mer med informasjonsuthenting.

3.8 Scenarier i Marchioninis modell for søking

For å undersøke hvordan søkeprosessen som er gjennomført ser ut i Marchioninis rammeverk, er den søkeprosessen som er nevnt ovenfor, satt opp i form av delprosessene i Marchioninis informasjonssøkingsmodell. Ved å gjøre dette kan man finne ut hvilke deler av den faktiske prosessen som passer overens med Marchioninis delprosesser.

Prosessen startet med at søkeren hadde et informasjonsproblem. Han fant så fram til de søkesystemene som ble vurdert som mest naturlige. Til å begynne med var dette BibSøk Nett (biblioteksystemet som er tilknyttet studiestedet). Selv om det ikke kan tilby alt som er ønskelig, bl.a. artikler i fulltekst online, forhindret ikke dette forfatteren fra å ta i bruk dette systemet først. Grunnen til dette kan være flere; tidligere kjennskap til systemet er en av dem.

3.8.1 Søkeprosessen fortalt i form av Marchioninis termer



Figur 3–1: Scenario i Marchioninis modell for informasjonsøking.

I Figur 3–1 er en oversikt over scenarioet framstilt skjematisk. Prosessen er mer utfyllende beskrevet i Appendix B. Prosessen startet med at søkeren aksepterte at det var et informasjonsbehov. Den interne motivasjonen var et kunnskapsbehov i forbindelse med denne avhandlingen, dernest et ønske fra veilederen om 15 relevante referanser i forbindelse med arbeidet.

Problemstillingen i arbeidet omhandler “personlige samlinger i distribuerte digitale bibliotek”. Dette inkluderer emner som “digitale bibliotek”, “samlinger i digitale bibliotek”, og “personlige digitale bibliotek”. I tillegg skulle 15 relevante referanser finnes for å øke kunnskapen om emnet. Primærmålet var å øke kunnskapen, sekundærmålet var å finne 15 referanser. Formen på resultatet av søkeprosessen er vanskelig å definere. For det sekundære problemet er det forholdsvis greit da man lett kan telle antall dokumenter som ble funnet. Det blir verre når relevansen til disse skal vurderes, og det blir ikke lettere å måle den økte kunnskapen, som er det primære målet.

BibSøk-Nett fra BIBSYS ble raskt valgt som første søkesystem. Etter valg av søkesystem ble det definert noen tilleggskriterier: ønske om å få generell kunnskap om domenet og det valgte søkesystemet. Deretter ble søkekriteriene valgt.

Etter søket ble resultatsettet evaluert og funnet for lite til at det kunne være representativt. Ettersom 15 relevante treff ikke var funnet ennå, måtte søket fortsette. Spørringen ble omformulert, men ga ikke noe resultat som var godt nok til at man kunne avslutte prosessen.

For å utvide søket ble BIBSYS ISI-Søk valgt. Dette systemet ble ansett som mer lovende på grunn av innholdet. Problemdefinisjonen forble uendret.

Resultatsettet ble mye større i dette søkesystemet, og flere nye formuleringer ble nødvendig for å redusere antallet. Søket ga til slutt en overkommelig mengde resultatposter, og av disse ble 15 valgt ut som relevante. Informasjonssøkeprosessen ble derfor avsluttet.

Søkeprosessen er bare delvis beskrevet ovenfor. Ideene kommer likevel godt fram. Det går fram av beskrivelsen at Marchininis modell godt kan brukes til å beskrive det som er blitt gjort av søk i forbindelse med denne avhandlingen. Det må også nevnes at dette bare var en innledende orientering og at søkeprosessen har pågått siden arbeidet med avhandlingen startet, og at den fortsatte fram til avhandlingen var ferdig. Dette passer godt inn i beskrivelsen til Ellis og Haugan.

3.9 Grensesnitt

For å innhente krav til en personlig samlingstjeneste for distribuerte digitale bibliotek har flere grensesnitt til søkesystemer blitt undersøkt. Formålet med denne undersøkelsen er å finne ut hvordan grensesnittene ser ut, og å danne et bilde av hvordan forskjellige søkesystemer tilbyr sine tjenester. Det er også ønskelig å finne enkelte karakteristika ved disse søkesystemene som kan danne grunnlaget for en prototyp av et nytt søkesystem. Det er videre et mål å se på hvordan man kan ta vare på data i et søkesystem, og dette vil bli tatt opp i kapittel 3.9.5.

Det må presiseres at dette ikke er en grundig evaluering av systemene, men en gjennomgang for å orientere om hva som finnes og hvilke løsninger som er valgt av andre institusjoner for å tilby søketjenester til deres brukere.

3.9.1 BibSøk-Nett

Figur 3–2: Komplett post fra BibSøk Nett

Poster i BibSøk Nett kan sees i to varianter: kort og komplett. Kortformen av en post inneholder: Nummer, Tittel, Forfatter, Utgivelsesår. Dette formatet brukes i resultatlistene. Et problem med dette formatet er at tekststrengen som presenteres noen ganger kuttes, og det kan gjøre det vanskelig å undersøke posten uten å velge å se den komplette varianten. Den komplette varianten inneholder: Tittel, Forfatter, Trykt, Sidetall, ISBN, Eiere. For å se den komplette posten må man klikke på posten i resultatsettet. En post som dette er vist i Figur 3–2 på side 53. I tillegg kan man få se emneord og eierbibliotek for den aktuelle posten, og man kan eksportere hele den bibliografiske posten i forskjellige formater. Dette kan man importere i forskjellige klientside-program som EndNote[7].

Søkesystemet lar brukeren avgrense søket til artikler og andre typer materiale som f.eks. noter og musikk.

7. EndNote er et bibliografi- og søkeverktøy. Se www.endnote.com for en presentasjon.

3.9.2 Bibsys ISI-Søk

For å søke i ISI-basen må man velge ”Søk litteratur” -> ”ISI” fra menyen på www.bibsys.no. Søkegrensesnittet er enkelt nok, og tilbyr en rekke felter å søke i: Base, Forfatter, Tittel, Fritekst, Nøkkelord, Tidsskrift, Emneområde, Sortering. Figur 3–3 på side 54 viser hvordan dette ser ut.

Postene inneholder ganske mye informasjon som f.eks. hvilket tidsskrift artikkelen er publisert i, sammendrag, type artikkel, emneområde og siteringer. Spesielt siteringer og oppsummering er en bra pekepinn for å ta en avgjørelse om dette er en artikkel som dekker søkerens behov.

Figur 3–3: Søkegrensesnittet i ISI-Søk

Det er få funksjoner å spore i ISI-Søk utover søk. Man har mulighet for siteringssøk som er en måte å finne artikler som er blitt mye sitert, og som dermed er verdifulle eller viktige eller “riktige”. Derimot finnes ingen måte for å ta vare på søkene, og det gjør det vrient å komme tilbake hvis man ikke husker søkene. For å ta vare på søkene er man nødt til å lagre søkeresultatene på en ekstern måte som notater e.l.

3.9.3 D-Lib Magazine

Søkegrensesnittet i D-Lib er forholdsvis likt “vanlige” søkemotorer som Google og AllTheWeb. Et utsnitt av søkedialogen er vist i Figur 3–4 på side 55. Siden de ikke opererer med søkbare metadatafelter, blir prosessen vanskeligere enn nødvendig. Ulempen med manglende metadatafelt kan til en viss grad veies opp av at søkene foregår mot fulltekstdokumenter. Søkemotoren som ligger til grunn for søkene er ht://dig[8]. Dette er en søkemotor som søker i nettsider, og ikke en bibliografisk søkemotor.

Figur 3–4: Søkegrensesnittet i DLib

En annen ting som bør påpekes, er at de opererer med treffsikkerhetsmålinger som ikke oppfattes som pålitelige nok, slik at søkesystemet kan miste troverdighet for noen etter hvert.

3.9.4 ACM Digital Library

ACMs søkegrensesnitt kan beskrives som både enkelt og komplisert. I sin enkleste form er det kun en tekstboks der man kan skrive inn en frase. Dette er som kjent ikke det samme som flere søkeord, og det kan ta litt tid å finne ut. Det avanserte søkesystemet tilbyr langt flere muligheter for spesifisering av søket. Har man lært seg å bruke søkegrensesnittet, kan man finne det ganske kraftig.

Ved et enkelt søk får man en resultatliste med en liten beskrivelse av posten og en lenke til selve posten. Selve metadataposten er ganske kompleks uten å bli vanskelig. ACM tilbyr en tjeneste de kaller Peer-to-Peer. Denne lar brukeren se andre artikler, og se hvilke andre brukere som har lest disse. Dette minner om tjenesten som tilbys gjennom Amazon.com nevnt ovenfor.

8. Se www.htdig.org for mer informasjon.

Figur 3–5: Søkegrensesnittet i ACM

3.9.5 *Ta vare på det du finner*

Til nå har vi sett på hvordan en bruker utfører søk i noen søkesystemer, og hvordan disse grensesnittene fungerer. Et aspekt som faller inn under grensesnittet, er en brukers mulighet til å ta vare på det man finner underveis i søkeprosessen. Det er et kjent fenomen at en søkeprosess kan ta lang tid, og da kan man være hjulpet av å ta vare på en del informasjon underveis. Dette kan sees på som den tiden det tar å fullføre prosessen “Trekke ut Informasjon” i Marchioninis søkeprosess. Driver man med forskning, er det vesentlig at man tar vare på referanser underveis. I dag har man flere muligheter for å bevare informasjon om interessante informasjonsobjekter. Felles for de fleste er at de kan være tungvint å bruke. En liten presentasjon av noen av disse følger her.

Den enkleste måten for å ta vare på bibliografisk informasjon, er å skrive den ned på papir. Dette er en teknologi som fungerer overalt, uavhengig av tid og maskinvare ellers. Alt man trenger er en printer, eller papir og penn og et underlag å skrive på. Dette fordrer en del ferdigheter som skriving og lesing i henhold til Marchioninis attributter for Informasjonssøkere, men dette anses ikke som et stort problem i dag. En ulempe med denne teknologien er at det er en ganske tungvint måte for å ta vare på slik informasjon. I tilfellet med en litte-

raturliste til et forskningsarbeid er det snakk om tekst som skal skrives i et skriveprogram senere, så det medfører litt ekstraarbeid.

En annen mulighet man har, er å klippe og lime teksten fra et nettleservindu til en tekstbehandler. Denne framgangsmåten er ikke spesielt lettvint, men reduserer behovet for skriveverktøy. Til gjengjeld krever den programvare til å skrive på datamaskin med. Teknologien er universell og kan brukes uavhengig av datamaskinsystemer, men den krever at man kan ta med seg det man har skrevet på datamaskinen. Om man sitter på offentlige datamaskiner, som f.eks. på en skole eller et bibliotek, kan det bli et problem å få med seg det man har lagret på datamaskinen.

En tredje teknologi for å ta vare på funn, er eksport av metadata. Søkesystemet til BIBSYS, BibSøk Nett, tilbyr en funksjon som lar brukeren eksportere en metadatapost i et bibliografisk format, som deretter kan importeres i forskjellige referanseverktøy etc. Slike referanseverktøy er ofte programmer som er avhengige av en bestemt datamaskin, noe ikke alle nødvendigvis har tilgang til. Dermed kan det ikke sies at denne måten for å ta vare på referanser, er spesielt enkel. En finesse som letter dette, er muligheten for å bruke slike referanseverktøy i søking. Bl.a. Endnote tilbyr denne muligheten.

Bygd på muligheten for å eksportere metadata, er det laget systemer som lar en bruker håndtere metadata som objekter i en nettbutikk. Når brukeren søker seg fram til et resultatsett, kan han velge "Lagre" eller tilsvarende for å legge posten inn i en slags handlekurv. Deretter kan brukeren velge å se på handlekurven sin, og f.eks. sende den til seg selv som e-post.

En tungvint løsning for å arkivere referanser er å ta vare på skjermbilder av søkesystemet. Denne metoden fungerer så lenge man har en datamaskin med et tegneprogram. Dette er forsåvidt vanlig, men det er uforholdsmessig tungvint å kopiere skjermbilder som må lagres separat i egne programmer. Man må dessuten være i stand til å ta med seg sine funn, og bilder er en uøkonomisk måte å lagre informasjon på i forhold til tekst. Det vil derfor være vanskelig å gjøre dette om man benytter offentlige datamaskiner. En fordel er likevel at denne metoden fungerer på alle typer datamaskiner uavhengig av operativsystem.

3.10 Oppsummering

Søkesystemene som er presentert i dette kapitlet, er svært ulike når man ser på hva de tilbyr en informasjonssøker. For en som er interessert i å ta vare på informasjon, er de ganske like. De tilbyr ikke mange løsninger som er tilpasset dette behovet. Det eneste systemet som tilbyr slike tjenester, er Aperitif.no. Her har man mulighet til å legge inn sine egne oppskrifter. Nedenfor er det satt opp

en tabell som gir en kort oversikt over tjenester de forskjellige grensesnittene

Tabell 3–2: Oversikt over søketjenester i omtalte søkesystemer

Søke-system	BIBSøk Nett	BIBSYS ISI-søk	D-Lib Magazine	ACM Digital Library
Post-typer[a]	Kort Full	Kort Full	Kort Full	Kort Full
Søketyper	kommando-basert Skjemautfylling	Siteringssøking Kommando-basert Skjemautfylling	Boolsk Skjemautfylling	Boolsk Skjemautfylling
Personlig tilpasning	Ingen	Ingen	Ingen	Videresøking
Formater for eksport av data	MARC Fortekstet ProCite Research Information Systems	MARC Fortekstet ProCite Research Information Systems	Ingen	Ingen
Lagring av poster elektronisk	Nei	Nei	Nei	Mulighet for personlig “handlekurv”

a. Kort/Full/Begge: Kort post inneholder lite informasjon. Ofte bare tittel og forfatter. Full post inneholder ofte mye mer som f.eks. beholdningsdata og annen metadata.

tilbyr. Som det går fram av tabellen er det ingen systemer som tilbyr lagring av poster elektronisk. Det er noen som tilbyr eksportmuligheter, men dette krever at man tar vare på data på en egnet måte.

4 KRAVSPESIFIKASJON

I dette kapitlet presenteres kravene som stilles til en personlig samlingstjeneste. Hovedtrekkene og kravene til systemet blir gjennomgått, da disse danner grunnlaget for systemarkitekturen som blir sett på i neste kapittel.

Først i dette kapittelet vil hovedtrekkene i systemet beskrives. Deretter blir de antatte brukerne kort gjort rede for. Dette legger føringer på hva slags krav som stilles til den personlige samlingstjenesten. Deretter gjennomgås de antatte kravene fra denne typen brukere. Videre nevnes en del systemkrav som ikke settes av brukerne, men som er en del av forutsetningen i dette arbeidet. Dette omhandler f.eks. protokoller og metadata-standarder etc. Til slutt vil det bli lagt fram en del pragmatiske krav som har stor innflytelse på avgrensingen av arbeidet.

4.1 Personlige samlinger

I kapittel 1 er det beskrevet hvordan man kan trekke en brukers erfaringer inn i et digitalt bibliotek, men det er også andre kvaliteter ved en bruker som kan være av interesse for informasjonssøkere. Det mange forskjellige brukere i et digitalt bibliotek. Det som er bra informasjon for noen brukere, kan være feil for andre. Det kan derfor være av meget stor interesse for informasjonssøkere å se *hvem* som har kommentert dokumenter. Da kan søkeren selv avgjøre om kommentaren er viktig eller ikke. Interessante egenskaper ved personene er bl.a. fagfeltet de tilhører og nivået på fagkunnskap de innehar.

I et tenkt eksempel søker en grunnfagsstudent i informatikk etter et dokument som kan være til hjelp i arbeidet med en semesteroppgave. Studenter bruker et system der andre brukere kan kommentere informasjonsobjektene. Når studenten ser kommentarene, er det viktig at han også ser hvem som har skrevet dem. Om en kommentar er skrevet av en professor innen emnet studenten har sin interesse, trenger det ikke å bety at denne professoren har samme behov

som studenten. Dette skillet mellom behovene til forskjellige brukere kan gjøre søkeprosessen lettere.

I tillegg til søkefunksjoner kan man lage en funksjon i de digitale bibliotekene som lar brukere registrere poster som del av deres egne personlige samlinger. En slik personlig samling kan sees på som en slags bokhylle der en person legger inn dokumenter han finner verdifulle, for senere referanse. Denne samlingen tjener flere mål enn å ta vare på en brukers dokumenter eller referanser. Ved å åpne de personlige samlingene for allmennheten, kan informasjonssøkere søke etter informasjon ved å gå etter brukerne som har dokumentene i sin bokhylle. Ved å se at en bruker har et dokument i sin bokhylle, og søkeren vet at han har sammenfallende interesser med "eieren" av denne personlige samlingen, kan han anta at brukeren har noe interessant i sin bokhylle. Søkeren kan da undersøke den personlige samlingen for å se om det er noe der av interesse.

4.2 Brukerne

For å lage et system er det viktig å vite hvem som er de antatte brukerne av det. Dette legger føringer på hvordan systemet skal interagere med brukerne. For dette systemet er det tatt utgangspunkt i at brukeren er en student eller ansatt ved et universitet. Det antas at det er lett å lære denne typen brukere å ta i bruk det systemet som foreslås. Det er definert tre hovedtyper brukere: studenter, forskere og institusjoner. Det er ikke dermed sagt at dette er de eneste gruppene brukere av dette systemet, men det er disse som er valgt ut av praktiske hensyn, med hensyn til evalueringen senere.

4.2.1 Studenter

For å bruke Marchioninis termer kan en student kort beskrives med termene "faglig kompetanse", "leseevne", "ressoneringsevne". For å beskrive brukeren nærmere vil disse termene kvantifiseres. Det antas at en student har middels til lav faglig kompetanse. Dette må selvsagt sees i sammenheng med forskere på det samme faglige området, og studenter vil ofte ha lavere kompetanse. Ved å innføre kategorier av kompetansenivåer, er det lettere å plassere brukerne i båser. Dette letter gjenkjennelsen av nivåene. Siden systemet fokuserer på et akademisk miljø, er det lett å ta fatt i brukerens studienivå. Dette er noe som er kjent for de aller fleste i et akademisk miljø. Studienivåene som brukes, er grunnfag, mellomfag, hovedfag, forsker/professor (forklares nærmere nedenfor). Siden brukeren er student, antas det at han kan både lese og skrive. En annen forutsetning er at vedkommende kan resonnere, slik at søkeprosessen ikke hemmes av dette. Det tas ikke her høyde for folk med synshemninger eller lesevansker.

4.2.2 Forsker

Sammenlignet med en student har en forsker større fagkunnskaper innen sitt eget og nærliggende kunnskapsdomener. De er dermed en stor kilde til infor-

masjon for andre. Forskere, som studenter, antas å ha gode leseevner, helse og ressoneringsevne.

4.2.3 Institusjon

En institusjon er i større grad en tilbyder av informasjon enn en informasjonssøker. Et institutt kan gi andre brukere innblikk i det de driver med, ved å tilby en bokhylle av egne forskeres publikasjoner eller annet lesestoff som kan være av interesse innen det fagområdet de driver med. Attributter for en informasjonssøker er derfor ikke så interessante for en institusjon.

Tabell 4–1: Oppsummering av brukeres evner

Bruker	Evner
Student	Lav til middels faglig kompetanse
	God leseevne
	God ressoneringsevne
Forsker	Middels til høy faglig kompetanse
	God leseevne
	God ressoneringsevne

4.3 Funksjonskrav

Basert på de antatte informasjonssøkerne (studenter og forskere) og bidragsyterne (institusjoner) kan vi begynne å ta for oss de kravene som danner grunnlaget for bruken av systemet. Funksjonskravene er delt inn i 4 kategorier:

1. Søkefunksjoner
2. Bokhyllefunksjoner
3. Grensesnitt
4. Institusjonsbrukere

Søkefunksjoner omhandler funksjoner som støtter søking generelt og søking i distribuerte personlige samlinger spesielt. Bokhyllefunksjoner tar for seg de funksjoner en bruker vil trenge i forhold til sin egen personlige samling. Søking i andres personlige samlinger er en del av søkefunksjonene. Grensesnittkravene tar for seg hvordan grensesnittet skal utformes. Kravene som stilles av institusjonsbrukere, sees på for seg.

4.3.1 Søkefunksjonskrav

Det er naturlig å se på et personlig samlingssystem i sammenheng med informasjonsøking, da det er det som er den primære oppgaven for brukerne. Derfor må vi se for oss et søkesystem som integrerer søking og personlige samlinger. I et søkesystem må det også være mulig å søke etter informasjonsobjekter. Man

burde kunne søke på vanlige bibliografiske data, så vel som å gjøre enklere søk. Et søkegrensesnitt i dag må også tilby brukeren mulighet til å søke videre på interessante treff underveis. Slik videresøking kan vi tenke oss på emneord og forfatter.

I tillegg ville det være naturlig å søke i personlige samlinger. Dette vil si at brukeren må kunne søke på personlige metadata-beskrivelser, og basert på dette få se informasjonsobjekter og brukere som gir treff. Videre må det være mulig å søke etter andre brukere for å få se deres personlige samlinger.

Tabell 4–2: Søkefunksjoner

Kategori	Funksjon
Vanlige søkefunksjoner	Søk på forfatter
	Søk på tittel
	Søk på beskrivelse/emne
Personlige samlinger	Søk på brukernavn
	Søk i personlig metadata-beskrivelser/ metadata-annotering
Videresøking	Søk på tittel
	Søk på forfatter
	Søk på beskrivelse/emne

4.3.2 *Bokhyllefunksjoner*

Bakgrunnen for personlige samlinger eller “bokhyller” er et ønske om å ta vare på informasjon man kommer over etter hvert som man leter etter informasjon til en oppgave (Se «Faktorer» på side 35. om oppgaven). Som vi ser i scenariene, er det flere måter å ta vare på informasjonsobjekter på, men ingen som er helt ideelle. Med en personlig samling kan man ta vare på ting man finner når man er i en informasjonssøkeprosess.

En bokhylle er en samling metadataposter man har valgt å ta vare på. Bokhyllefunksjoner er de oppgavene man kan utføre på sin egen bokhylle.

Av bokhylle-funksjonalitet må en bruker være i stand til å opprette og vedlikeholde sin egen bokhylle. Dette innebærer å lagre informasjon om seg selv og det å ta vare på informasjonsobjekter. Man må kunne legge til nye informasjonsobjekter og slette dem. Man bør også kunne ta vare på bokhyllen utenom den personlige samlingstjenesten. Dette vil kreve at man har mulighet for å eksportere dataene man har lagret til et egnet format. Det burde også være mulig for brukerne å beskytte sin bokhylle mot innsyn, eller beskytte kommentarer mot offentligheten.

Metadata

I detalj må en bokhylle kunne ta vare på personlige data. Dette er brukerens navn, adresse, faglig interesse, og akademisk grad/faglig nivå. Se Tabell 4–3. Navn og adresse er greit å vite om man ønsker å komme i kontakt med den aktuelle brukeren, om man bare har denne brukerens bokhylle å forholde seg til. Faglig interesse er med på å bygge opp en forståelse av hva denne brukeren jobber med og eventuelt kan noe om. En benevnelse av akademisk grad eller tilsvarende kan gi en person en pekepinn på om vedkommende bruker er dyktig i dette spesielle fagområdet. I en større sammenheng er det naturlig å ønske å ta vare på mer informasjon om brukeren. Dette kan være data om alder, betalingsopplysninger (om man ønsker å ta betalt for deltakelse i det personlige samlingssystemet), eller andre data. Dette vil bli sett bort fra i det aktuelle systemet.

Tabell 4–3: Brukermetadata

Navn
Adresse
Faglig interesse
Faglig grad/ nivå
Institusjon

Det er ikke nok å ta vare på informasjon om brukerne i et personlig samlingssystem. En bokhylle må også inneholde metadata som brukeren har lagt til det enkelte objekt. For et objekt i bokhyllen må systemet ta vare på en beskrivelse lagt til av brukeren. Det må også være mulig å finne den metadata-posten som brukeren har kommentert. Altså må det finnes en identifikator i bokhyllen som peker på den originale metadataposten. Videre trengs det emneordsangivelser på postene. Dette gjør at det er lettere å søke etter bokhylle-poster.

Tabell 4–4: Personlig metadata

Personlige beskrivelse
Identifikator
Emneord

4.3.3 Grensesnitt

Da grensesnittet i søkesystemet ikke er av spesiell interesse for det Personlige Samlingssystemet, er det ikke interessant å spesifisere alle detaljer rundt brukergrensesnitt-design her. Det forutsettes at et system implementerer veloverveide krav til design av brukergrensesnitt og brukbarhet. Dette vil sikre at systemet fungerer og tilfredsstiller sluttbrukeres behov for gode brukergrensesnitt.

4.3.4 Institusjonsbrukere

Som nevnt ovenfor har en institusjon andre mål med å bruke en personlig samling enn vanlige brukere. Institusjoner har bl.a. ikke interesse av å søke. Aspektene der institusjoner tilgjengeliggjør informasjonsobjekter er derimot mer interessante. For å klare dette må man kunne lage en personlig samling for institusjoner, og man må kunne søke disse opp, eller tilgjengeliggjøre dem på annet vis. I bunn og grunn trengs det ikke andre krav enn de vi allerede har beskrevet, bortsett fra at det ikke gir mening å sammenligne det akademiske nivået på en institusjon og en sluttbruker. Derfor er det valgt å droppe spesielle krav for institusjoner.

4.3.5 Generelle krav.

For at brukerne skal ha tilgang til systemet må krav for *tilgjengelighet* fastsettes. Brukere jobber i større grad uavhengig av eget kontor og utenom vanlig kontortid. Derfor kreves det at systemet støtter dette. Dette vil si at systemet må være tilgjengelig flere steder for samme bruker, som f.eks. på Internett. Videre er det et krav at brukeren kan bruke systemet når det måtte passe han.

Det er også et krav at brukeren ikke skal være avhengig av *spesielt utstyr* for å ta i bruk den personlige samlingstjenesten. Dette vil si at man med en normal datamaskin med Internett-tilgang kan bruke systemet. Systemet krever med andre ord ikke et spesielt operativsystem eller en spesiell nettleser. Det kreves riktignok at brukeren har en nettleser av forholdsvis ny dato, slik at den følger de gjeldende Internett-standardene fra W3C[54] .

Tabell 4–5: Oppsummering av funksjonskrav

Krav	Beskrivelse
Vanlige søkefunksjoner	Søk på forfatter
	Søk på tittel
	Søk på beskrivelse/emne
Søkefunksjoner knyttet til personlige samlinger	Søk på brukernavn
	Søk i personlig metadata beskrivelser/metadataannotering
Videresøking	Søk på tittel
	Søk på forfatter
	Søk på beskrivelse/emne
Bokhyllefunksjoner	En bruker må kunne legge til et objekt i bokhyllen
	En bruker må kunne slette et objekt fra bokhyllen
	En bruker må kunne se sin egen bokhylle
	En bruker må kunne se en anne brukers bokhylle

4.4 Systemkrav

For at en sluttbrukers funksjonskrav skal kunne tilfredsstille en personlig samlingstjeneste, må den samarbeide med andre systemer. Dette gjør at det må stilles krav til den personlige samlingstjenesten om hvordan den skal forholde seg til andre systemer. Dette stiller krav til protokoller og metadataformater.

For å kunne utveksle metadata med andre systemer, er det nødvendig å forholde seg til standardiserte metadataformater som f.eks. Dublin Core (DC)[12] , Marc[34] eller RFC-1807[25] . Av disse er Dublin Core det enkleste, og for en sluttbruker som skal kunne beskrive ressurser selv, er enkelhet et viktig punkt. Om metadataformatet blir for komplisert, blir det for vanskelig for vanlige brukere å beskrive det de ønsker i tjenesten. Derfor er det nærmest et krav at metadata lagres i DC-formatet. Det er riktignok mulig å lage et metadataformat spesielt tilpasset personlige samlinger, men det er ikke nødvendig så lenge DC tilbyr det som trengs for å registrere korrekte og gode nok metadata. Ved å lagre de personlige samlingene som metadataposter i DC, kan man lett utveksle disse.

For å kunne spesifisere enkelte bokhyller, må hver bokhylle ha en identifikator. Det er ikke noe krav til identifikatoren annet enn at den må være unik. Her kan f.eks. Handle System eller DOI eller andre brukes.

For å tilby personlige samlinger i distribuerte digitale bibliotek, er det nødvendig å holde seg til protokoller for å kommunisere med sluttbrukertjenestene. Det er flere protokoller som er mulig å benytte. Z39.50 er en søkeprotokoll som er spesielt utviklet for å søke i metadata. Det er mulig å gjøre presise søk og å hente poster via Z39.50[60] . Protokollen er imidlertid stor og vanskelig å sette seg inn i. Dette gjør at det er vanskelig å lage programmer som benytter den. Det er mulig å bruke http som overføringsprotokoll og lage et søkesystem som bruker http. Dette er en mye enklere protokoll å forholde seg til da den kun er ment som en overføringsprotokoll. Det er dessuten laget mange overbygg på http som åpner for mange ulike tjenester. En av disse er Simple Object Access Protocol (SOAP)[51] . Dette er en protokoll for sending av meldinger basert på XML[49] og http[50] . Dette er en del av WebServices[52] arkitekturen.

4.5 Avgrensinger

For å avgrense arbeidet er enkelte av de overfor nevnte kravene sett bort fra. Dette gjelder alle kravene som har med søkegrensesnitt å gjøre, da de ikke er spesielt interessante for å finne ut om selve arkitekturen fungerer. Derfor vil ikke grensesnittet bli fullstendig implementert.

Et stort utvalg av databaser er heller ikke avgjørende. Det vesentlige er at man har definerte krav til databasene man gjør tilgjengelig. Dette er viktig, da man på et senere tidspunkt kan finne flere databaser som tilfredsstiller disse kravene, og man kan så legge disse til i systemet.

Det vil heller ikke gjøre noe fra eller til om man bare bruker én protokoll og ett metadata-format. Dette vil gjøre arbeidet mer oversiktlig og mer fokusert på problemstillingen.

At systemet støtter W3C-standarder fullt ut er heller ikke noe krav for at man skal kunne undersøke problemstillingen. Det er riktignok meget gunstig for videre arbeid som kan gjøres. Ved en eventuell utnyttelse av ideene i dette

arbeidet i andre arbeider vil det være en fordel om det er mulig å utvikle samlingstjenesten i w3c-standarder.

Når det kommer til protokoller, er det ikke avgjørende, men dog interessant, å undersøke bruk av flere protokoller. Det er imidlertid ikke avgjørende for undersøkelsen av ideen om personlige samlinger. Det vil derfor kun bli brukt én protokoll for overføring av personlige samlinger, og søking i personlige samlinger blir ikke utviklet fullstendig. Dette er en del som må utvikles mye bedre i en fullstendig versjon av systemet, men det er ikke av avgjørende betydning for å teste ideen.

4.6 Systemtyper

Det er tre hovedtyper av system-implementasjoner som er naturlige å tenke seg i forbindelse med en personlig samlingstjeneste. Den første er et selvstendig system, der all søking og bruk foregår i ett system. Den andre er en personlig samlingstjeneste som en tilleggstjeneste i et annet søkesystem, og den tredje er et system som formidler søk i flere søketjenester og integrerer resultatet med en personlig samlingstjeneste.

I et system der all søking og lagring foregår i ett system, må all metadata ligge lagret i én tjeneste, og den personlige samlingen knyttes opp mot denne. Det er flere problemer med denne måten å løse problemet på. Metadata må synkroniseres med den tjenesten som produserer metadataene. En vel så viktig problemstilling er tilgangen til den personlige samlingen. Den vil kun være tilgjengelig fra dette ene systemet. Det vil si at man mister muligheten til å ha en personlig samling som er tilgjengelig i flere søkesystemer. Det er dessuten urealistisk å få alle brukere over på ett system. Det vil nesten alltid være et annet søkesystem som tilbyr andre data som er viktige i enkelte situasjoner, og som i akkurat det øyeblikket ikke var tilgjengelig via den personlige samlingstjenesten. Brukere vil derfor måtte forholde seg til flere systemer, uten at de kan bruke den samme personlige samlingen.

Den andre typen system er en personlig samlingstjeneste som en tilleggstjeneste i et annet søkesystem. Dette kan f.eks. være et allerede eksisterende digitalt bibliotek, som utvides med en personlig samlingstjeneste. I denne løsningen vil brukeren har tilgang til ett søkesystem, mens den personlige samlingstjenesten er tilgjengelig eksternt sett fra søkesystemet. Det vil si at det er mulig å bruke den samme personlige samlingstjenesten i flere søkesystemer. Dette er ønskelig, som nevnt tidligere. Videre har man i denne løsningen bare tilgang til én metadatabase. Dette er ikke nok, da mange brukere helt sikkert vil trenge metadata som ikke ligger lagret i akkurat dette systemet. Det vil si at brukeren må forholde seg til et nytt søkesystem. Fordelen er som sagt at dette nye systemet kanskje har implementert støtte for personlige samlinger, så tilgangen til dem er den samme som i det første systemet.

Den siste typen system, er et søkesystem der søkesystemet tillater søk mot mange eksterne metadatakilder. Ved å legge til en personlig samlingstjeneste,

vil de eksisterende systemene knyttes til personlige metadataposter. Denne løsningen lar brukeren få tilgang til mange søkesystemer fra ett brukergrensesnitt, og gir også tilgang til de personlige samlingene. Dette er en fordelaktig løsning. Den innfører ikke problemene fra de to foregående løsningene ved at flere metadata-kilder er tilgjengelige, og ved at den personlige samlingen ikke er knyttet til dette ene grensesnittet.

Det er selvsagt ulemper også med denne løsningen, som med de to andre. Ett samlet søkegrensesnitt mot mange søketjenester er mer komplisert å vedlikeholde. Det krever at man kan koble seg til andre søketjenester. Dette kan løses ved å bruke dedikerte søkeprotokoller som Z39.50[60] . Dette er gjort av f.eks. BIBSYS med søketjenesten ZSøk[9]. Søketjenesteutvalget i en slik samletjeneste er ikke nødvendigvis stort nok. Om utvalget ikke dekker brukerens behov, faller mye av begrunnelsen for et slikt samle-søkesystem bort.

4.6.1 Lokal applikasjon eller Nettapplikasjon

Spørsmålet om den personlige samlingstjenesten skal være en internett-applikasjon eller et selvstendig kjørende program, er allerede implisert, men ikke besvart. Til nå i kapitlene som omhandler den personlige samlingstjenesten, er det antatt at applikasjonen er tilgjengelig på Internett, enten som selvstendig søketjeneste, eller som en tilleggstjeneste i andre søketjenester på Internett, men applikasjonen kunne også være tilgjengelig som et lokalt program på brukerens datamaskin. Denne løsningen vil bli drøftet nedenfor.

I en såkalt Peer-To-Peer plattform har hver bruker et program på sin egen datamaskin som brukes for å kommunisere med andre applikasjoner og en tjener via Internett. Denne typen applikasjon er i dag meget utbredt som fildelingsprogrammer[10]. For en personlig samlingstjeneste vil hver bruker ha et eget søkeprogram på sin egen datamaskin, som kobler brukeren på Internett og åpner denne brukerens personlige samling for andre brukere. I tillegg vil det måtte eksistere en tjener som tilbyr objektiv metadata, slik at man ikke bare har personlige samlinger å søke i. Som det går fram, vil hver bruker med dette systemet ha sin egen personlige samling tilgjengelig på sin egen datamaskin, og er derfor ikke avhengig av Internett-systemer for å få tak i sin personlige samling. Dette vil gi en fordel for tjenermaskinen som tilbyr objektiv metadata. Det vil være mindre for den å lagre, så det vil da bli “billigere” i drift. Det kreves heller ikke noen applikasjon som lager grensesnitt til brukere og bearbeider søkene på noe som helst slags måte. Dette vil bli håndtert av den lokale applikasjonen.

Det er ikke bare fordeler med at en lokal applikasjon skal håndtere så mye av informasjonen. For at et system som dette skal kunne fungere, må mange brukere være pålogget nettverket av personlige samlinger. Hvis man antar at brukere logger på nettverket, utfører sine søk, og så logger av igjen etter forholdsvis liten tid, vil denne brukerens personlige samling ikke være tilgjengelig for andre brukere. For de brukerne av Internett som ikke har bredbåndsforbindelse, kan det være ønskelig å være pålogget så lite som mulig. For de

9. Søketjenesten ZSøk er tilgjengelig på http://www.bibsys.no/zsok
10. Eksempler på fildelingstjenester er Dircet Connect (DC), Kazaa og Napster.

brukerne som betaler for den mengden data de sender eller mottar, vil det også være ønskelig med lite trafikk. Dette vil begrense viljen til å dele sine filer med andre brukere, og dermed begrense selve samlingstjenesten.

Det er andre ulemper også. Når den personlige samlingen er knyttet opp til en bestemt datamaskin med et bestemt program installert, vil dette være et hinder for de brukerne som ikke har tilgang til en slik datamaskin. Dette gjelder ofte studenter, som bare har tilgang til datasaler uten at hver student har sin datamaskin. Til slutt kreves det selvsagt at brukeren kan installere et slikt program og koble det til Internett.

Alt i alt er en av de største ulempene med en slik Peer-To-Peer applikasjon at det kreves at brukeren laster ned dette programmet, og installerer det på sin egen datamaskin. Dernest vil mangelen på tilgjengelighet til den personlige samlingen når brukeren ikke er logget på, være et viktig ankepunkt mot en slik applikasjon. En søketjeneste som en personlig samlingstjeneste, er lite verdt uten at brukere har personlige samlinger å dele med de andre brukerne. Fordelen med denne typen applikasjon er først og fremst tilgang til den personlige samlingen uten å måtte logge på den personlige samlingstjenesten på Internett. Dette kan løses i en Internett-applikasjon ved hjelp av en eksportfunksjonalitet som lar brukere laste ned sine personlige samlinger til sin egen datamaskin.

5 BESKRIVELSE AV SAMLINGSTJENESTE

I dette kapitlet vil arkitekturen for en personlig samlingstjeneste beskrives. Det er i hovedsak to modeller et slikt system kan beskrives med. Den første er et system som står på egne ben og lar brukere søke i ett digitalt bibliotek eller flere, og i tillegg søke i personlige samlinger. Den andre modellen er et system som kan legges til i et vanlig søkesystem omtrent som en modul. Denne løsningen legger til funksjonalitet for personlige samlinger i et allerede eksisterende system.

Dette kapitlet vil beskrive og diskutere den konseptuelle arkitekturen for systemet. Det er utarbeidet flere forslag til løsninger som tilfredsstiller kravene fra forrige kapittel i varierende grad. Valget av modell for systemet påvirker valget av arkitektur. Valg av arkitektur påvirker også hvilken modell av systemet som er best egnet for implementasjon.

5.1 Arkitekturer

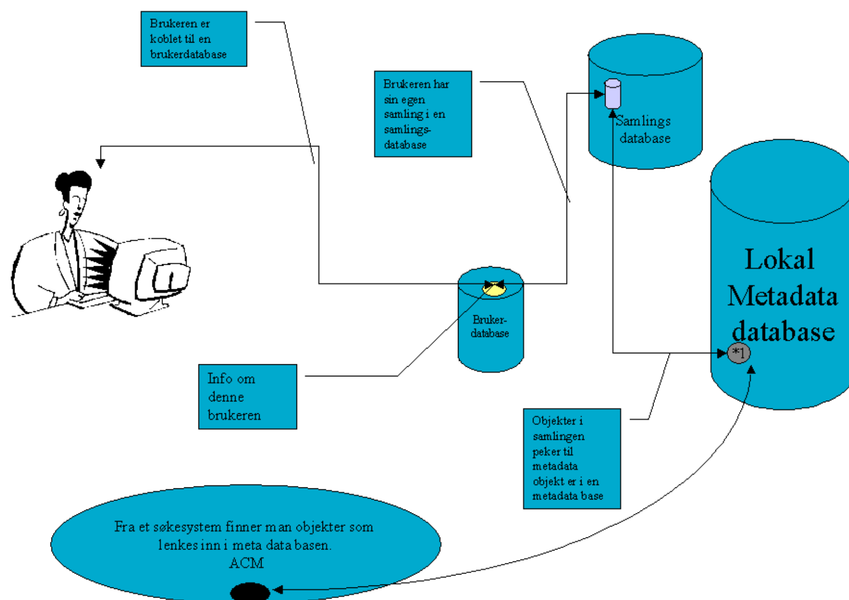
5.1.1 Lagdeling og komponenter

Arkitekturene som presenteres i dette kapitlet er lagdelte. Med det menes at de har et databaselag i bunnen, deretter et applikasjonslag som fungerer som grensesnitt mot databasene. Videre ligger det et applikasjonslag som fungerer som selve samlingstjenesten. Til slutt må det legges et brukergrensesnitt på toppen. Sistnevnte håndterer kommunikasjon mot brukerne.

Ved å dele arkitekturen inn i slike lag, blir det lettere å vedlikeholde applikasjonen. Man kan bytte ut komponenter med nye uten at man trenger å fikse på hele programsystemet. Oppdager man f.eks. at Brukerdatabasen er treg, kan man benytte et annet databasestyringssystem, eller endre applikasjons-grensesnittet mot databasen.

For å utarbeide en systemarkitektur er det laget tre forslag til systemarkitekturer som oppfyller de kravene som ble gjennomgått i forrige kapittel. Det er fordeler og ulemper med alle disse systemarkitekturene, og disse vil bli diskutert etter hvert. Videre er det utarbeidet scenarier for de forskjellige arkitekturene. Disse scenariene er de samme for hver arkitektur, og vil bli diskutert etter hvert.

5.1.2 Systemarkitektur 1



Figur 5–1: Systemarkitektur 1

Systemarkitektur 1 danner utgangspunktet for en applikasjon som kan brukes uavhengig av andre applikasjoner. Den baserer seg på tre komponenter: Brukerdatabase, Lokal metadatabase og en Samlingsbase. Disse tre komponentene vil bli nærmere presentert nedenfor.

Brukerdatabase (BDB)

Brukerdatabasen inneholder primært bare én ting: informasjon om brukerne i systemet. Om hver bruker må vi ha minst følgende informasjon: Navn, Identifikator, Stilling (student/forsker/professor), Grad (Dr./Cand.Scient./Cand. Mag. etc.), Institusjonell tilknytning (universitet/sykehus/privat forskningsinstitusjon), Arbeidssted/studiested. En videre forklaring av disse er gitt i Tabell 5–1.

Disse opplysningene lagres i en egen database, men man kan bruke en institu-

Tabell 5–1: Metadatapost i Brukerdatabasen

Post	Beskrivelse
Navn	Brukerens navn.
Identifikator	En entydig identifikator som identifiserer denne brukeren.
Stilling	En angivelse av brukerens stilling. Dette kan være forsker, student, professor e.l.
Grad	En benevnelse av akademisk grad.
Institusjonell tilknytning	Angir om denne personen er tilknyttet et syke-hus, universitet e.l.
Arbeids- eller studiested	Navnet på den institusjonen brukeren er tilknyt-tet.

sjons egen brukerdatabase om denne allerede eksisterer.

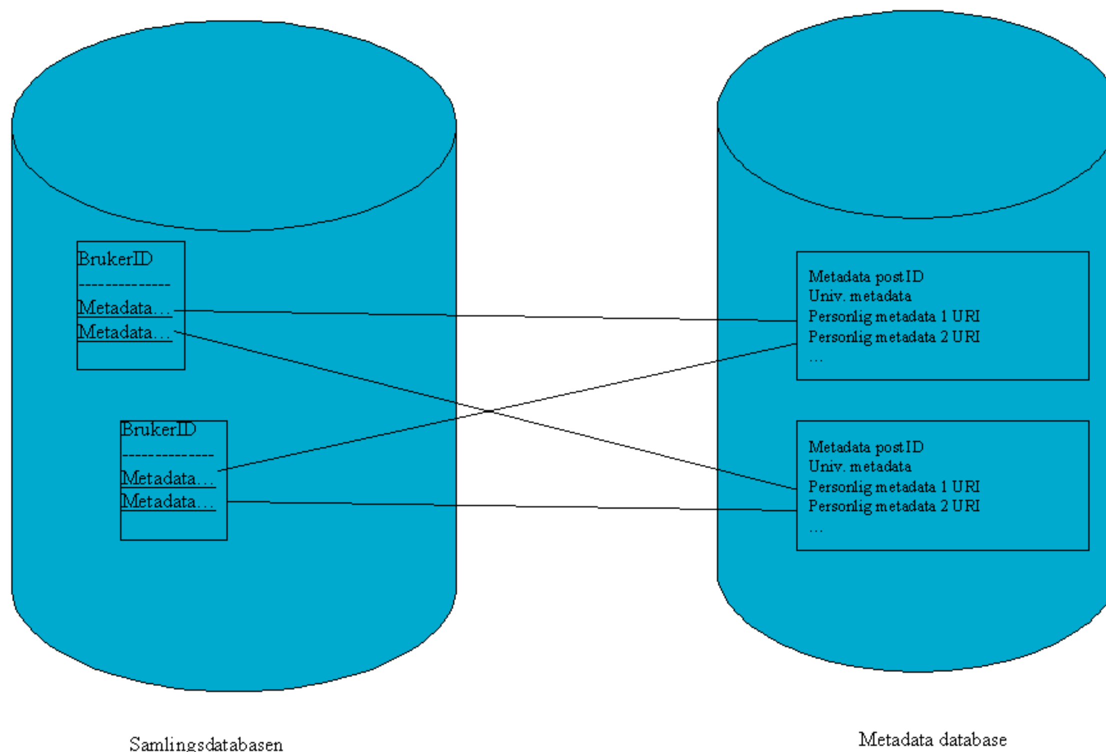
Brukerdatabasen tilbyr funksjoner som lar en bruker registrere seg, endre på, og hente fram personlig informasjon, og søke etter brukere.

Samlingsdatabasen
Samlingsdatabasen inneholder hver brukers samlinger. Det vil si hvilke informasjonsobjekter brukeren har "tatt vare på". Dette er bare en kobling mellom en "BrukerID" og en "Metadata postID". Dette vil forklares nærmere i forklaringen av lokal metadata-base. Samlingsdatabasen må kunne aksesseres med både en henvisning til en bruker og en henvisning til en metadata-post. Dette åpner for at en bruker kan finne alle objekter en annen bruker har i sin samling, og for at man kan finne alle brukere som har en gitt metadata post i sin samling.

En post i samlingsbasen inneholder en BrukerID som identifiserer brukeren som denne posten tilhører, og en rekke personlige metadataposter. Disse postene er bygd opp av en metadata-URI som peker på en post i den lokale metadatabasen for å lenke til den objektive metadataen der. Videre inneholder de en vanlig metadatapost som er lagt til av brukeren.

Lokal metadatabase
I denne arkitekturen er det lagt opp til at all lagring av objektiv metadata skjer i en lokal metadatabase. Det vil si at man ikke bare lenker til en metadata-post i for eksempel BIBSYS, ACM eller ISI-basen, men laster inn hele posten og lagrer den. Hver metadata-post er bygd opp av flere elementer:

> Metadatapost-Identifikator: Dette er en entydig identifikator for denne posten i den personlige samlingstjenesten. Den er gyldig i både den lokale metadatabasen og i samlingsbasen.



Figur 5–2: Metadata-struktur i systemarkitektur 1

Objektiv Metadata: Dette er metadata som er katalogisert av en profesjonell, f.eks. en bibliotekar. Det er denne typen data som importeres til den personlige samlingstjenesten fra eksterne eller andre interne kilder.

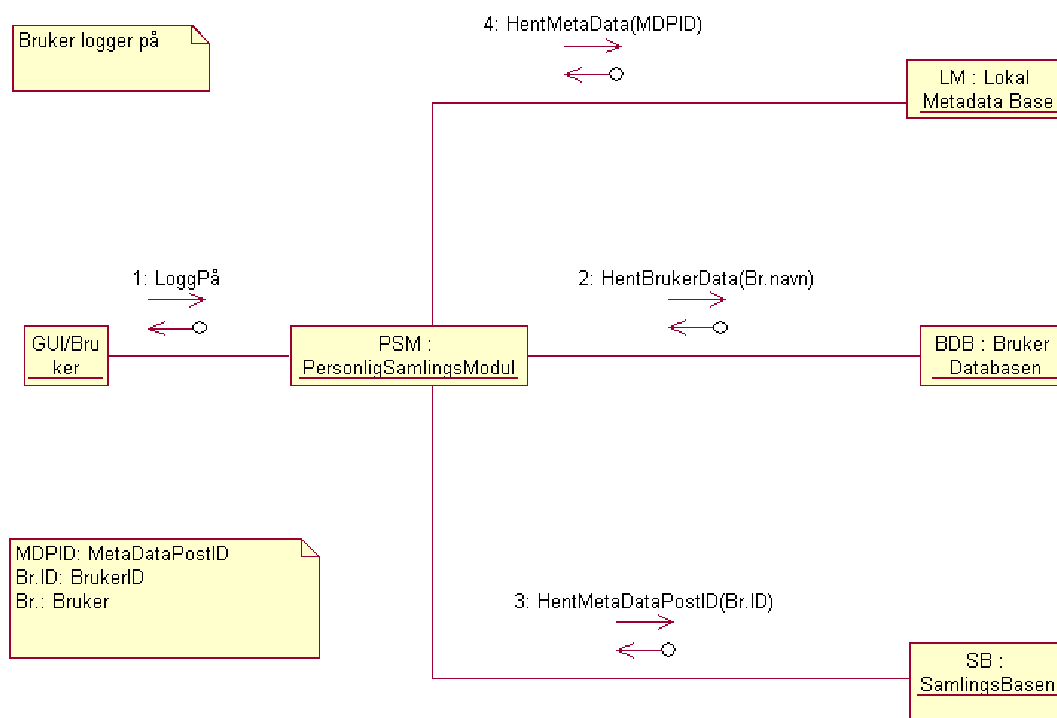
Personlig Metadata-URI: Dette er en lenke som refererer til den posten i samlingsbasen som inneholder en personlig kommentar til denne metadataposten.

Som det går fram av modellen så langt, er en metadatapost todelt. Den personlige delen av metadataposten lagres i Samlingsbasen, og den objektive delen i den Lokale metadatabasen. Denne sammenhengen er vist i Figur 5–2 ovenfor.

Scenario i systemarkitektur 1 i et brukerperspektiv

Vi antar at en bruker av den personlige samlingstjenesten har et informasjonsbehov, og velger å forsøke å få dette dekt av den personlige samlingstjenesten. Denne brukeren må først "gjenkjenne og akseptere-", og "definere problemet", og deretter velge den personlige samlingstjenesten som system for å dekke informasjonsbehovet. Brukere logger på systemet, og tjenesten henter så fram den personlige samlingen til denne brukeren. Prosessene bak dette kallet er forklart i scenariet nedenfor.

Brukeren har startet fra en nettside som lar han logge seg på systemet. Dette ble gjort ved å skrive inn brukernavn og passord. Dette ser man på Figur 5–3 som kall 1. Fra grensesnittet sendes brukernavn og passord til hovedprogrammet som tar hånd om denne handlingen og videreformidler den. Brukernavnet sendes til BrukerDataBasen for å hente brukerinformasjon. Dette sees som kall



Figur 5–3: Scenario1 i systemarkitektur 1: bruker logger på systemet.

2 i figuren. Om brukernavn godkjennes, vil hovedprogrammet sende BrukerID som hentes fra Brukerdataposten, til SamlingsBasen (kall 3 i figuren). Der vil poster som har den aktuelle BrukerID'en hentes fram, og personlig metadata og MetadatapostID'er sendes i retur (kall 4 i figuren). Hovedprogrammet sender så forespørsler til metadatabasen etter metadataposter som har identifikatorer som tilsvarer de som kom fra samlingsbasen. Til slutt sendes metadatapostene i retur til hovedprogrammet som genererer data for å sende tilbake til grensesnittet og sluttbrukeren.

Brukeren undersøker så om objektene i den personlige samlingen gir svar på problemet. Dette innebærer mye av de samme prosessene som Marchionini beskriver i sin modell: både å utforske resultatsettet og refleksjon gjør at brukeren tar en avgjørelse med hensyn til informasjonsproblemet. Utforske resultatsettet går ut på å undersøke den personlige samlingen og se om det er noen poster der som gir svar på problemet. Deretter må brukeren trekke ut informasjon fra resultatet. Dette kan f.eks. løses ved å bestille en artikkelkopi av et objekt i den personlige samlingen, eller å laste ned et fulltekstdokument. Til slutt må brukeren reflektere over resultatet. Er problemet blitt løst? Om ingen objekter passer brukerens informasjonsbehov, må brukeren velge et nytt søkesystem.

Det er verdt å påpeke at det i dette scenariet ikke er blitt utført et aktivt søk. Situasjonen er heller den at et søk er blitt utført av systemet og levert til brukeren. Dette søket er på ingen måte dynamisk. Det er kun blitt hentet poster som brukeren tidligere har valgt å ta vare på, og om brukeren har et helt annet infor-

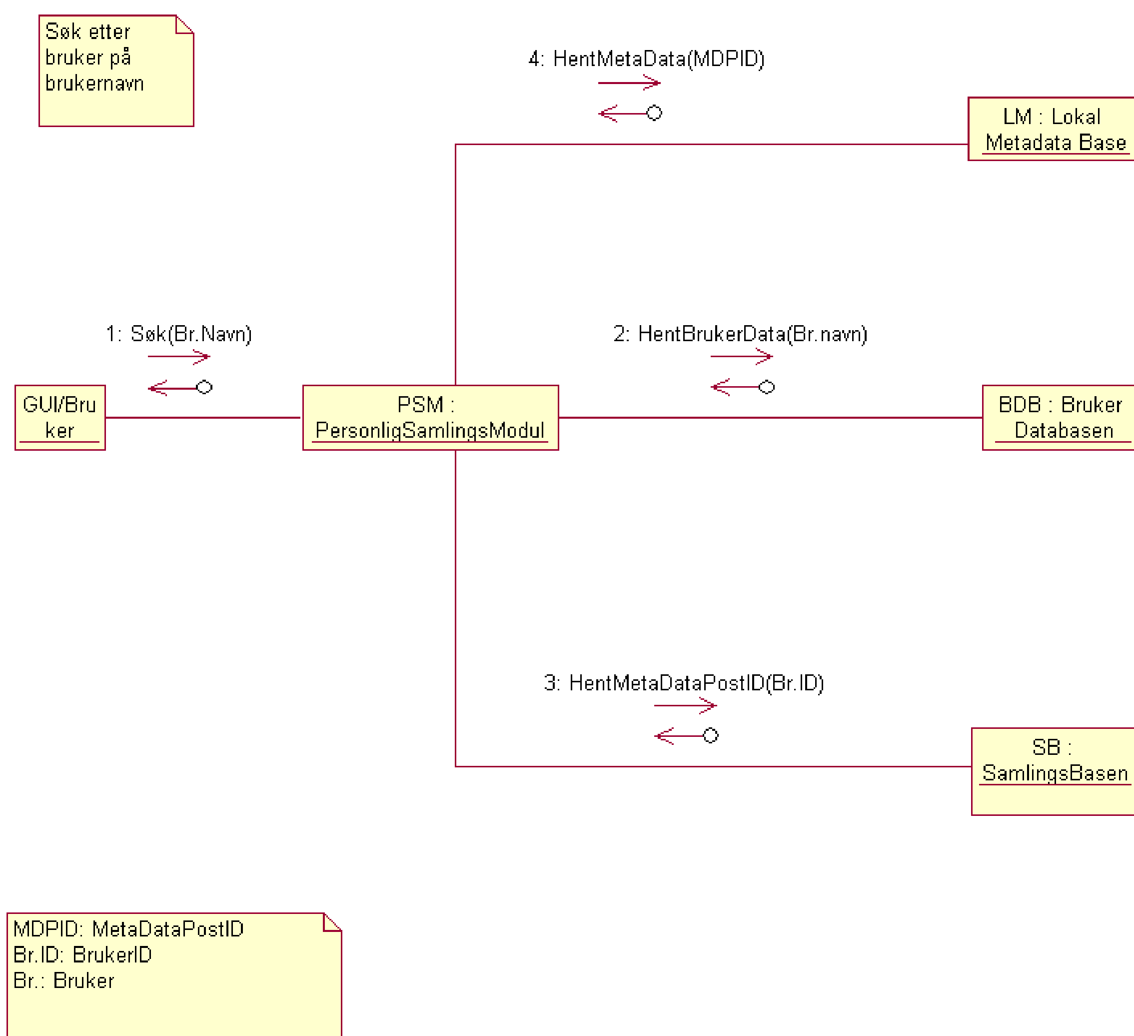
masjonsproblem i dette scenariet enn tidligere, vil dette søket antakelig være forholdsvis verdiløst. Det er på den andre siden mulig at problemet er av typen "Hva var det nå Marchioninis modell for informasjonssøking i elektroniske ressurser egentlig gikk ut på". Har man studert Marchionini tidligere, vil man i den personlige samlingen sannsynligvis ha en referanse til boka der Marchionini presenterer modellen, og problemet vil være løst allerede ved innlogging i tjenesten.

Om ikke den personlige bokhyllen ga løsning på problemet, må brukeren søke i andre systemer. Brukeren må da velge om han vil søke etter andre brukere i Det Personlige samlingssystemet, eller søke i andre ressurser. Vi antar at brukeren velger å søke etter andre brukere først. Brukeren angir et brukernavn til en bruker han tror kan ha fornuftige poster i sin personlige samling. Et eksempel her kan være at en student søker etter den personlige samlingen til foreleseren i et fag, og forventer å finne pensumrelevant litteratur. Etter søket presenterer tjenesten den personlige samlingen til den angitte brukeren. Søkeren må da gå gjennom de samme prosessene som nevnt for undersøkelse av ens egen personlige samling.

Om dette søket ikke gir det ønskede resultatet, vil brukeren måtte utvide søket. Dette kan gjøres ved at brukeren velger å søke i emner i metadata. Et søk i metadata resulterer i en liste poster som tilfredsstiller det brukeren ber om. Den vesentlige forskjellen mellom et tradisjonelt søkesystem og den personlige samlingstjenesten, er at den siste presenterer poster som også brukere har gitt emneord som sammenfaller med det søkeren har bedt om. Resultatsettet etter et søk i den personlige samlingstjenesten har antakelig flere poster enn et tradisjonelt søk, men det er mulig å få treff på poster som ikke ville gitt treff ved tradisjonell katalogisering. Et vesentlig tilskudd til metadatapostene er at brukere er knyttet til metadataposter, og søkere kan dermed finne brukere som kan ha interessant informasjon. En søker kan da f.eks. se etter hos andre brukere om de har noe interessant som søkeren ikke har kommet over. Tradisjonelle søkesystemer støtter ikke denne typen browsing i dag.

Scenario 2: Bruker søker etter bruker ved brukernavn

Når en bruker ønsker å søke etter andre brukere, må han først finne fram til søkemuligheten for dette i brukergrensesnittet. Hvordan dette fungerer, er selvsagt avhengig av brukergrensesnittet. Forespørselen om brukerinformasjon går fra brukergrensesnittet og inn til samlingsmodulen (kall 1 i Figur 5–4 nedenfor). Den personlige samlingstjensten sender så en forespørsel til Brukerdatabasen etter brukerinformasjon om denne brukeren (kall 2 i Figur 5–4). Deretter blir brukerinformasjon returnert. BrukerID'en hentes så ut fra brukerinformasjonen, og sendes til Samlingsbasen (kall 3 i Figur 5–4). Der hentes alle poster som har den gitte BrukerID'en fram og sendes i retur til hovedtjenesten. I disse postene fra samlingsbasen finner hovedtjenesten fram til Metadatapost-ID'er. Disse sendes så til den lokale metadatabasen (kall 4 i Figur 5–4). Den returnerer postene som forespurt. Om det er flere metadatapost-forespørsler, vil dette siste kallet bli gjentatt til alle postene er hentet. Deretter vil hovedtjenesten lage en post som inneholder all informasjon som er blitt hentet fram (brukerinformasjon og metadataposter), og sender dette til brukergrensesnittet

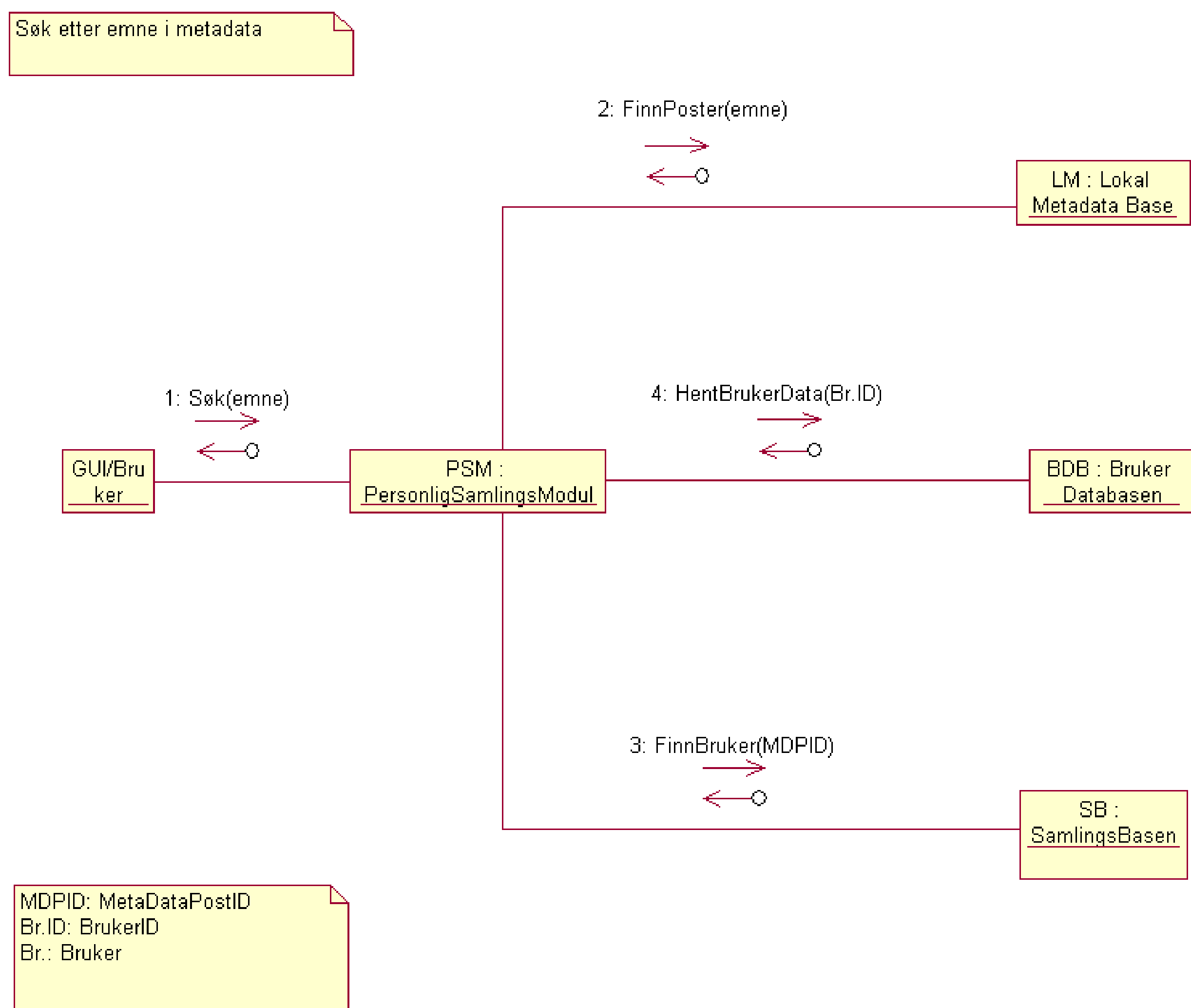


Figur 5–4: Scenario 2 i systemarkitektur 1: Bruker søker etter bruker ved brukernavn

som lager en presentasjon av den personlige samlingen til den angitte brukeren.

Scenario 3: Bruker søker i metadata etter emne

Når en bruker søker etter emner, vil han måtte angi dette i brukergrensesnittet som sender denne forespørselen til den personlige samlingstjenesten. Dette ser man som kall 1 i Figur 5–5. Deretter sender tjenesten en forespørsel til metadatabasen om å returnere alle poster som har det angitte emneordet registrert (kall 2 i Figur 5–5). Metadatabasen sender så postene i retur, og fra disse postene henter samlingstjenesten ut ID'ene. Disse sendes så mot samlingsbasen som returnerer alle poster den finner (kall 3 i Figur 5–5). Disse postene inneholder som kjent bl.a. BrukerID'er, og disse sendes til brukerdatabasen for å hente ut informasjon om brukerne som har de angitte postene i emnesøket (kall 4 i Figur 5–5.) Til slutt sender samlingstjenesten postene til brukergrensesnittet.



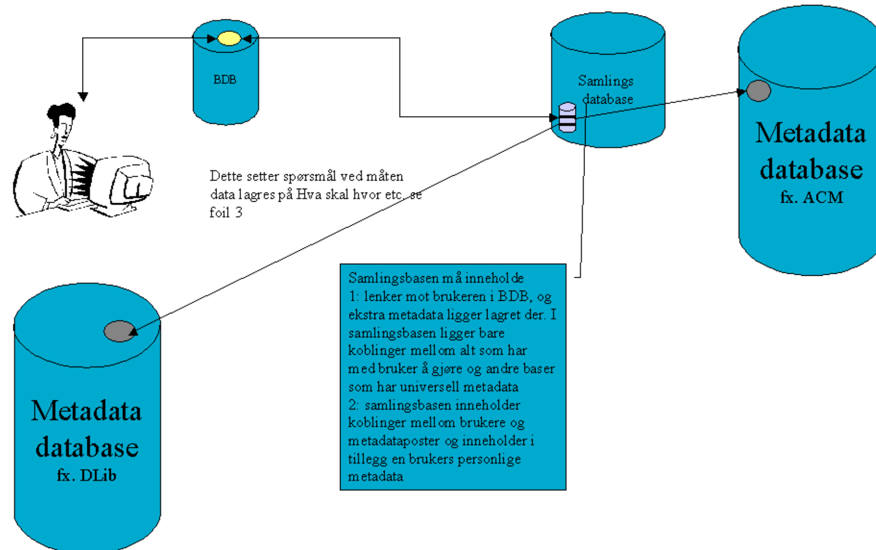
Figur 5–5: Scenario 3, systemarkitektur 1:

Hva er fordeler og ulemper med denne arkitekturen?
Denne arkitekturen danner et bilde av et søkesystem som ikke trenger andre systemer for å kunne operere. Det er naturlig å se det som et selvstendig søkesystem, og ikke som en innpluggsmodul til andre søkesystemer (selv om dette ikke er umulig). Siden all metadata er tilgjengelig uavhengig av andre metadata-systemer, vil man ikke "miste" data om de andre systemene ikke er tilgjengelige. Videre er dette en raskere måte å få tilgang til metadata på. I dag vil ofte oppslag i tjenester som baserer seg på Internett, være tregere enn når data er lagret lokalt.

Ulempen med dette valget er at metadata ikke er synkronisert med kilden. Hvis de som opprinnelig er "eiere" av metadataposten gjør endringer i den, vil ikke disse endringene gjenspeiles i den personlige samlingstjenesten. Dette gjør at brukerne ikke har tilgang til oppdatert informasjon, og systemet kan tape troverdighet. Det er også problemer knyttet til rettigheter i denne løsningen, siden man er avhengig av å få tilgang til alle metadatapostene for å kunne legge dem inn i den personlige samlingstjenesten.

En separat brukerdatabase vil være en fordel for de som allerede har en brukerdatabase. Da trenger man ikke vedlikeholde to brukerregistre, noe som er en fordel da slike vedlikeholdsoppgaver har en tendens til å bli dyrere og dyrere etter hvert som tiden går.

5.1.3 Systemarkitektur 2



Figur 5–6: Systemarkitektur 2

Systemarkitektur 2 skiller seg fra systemarkitektur 1 ved at den bruker de forskjellige søkesystemene i større grad for å nyttiggjøre seg metadata. Metadataposter lagres ikke lokalt, men refereres til hos andre leverandører. Det vil derfor medføre at man ikke kan lage et helt selvstendig system med denne arkitekturen. Brukerdatabasen har her samme funksjonalitet som systemarkitektur 1.

Samlingsbasen

I denne arkitekturen vil samlingsbasen være bærende for de personlige samlingene. De vil inneholde koblingene mellom metadataposter og brukere, og personlig metadata for hver bruker. En post i denne basen ser ut som i Figur 5–7. Posten har en BrukerID som identifiserer brukeren som har laget denne personlige samlingen. Deretter følger en eller flere poster som inneholder en URI til metadataposten, og en personlig metadatapost som gjelder denne URI'en. Siden metadata er lagret distribuert på andre systemer, må hver metadatapost identifiseres med en globalt unik identifikator. Noen søkesystemer, som f.eks. ACM, støtter dette i dag ved hjelp av DOI. For andre systemer som ikke har entydige, persistente identifikatorer kan dette bli en utfordring å få til. Det vil si at visning av en komplett post innebærer et kall til hovedsystemet (f.eks. ACM) der metadata ligger lagret, samt et oppslag i samlingsbasen.

Bruker ID

Metadata post URI
Personlig metadata
om denne posten
Bokhyllemerke

Metadata post URI
Personlig metadata
om denne posten
Bokhyllemerke

Figur 5–7: Metadatapost for samlinger i systemarkitektur 2

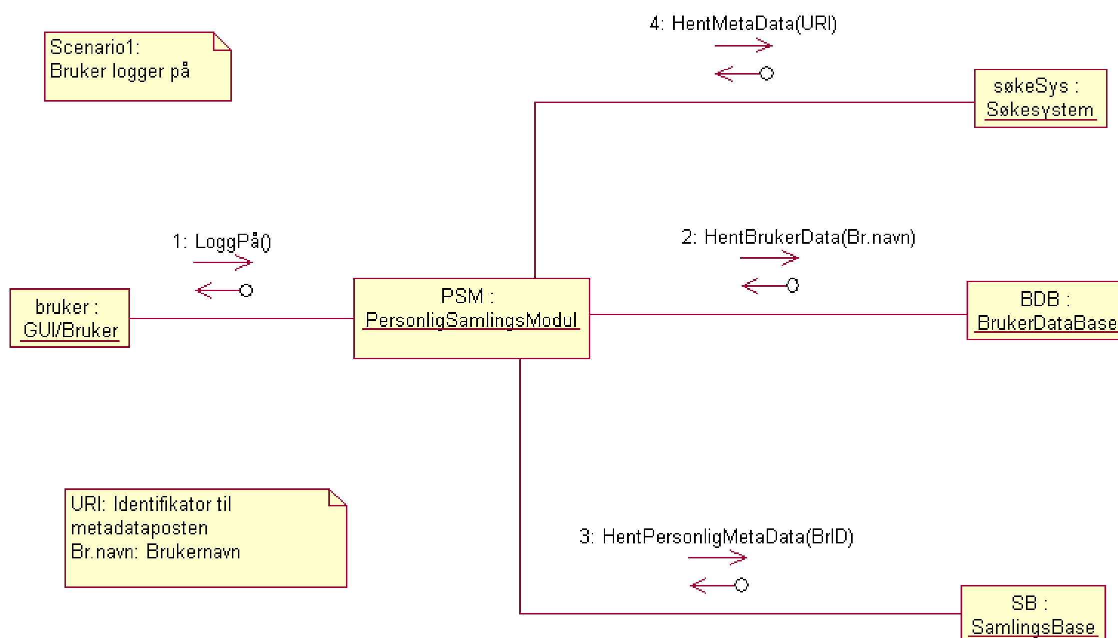
For at man skal kunne skille ut de postene som skal være med i en brukers personlige bokhylle, fra de postene brukeren kun har kommentert, må det registreres et merke på de postene i samlingsbasen som skal tilhøre en personlig samling.

Scenario av systemarkitektur 2 i bruk

Det er få forskjeller mellom dette scenariet og det foregående når det gjelder interaksjonen mellom søkeren og systemet.

Vi antar her som tidligere at søkeren har et informasjonsbehov som er definert, og at han velger å undersøke den personlige samlingen. Søkeren logger på samlingstjenesten, og får se sin egen personlige samling.

Den personlige samlingtjenesten får tilsendt en forespørsel fra brukergrensesnittet om at en bruker med et gitt brukernavn vil logge seg på (sees som kall 2 i Figur 5–8). Deretter sendes brukernavnet videre til Brukerdatabasen for å returnere brukerinformasjon om den gitte brukeren (kall 2 i figuren). Deretter brukes BrukerID som parameter inn til samlingsbasen for å hente alle poster som er registrert med den gitte BrukerID'en (kall 3 i figuren). Dette gir poster med personlig metadata i retur, og disse postene har en URI til den metadataposten som den personlige posten er en kommentar til. Samlingstjenesten bruker så denne URI'en til å hente metadataposten på det eksterne søkesystemet (kall 4 i figuren). Når samlingstjenesten har hentet alle metadataposter både fra den personlige samlingen og de eksterne tjenestene, lages det en samlet post som sendes tilbake til brukergrensesnittet for videre bearbeiding og presentasjon til søkeren.



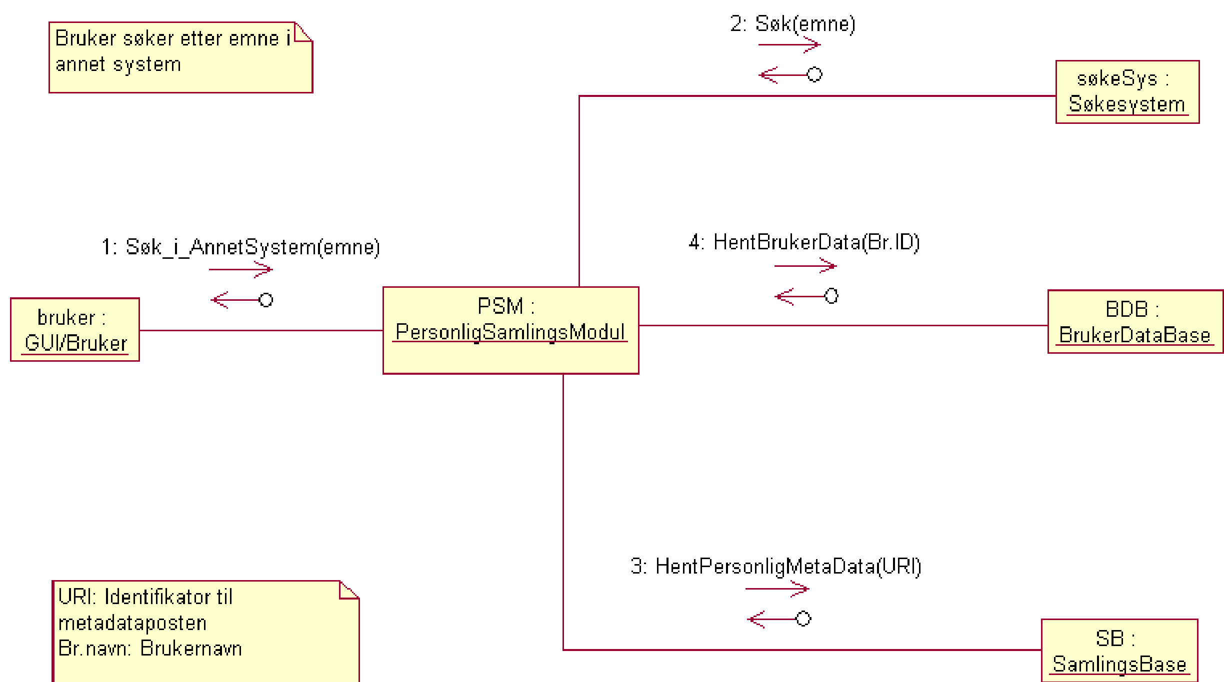
Figur 5–8: Scenario 1, Systemarkitektur 2

Søkeren undersøker så om objektene i bokhyllen gir svar på problemet. Dette innebærer at han undersøker resultatsettet og evt. anskaffer dokumenter for informasjonsuthenting. Etter at søkeren har hentet ut informasjon, kan han avgjøre om informasjonsbehovet er stilt, eller om det må foretas flere søk.

Det antas at informasjonsbehovet ikke er stilt etter søket i den personlige samlingen. Søkeren velger så å søke etter emner i metadata generelt. Brukeren velger da søkefunksjonen i brukergrensesnittet, angir søketermen, og sender denne til søkesystemet.

Den personlige samlingstjenesten mottar søket fra søkeren (kall 1 i Figur 5–9) via brukergrensesnittet, og videreformidler dette først til det eksterne søkesystemet (kall 2 i figuren). Som retur på dette søket får den personlige samlingstjenesten metadataposter med bl.a. en metadatapost-identifikator. Denne identifikatoren blir så brukt av den personlige samlingstjenesten som parameter i et kall til samlingsbasen der de personlige metadata om de aktuelle postene ligger lagret (kall 3 i figuren). Deretter hentes BrukerID'ene fra de personlige metadatapostene, og disse brukes som parametere i kall til brukerdatabasen for å hente ut informasjon om de brukerne som har kommentert de metadatapostene som ble funnet i det første søket (kall 2). Til slutt lages det en datapost av all denne metadataen som sendes til brukergrensesnittet og videre til søkeren.

Når brukeren får resultatet av søket, må han bearbeide dette og på bakgrunn av informasjonen i søkesettet avgjøre hva han skal gjøre videre. I Marchioninis modell for søking i elektroniske systemer har søkeren flere valg. Ett av dem er å undersøke dokumentene som noen av resultatpostene peker på, og deretter avgjøre om dette er nok.



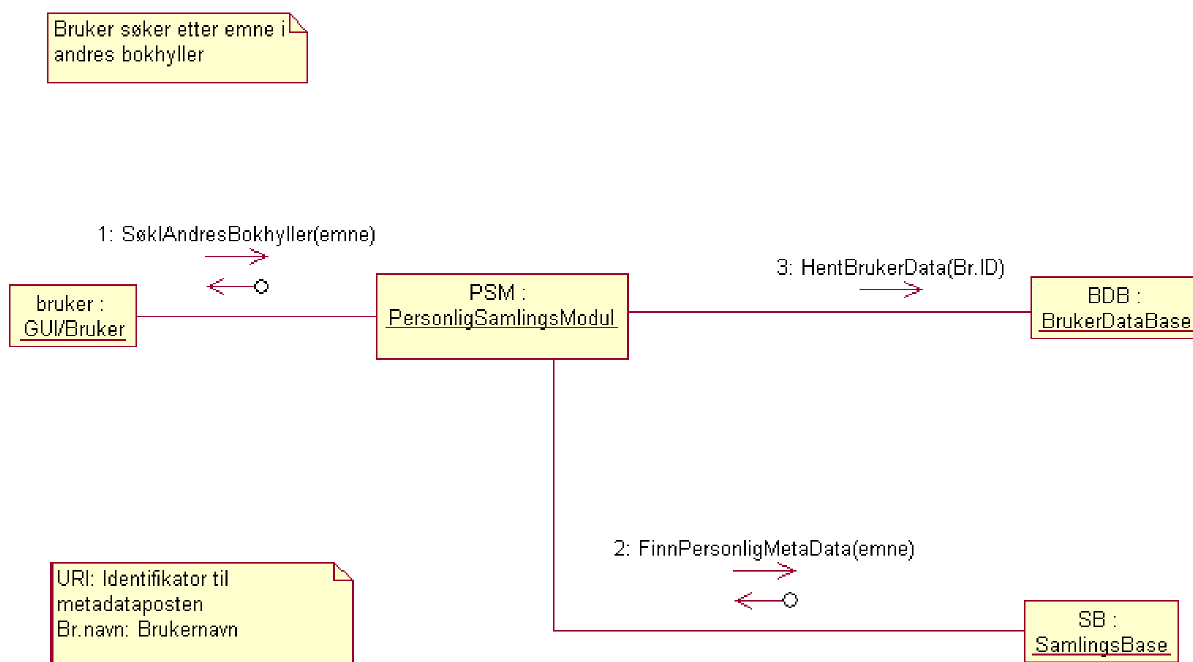
Figur 5–9: Scenario 2, Systemarkitektur 2

Finner søkeren ut at informasjonsbehovet ikke ble dekt av de foregående søkene, kan han utføre et søk i personlige metadataposter. Dette vil muligens gi et noe bredere grunnlag å trekke slutninger på, og kan lede til ellers utilgjengelig materiale. Et søk i personlig metadata sendes av brukeren som emneord fra brukergrensesnittet til den personlige samlingstjenesten. Dette ser man som kall 1 i Figur 5–10. Den personlige samlingstjenesten sender så en forespørsel til samlingsbasen etter poster som har de angitte emneordene registrert (kall 2 i figuren). Som retur får samlingstjenesten en rekke metadataposter som bl.a. inneholder bruker-identifikatorer til de brukerne som har laget postene. Identifikatoren sendes til brukerdatabasen, og brukerdata om disse brukerne sendes i retur til tjenesten (kall 3 i figuren). Til slutt lages det en post som sendes tilbake til brukergrensesnittet og videre derfra til søkeren.

Basert på dette resultatsettet kan brukeren fortsette informasjonssøkeprosessen med f.eks. å trekke ut informasjon fra de dokumentene som ble funnet. Det er mulig løsningen på informasjonsproblemet ligger i de resultatene som ble funnet. Det er også mulig at søkeren ikke fant det han lette etter, og da må søket fortsette. Dette kan resultere i reformuleringer av søkene i dette søkesystemet, valg av andre søkesystemer, eller en mer radikal endring i problemet.

Fordeler og ulemper

Systemarkitektur 2 er som nevnt avhengig av andre systemer for å få tilgang til metadata. Nettopp denne distribuerte lagringen av metadata vil føre til at systemet kan bli tregere enn om all data lå lagret på den maskinen som har de personlige samlingene og aksesskontrollen. Ved å implementere denne arkitekturen som et søkesystem for mange forskjellige søketjenester, og i tillegg



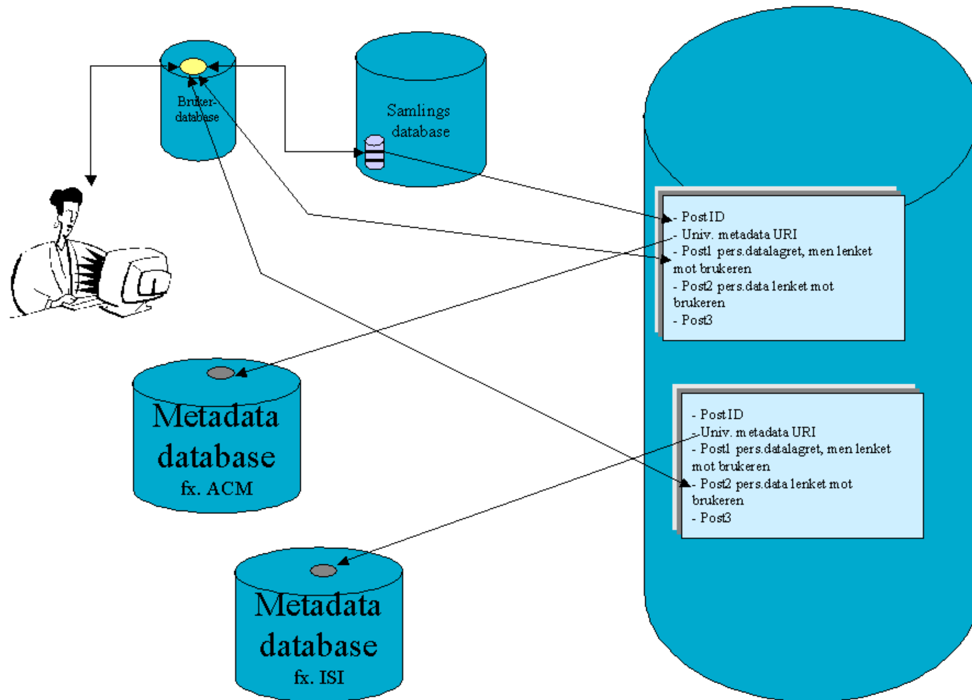
Figur 5–10: Scenario 3, systemarkitektur 2

tilby de personlige samlingene, kan dette føre til at systemet vil oppleves som tregt. For å unngå dette kreves god kommunikasjon til denne tjenermaskinens omverden. Distribuert lagring av metadata er samtidig den største fordelen i forhold til systemarkitektur 1, da all metadata vil være oppdatert. Det vil også kreves betydelig mindre lagringsplass på maskinen som skal kjøre den personlige samlingstjenesten.

Den personlige samlingen må hentes fram ved å søke i hele samlingsbasen etter poster som har riktig bruker-id og et merke som indikerer at denne posten er en del av en personlig samling. Dette kan være en treg prosess om samlingsbasen er stor.

5.1.4 Systemarkitektur 3

Systemarkitektur 3 er en variant av systemarkitektur 2 der det benyttes tre databaser for å håndtere personlige samlinger, brukere og personlig metadata. Brukerdatabasen er her, som i de andre arkitekturene, en selvstendig enhet som kan byttes ut om nødvendig. Samlingsbasens rolle er å ta vare på de personlige samlingene. Her lagres bare referanser til de postene som inngår i brukerens personlige samling, og brukerens id, som referanse-par. Dette gjør at man kan ta vare på referanser til poster som en bruker ønsker skal inngå i en personlig samling, uten at man trenger å lagre dette i en metadatapost eller lignende.



Figur 5–11: Systemarkitektur 3

Den lokale metadatabasen er lager for personlig metadata fra alle brukere. I en post i den lokale metadatabasen lagres det personlig metadata for hver objektive metadatapost. Et eksempel på en slik post kan sees i Figur 5–12. En post i denne arkitekturen vil inneholde en URI til en metadatapost, en bruker-id, og personlige metadata som gjelder den aktuelle metadata-posten.

Metadata post URI
Universell metadata
Bruker ID Personlig metadata Bruker ID Personlig metadata Bruker ID Personlig metadata

Figur 5–12: Metadatapost i systemarkitektur 3

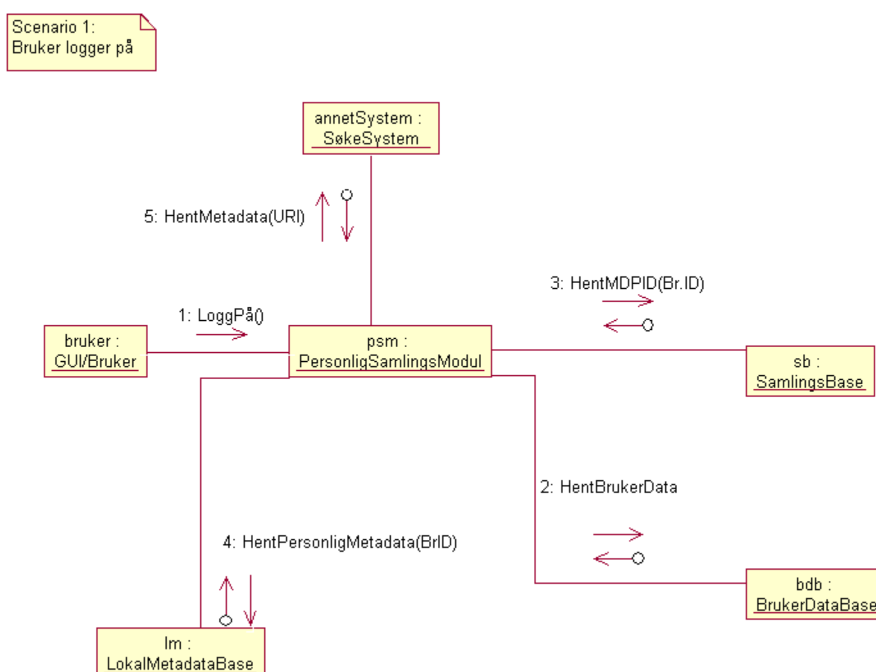
Som helhet kan et system basert på systemarkitektur 3 brukes som et selvstendig søkesystem der man søker i distribuerte systemer, eller man kan bruke det som en ekstra tjeneste som kan legges til i allerede eksisterende søkesystemer. Dette åpner for at systemer forholdsvis enkelt kan implementere støtte for personlige samlinger, og man kan ha en tjener for de personlige samlingene uavhengig av søkesystem. Man kan altså få tilgang til sin personlige samling fra flere søkesystemer, og har tilgang til den samme personlige samlingen hver gang.

Scenarier i systemarkitektur 3

Som i de tidligere scenariene tas det utgangspunkt i et definert informasjonsproblem, der valget av søkesystem har falt på den personlige samlingstjenesten. For variasjonens skyld antas det i dette scenariet at systemet er en

tilleggstjeneste i et annet søkesystem, som gjør søkerens interaksjon med systemet litt annerledes enn i de to foregående systemene.

Til å begynne med må brukeren logge på søkesystemet. Det antas at dette systemet har en måte å sende en innlogging til den personlige samlingstjenesten på. Siden brukere primært er brukere av et vanlig søkesystem, vil ikke den personlige samlingen bli vist før brukeren ber om det. Når søkeren ber om å få se sin personlige samling, sendes det en forespørsel til den personlige samlingstjenesten om å hente fram en personlig samling basert på brukernavnet til søkeren. Dette ser man som kall 1 i Figur 5–8. Samlingstjenesten henter deretter bruker-



Figur 5–13: Scenario 1, Systemarkitektur 3.

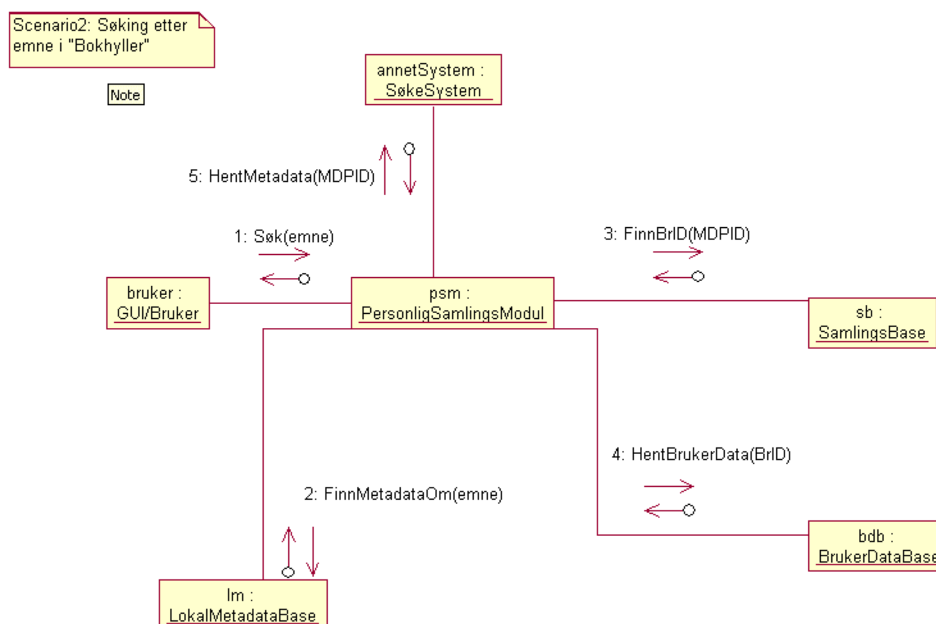
data fra brukerdatabasen basert på dette brukernavnet (kall 2 i figuren). Deretter henter den fram brukeridentifikatoren fra denne posten, og deretter alle poster fra samlingsbasen som har den aktuelle bruker-ID'en (kall 2 i figuren). Postene som returneres fra samlingsbasen, inneholder data om hvilke metadataposter i den lokale metadatabasen som skal inngå i den aktuelle personlige samlingen. Tjenesten henter så fram alle disse postene (kall 4 i figuren). Deretter henter tjenesten fram metadata fra de eksterne søkesystemene, basert på metadatapost-URI'en registrert på hver lokale metadatapost. Til slutt lages det en post som sendes til brukergrensesnittet, som presenterer den personlige samlingen for brukeren.

Brukeren kan nå undersøke om postene i den personlige samlingen gir løsninger på det definerte problemet. Søkeren må først sjekke resultatsettet (den personlige samlingen), og deretter sjekke selve dokumentene. Ble ikke informasjonsbehovet dekt av dette første søket, kan søkeren velge å fortsette søket.

Søkeren kan nå f.eks. velge å søke etter emneord i andres personlige samlinger. Dette gjøres i søkesystemet, som må implementere et grensesnitt for dette. Søkeren angir emneordene, og sender disse til søkesystemet. Søkesystemet sender så dette til den personlige samlingstjenesten, som behandler søket videre (kall 1 i Figur 5–13). Emneordene sendes til den lokale metadatabasen som inneholder de personlige metadatapostene (kall 2 i figuren). Basen returnerer alle poster som tilfredsstiller søkekriteriene, og samlingstjenesten bruker metadatapost-identifikatorene som søkekriterium for oppslag i samlingsbasen etter brukeridentifikatorer (kall 3 i figuren). Bruker-id'ene blir sendt som parametere til brukerdatabasen for å hente fram brukerinformasjon om brukerne som har de aktuelle postene (kall 4 i figuren). Den objektive metadataposten blir hentet på bakgrunn av metadatapost-URI'en som ligger i den personlige metadataposten. Til slutt lager tjenesten en post som sendes til søkesystemet og brukergrensesnittet for presentasjon til søkeren.

Basert på informasjon i den personlige samlingen må brukeren trekke ut informasjon ved f.eks. å lese dokumentene, og ta en beslutning om søkeprosessen skal opphøre eller fortsette. Om brukeren ønsker å fortsette søkeprosessen i de personlige samlingene, kan han f.eks. utføre et emnesøk.

For å få til dette må søkeren finne riktig funksjon i brukergrensesnittet, og der skrive emneordet han ønsker å søke på. Brukergrensesnittet sender så emneor-



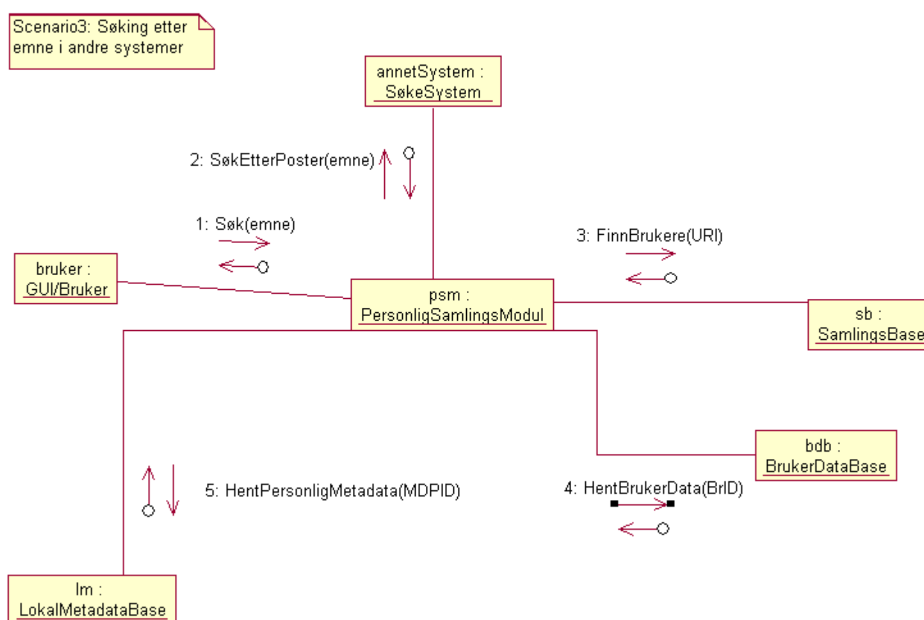
Figur 5–14: Scenario 2, Systemarkitektur 3.

det til den personlige samlingstjenesten (kall 1 i Figur 5–14), som sender hvert emneord (om det er flere) til den lokale metadatabasen (kall 2 i Figur 5–14). Den lokale metadatabasen sender så de resulterende postene tilbake til samlingstjenesten, og den henter ut metadatapost-ID'en. Denne sendes til samlingsbasen, for å returnere bruker-id'er til alle brukere som har kommentert den gitte posten (kall 3 i figuren). Til slutt sendes denne bruker-ID'en til bru-

kerdatabasen for å returnere brukerinformasjon om de aktuelle brukerne. Deretter hentes objektiv metadata om de aktuelle postene fra eksterne kilder. Det hele settes sammen til en post og returneres til søkeren.

På nytt vil søkeren få en resultatliste med objektiv metadata, personlig metadata og brukerinformasjon. Han kan så begynne å undersøke resultatlisten, og trekke ut informasjon. Til slutt må brukeren igjen avgjøre om målet for søkeinnsatsen er nådd.

Ved å snu på scenariet ovenfor kan man søke i andre søkesystemer, og deretter framhente personlig informasjon om det man der finner. Ved at søkeren utfø-



Figur 5–15: Scenario 3, Systemarkitektur 3.

rer et vanlig søk i søkesystemet, vil søket først gå til de eksterne kildene (kall 1 i Figur 5–15). Disse kildene vil gi et søkeresultat, og postene har identifikatorer som brukes som parametere mot kall mot samlingsbasen på jakt etter brukere som har kommentert den aktuelle metadataposten (kall 3 i Figur 5–15). Dette kallet gir bruker-ID'er til disse brukerne, og disse brukes som parametere til brukerdatabasen for å hente ut brukerinformasjon om brukerne (kall 4 i figuren), og mot den lokale samlingsbasen for å hente de personlige metadatapostene (kall 5 i figuren). Det hele vil så bli sendt til brukeren på en fornuftig måte.

Bearbeidingen av dette resultatsettet er ikke annerledes enn det som skjer ved forrige søk.

Fordeler og ulemper med systemarkitektur 3

Mange av fordelene og ulempene med denne arkitekturen er de samme som fordelene og ulempene nevnt for de andre systemarkitekturene. Denne arkitekturen bruker en ekstern metadata-tjeneste på samme måte som systemarkitek-

tur 2. Fordelen med dette er den samme her som der: oppdaterings-problematikken. Vi har alltid tilgang til de sist oppdaterte metadata. Dette krever heller ikke mye lagringsplass på tjenermaskinen som kjører den personlige samlingstjenesten. Dette er som nevnt heller ikke bare positivt. Man er avhengig av disse tjenestene for å få tilgang til metadata. Tilgangen er også avhengig av nettverket mellom samlingstjenesten og metadatatjenesten. Dette setter også en liten bremse på bruken av den personlige bokhyllen. I scenario 1 forsøker søkeren å få tilgang til sin personlige samling, og dette kan bli tregt om nettverket er tregt. Denne funksjonen er nemlig også avhengig av de eksterne metadata.

5.2 Integrasjon med andre systemer

Ovenfor er tre forskjellige systemarkitekturer gjennomgått. De har forskjellige styrker og svakheter, og disse forskjellene gjør det lettere å implementere forskjellige typer systemer som ble beskrevet i kapittel 4.6.

Den første typen system, der all søking og bruk foregår i ett sentralisert system, er best ivaretatt av systemarkitektur 1. Denne arkitekturen legger opp til at all data lagres lokalt i tjenesten, og dette passer bra med et komplett system som også inneholder all data. Dette vil som nevnt i kapittel 4.6 kreve at brukerne bare får tilgang til den personlige samlingen sin i ett system.

Den andre typen system er en personlig samlingstjeneste som en tilleggstjeneste i et annet søkesøkesystem. Dette støttes av systemarkitektur 3.

Den siste typen system er et søkesystem der søkesystemet tillater søk mot mange eksterne metadatakilder. Ved å legge til en personlig samlingstjeneste, vil de eksisterende systemene knyttes til personlige metadataposter.

5.3 Valgt systemarkitektur

Basert på fordelene og ulempene med de forskjellige systemarkitekturene og integrasjonen med andre systemer, er det systemarkitektur 3 som passer best til å løse oppgaven som en personlig samlingstjeneste. Med systemarkitektur 3 vil vi få et system som håndterer personlige samlinger i et selvstendig system. Dette systemet kan så gjøres tilgjengelig i andre søkesystemer, og vil integrere de personlige samlingene med de objektive metadata som skaffes av søkesystemet. Brukeren kan fortsette å bruke de søkesystemene har kjenner fra før, og samtidig få tilgang til en personlig samling.

Systemarkitektur 3 er avhengig av andre systemer for å få tilgang til metadatatjenester. Dette oppnås gjennom å bruke et søkesystem som nevnt over. De objektive metadata er åpne for alle endringer eierne av disse metadata gjør, og vil alltid bli presentert i den mest oppdaterte formen. Systemarkitektur 3 kre-

ver i tillegg mye mindre lagringsplass enn spesielt systemarkitektur 1, men også systemarkitektur 2.

Avhengigheten av andre søkesystemer gjør den personlige samlingstjenesten mer sårbart overfor tilgjengelighetsproblemer: Stor nett-trafikk og andre systemers problemer kan ramme tilgangen til den personlige samlingstjenesten. Dette er likevel ikke et så stort problem at det ansees som vesentlig for valget av systemarkitektur 3.

6 SYSTEMDESIGN OG DOKUMENTASJON

En arkitektur for en personalisert samlingstjeneste er presentert i forrige kapittel, og det er naturlig å undersøke om arkitekturen fungerer som tiltenkt. Det er valgt å implementere en pilot av en personlig samlingstjeneste for å undersøke arkitekturen i praksis. Med en implementasjon av arkitekturen kan man også utvide denne, slik at man etter flere implementasjonsfaser har et bedre system for hver gang. Denne måten å utvikle programmer på, gjør at utviklerne tidligere ser om arkitekturen må forbedres.

6.1 Mål for piloten

Hovedmålene for implementasjonen av arkitekturen:

1. Teste arkitekturen for en personlig samlingstjeneste utviklet i forrige kapittel.
2. Undersøke WebServices som en mulig plattform å lage en personlig samlingstjeneste på.

Mindre viktige mål for implementasjonsarbeidet er:

3. Se hvordan en slik tjeneste kan oppføre seg, og hvordan en slik tjeneste kan kobles til andre systemer i praksis.
4. Undersøke bruken av xml-databaser i forbindelse med et digitalt bibliotek.

Design og implementasjon av piloten er i stor grad basert på åpne standarder og gratis programvare. Fordelen med åpne standarder er at det er lett for andre brukere å sette seg inn i applikasjonen og de systemer som er brukt for å utvikle den. Det er videre ikke knyttet utgifter til programvarelisenser til bruken av programsystemet. Programvare med åpen kildekode har også ofte et stort brukermiljø med engasjerte frivillige brukere som kan hjelpe om det oppstår feil ved bruk av programkomponentene. Ulempen med åpen kildekode er at det

ikke er knyttet hjelpetjenester til programvaren man bruker; oppstår det problemer med programvare, er man prisgitt frivillig hjelp.

Systemet er utviklet i Java, da dette er et programmeringsspråk som er kjent for utvikleren, og som er meget lett å ta i bruk i de komponentene systemet er bygd opp av. Videre er det mange gratissystemer som ble brukt i utviklingen av piloten. Disse komponentene vil bli forklart nærmere nedenfor.

På grunn av praktiske hensyn og arbeidsmengden er piloten kun implementert i en begrenset grad. Den funksjonaliteten som er implementert, vil bli forklart nedenfor.

Videre i avhandlingen vil de teknologiske komponentene drøftes. Deretter vil implementasjonen av den personlige samlingstjenesten presenteres.

6.2 Teknologisk bakgrunn

Allerede tidlig i designfasen ble det aktuelt å teste ut WebService[52] arkitekturen i implementeringen av den personlige samlingstjenesten. WebService-arkitekturen er godt på vei inn i bedriftsmarkedet som en måte for bedrifter å kommunisere til andre bedrifter på via datasystemer, såkalt Business To Business (B2B) kommunikasjon. Denne arkitekturen består av flere komponenter. Når en bedrift har laget en tjeneste som de vil tilby til andre, lager de en beskrivelse av denne tjenesten, og legger beskrivelsen inn i en Universal Discovery and Description Index (UDDI)[47] . Når noen er på jakt etter en tjeneste, gjør de et oppslag i denne UDDI'en, og får beskrivelsen av tjenesten. Denne beskrivelsen bruker de så for å lage meldinger som de kan sende til tjenesten. Et typisk eksempel på en slik måte å bruke WebServices på, er en kaffebar som har et datasystem som bruker en web-service for å bestille ny kaffe. Vi har da en kaffeleverandør som har laget en web-service slik at kunder kan bestille kaffe. Denne tjenesten er beskrevet med en WSDL-fil (WebSeriviceDescriptionLanguage)[53] , og distribuert til en UDDI. Vår kaffebars datasystem vil så finne fram til denne kaffeleverandøren via UDDI'en, og finne beskrivelsen av leverandøren. På bakgrunn av beskrivelsen kan kunden lage en bestillingsmelding basert på SOAP[51] , som så kan sendes til leverandøren.

En av de største fordelene med denne arkitekturen er at den tillater helt forskjellige systemer å kommunisere med hverandre uten å kjenne til annet enn WSDL-beskrivelsen av hverandre. En web-service trenger ikke være så kompleks som en bestillingstjeneste for varer. Det er laget mange forskjellige tjenester, som f.eks. en valutakalulator. Her sender man et beløp, i en gitt valuta og en ønsket returvaluta, til en web-service, og får et returbeløp tilbake. Disse meldingene er også basert på SOAP. Siden alle meldingene som går mellom systemene er systemuavhengige xml-meldinger, betyr det at man kan lage en slik web-service på en hvilken som helst måte på en hvilken som helst datamaskin, bare den følger standarden. En tjeneste programmert i Java kan godt kalle en tjeneste programmert i Microsofts .Net-system. Dette gjør at tjenesten blir mer og mer uavhengig av operativsystem og programmeringsmiljø.

En WebService er bygd opp av flere komponenter. Til nå er WSDL, UDDI, og SOAP nevnt. WSDL er et språk for å beskrive en web-service. Ved å bruke dette språket, kan man beskrive hvordan meldingene tjenesten kan motta må se ut. UDDI er som nevnt en indeks over web-servicer. Ved å kontakte en UDDI kan man få tak i WSDL-filen for en tjeneste. Dette er som nevnt alt man trenger å vite for å kunne benytte seg av tjenesten. Meldingene man sender mellom web-services, er SOAP-meldinger. Disse meldingene er skrevet i xml-syntaks, og er underlagt de regler som gjelder xml-dokumenter. Fordelen med å benytte xml er at det er plattform-uavhengig og dermed kan leses av de fleste.

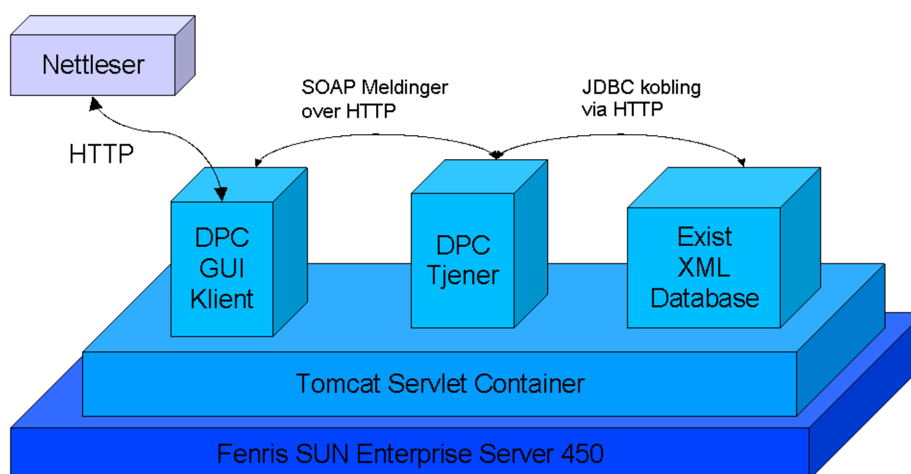
For å bygge programsystemet bak en WebService, kan man benytte flere plattformer. I hovedsak er det Microsofts .Net-plattform[38] eller JAVA [44] som benyttes. Det er også mulig å lage WebServices i andre programmeringsmiljøer som f.eks. PERL[45] . Det eneste kravet er at det skal kunne motta og sende SOAP-meldingene i xml-formatet. I Java-plattformen er det spesielt to komponenter som er verdt å nevnt i denne sammenheng: Servlets og Servletcontainer.

Servletcontaineren er et programsystem som kjøres som en server på en maskin som lar serveren benytte servlets for å tilby tjenester. Det finnes flere servletcontainere, men den mest utbredte gratisprogramvaren er Tomcat fra Apache-prosjektet[46] . En servlet er et lite program som betjener forespørsler fra Internett. Servlets er et alternativ til cgi-skript og andre teknologier som lar program vise informasjon på Internett.

6.3 Beskrivelse av piloten

Programsystemet til den personlige samlingstjenesten er laget som en nett-tjeneste som kan brukes av andre søkesystemer. Systemet er bygd opp av tre komponenter. I bunnen har vi en database som fungerer som lager for brukerdata, personlige samlinger, og personlig metadata. Over databasen er det et applikasjonslag som fungerer som en tjener som kobles til databasen og tilbyr tjenester for å hente og lagre data i databasen. Mellom databasen og applikasjonslaget foregår kommunikasjonene ved hjelp at JDBC. Øverst har vi en klient som tilbyr mulighet for å hente fram data for presentasjon i et nettleservindu. Kommunikasjonen mellom klienten og applikasjonslaget foregår ved hjelp at SOAP-meldinger. Dette kan man se skjematisk i Figur 6–1. Som det går fram av figuren kjører hele applikasjonen på en datamaskin, og siden kommunikasjonen mellom komponentene ikke avhenger av denne maskinen, men går over Internett, vil det være mulig å flytte komponentene til forskjellige maskiner.

De overnevnte komponentene blir forklart og drøftet nedenfor.



Figur 6–1: Systemkomponenter i Den Personlige Samlingstjenesten

6.3.1 Sluttbrukertjenester

Sluttbrukertjenesten er den tjenesten som brukeren har kontakt med. Denne ser man som “DPC GUI Klient” til høyre på Figur 6–1. Av praktiske årsaker er det ikke laget noe grafisk grensesnitt mot sluttbrukertjenestene, da dette ikke er nødvendig for å teste ut arkitekturen ved hjelp av piloten.

Den funksjonaliteten som tilbys av samlingstjenesten i piloten, er visning av personlige samlinger for en gitt bruker og søking på emner i metadata i de personlige metadatapostene. Videre er det mulig å vise personlige metadataposter for en gitt identifikator.

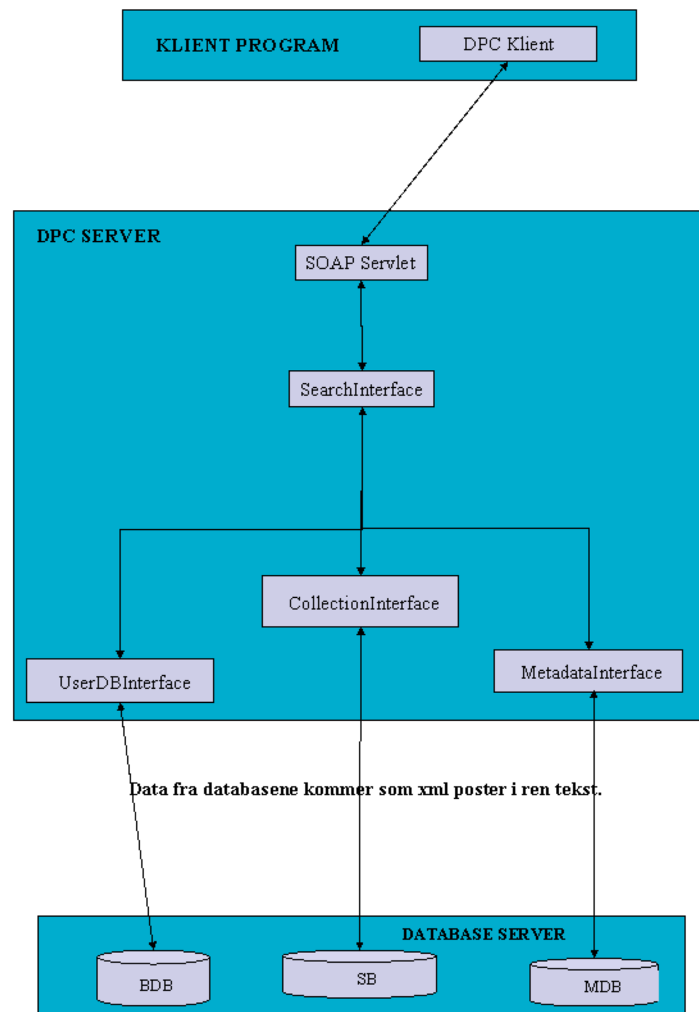
Sluttbrukertjenestene som er implementert i forbindelse med denne avhandlingen, er kun implementert som en servlet som mottar parametere via URL’en brukeren skriver inn. Ved å angi en kommando til Sluttbruker-servleten, vil denne generere en SOAP-melding som sendes til DPCServleten, og som deretter prosesserer denne i henhold til kommandoen angitt i SOAP-meldingen. En slik URL kan se ut som følger:

http://<maskinnavn>/dpc/getUser?uid=ntnu216247

Denne URL’en vil hente fram en post fra DPC-systemet som viser en bruker med bruker-ID’en gitt med parameteret uid.

6.3.2 Samlingstjenesten

Selve samlingstjenesten består av flere deler i henhold til arkitekturen. En figur av dette kan sees sentralt i Figur 6–2 markert i boksen “DPC SERVER”. Nærmest databasen ligger det tre komponenter som håndterer kommunikasjonen mellom selve samlingstjenesten og databasene: UserDbInterface, CollectionDBInterface og MetadataDBInterface. Disse tre komponentene er implementert som tre klasser, og disse har metoder for å hente ut og lagre data i databasene. Dette er funksjoner som f.eks. GetUser som henter brukerinformasjon om en

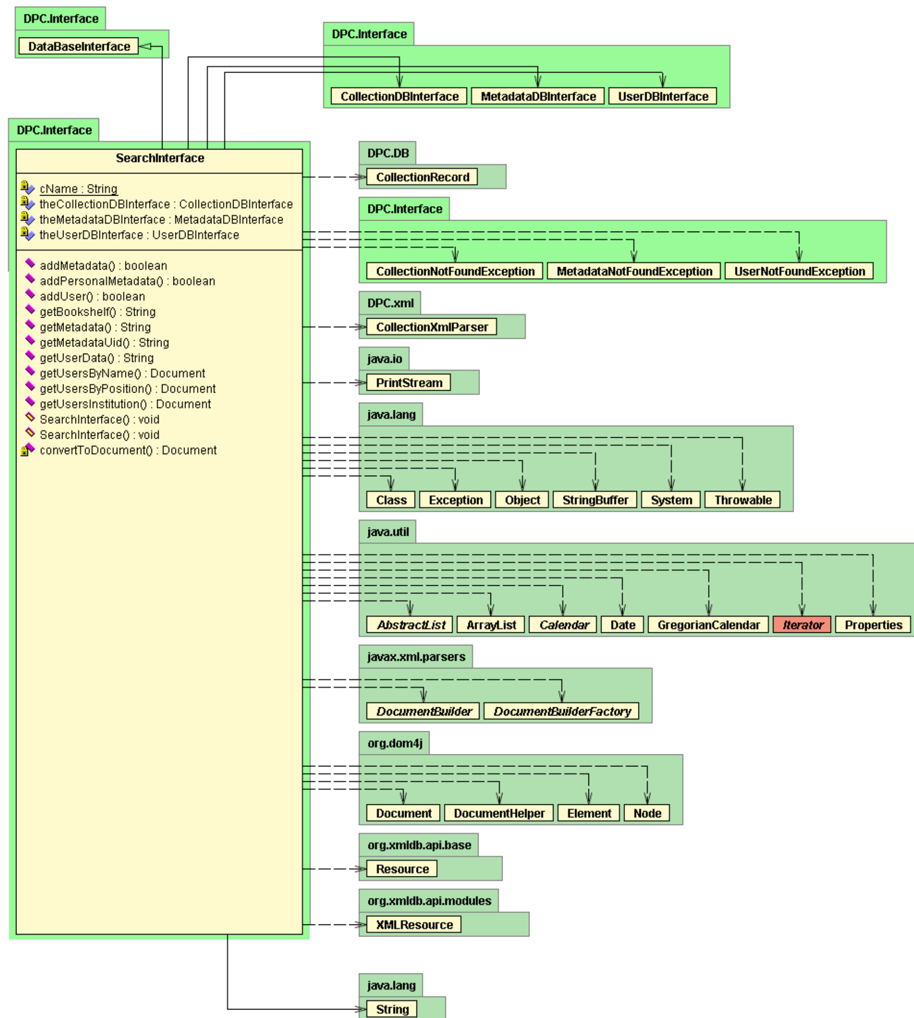


Figur 6–2: Systemdesign for den personlige samlingstjenesten.

bruker, eller GetMetadata som henter personlig metadata etter et gitt kriterium. En oversikt over klassediagrammer for klassene i systemet finnes i appendiks E. En oversikt over de spesifikke metodene kan sees i appendiks E. Disse tre klassene arver en del grunnleggende funksjonalitet fra en DatabaseInterface-klasse som også inneholder informasjon om adresse til database-maskinen (se Figur 6–3). Hver enkelt av kommunikasjonsklassene har selv oversikt over hvor i databasen dataene de skal ha tak i, ligger lagret. Dette gjør at det er enklere å flytte plasseringen av dataene i databasen.

I laget over databaseaksesskomponentene ligger en programkomponent (SearchInterface) som håndterer søkene som går på tvers av databaser. Når en bruker ber om å få se sin personlige bokhylle, er det mange komponenter som håndterer denne meldingen. Dette søkegrensesnittet har metoder for å hente fram typer data som brukerne ber om. Dette er f.eks. GetBookshelf som henter fram en personlig samling for en gitt bruker.

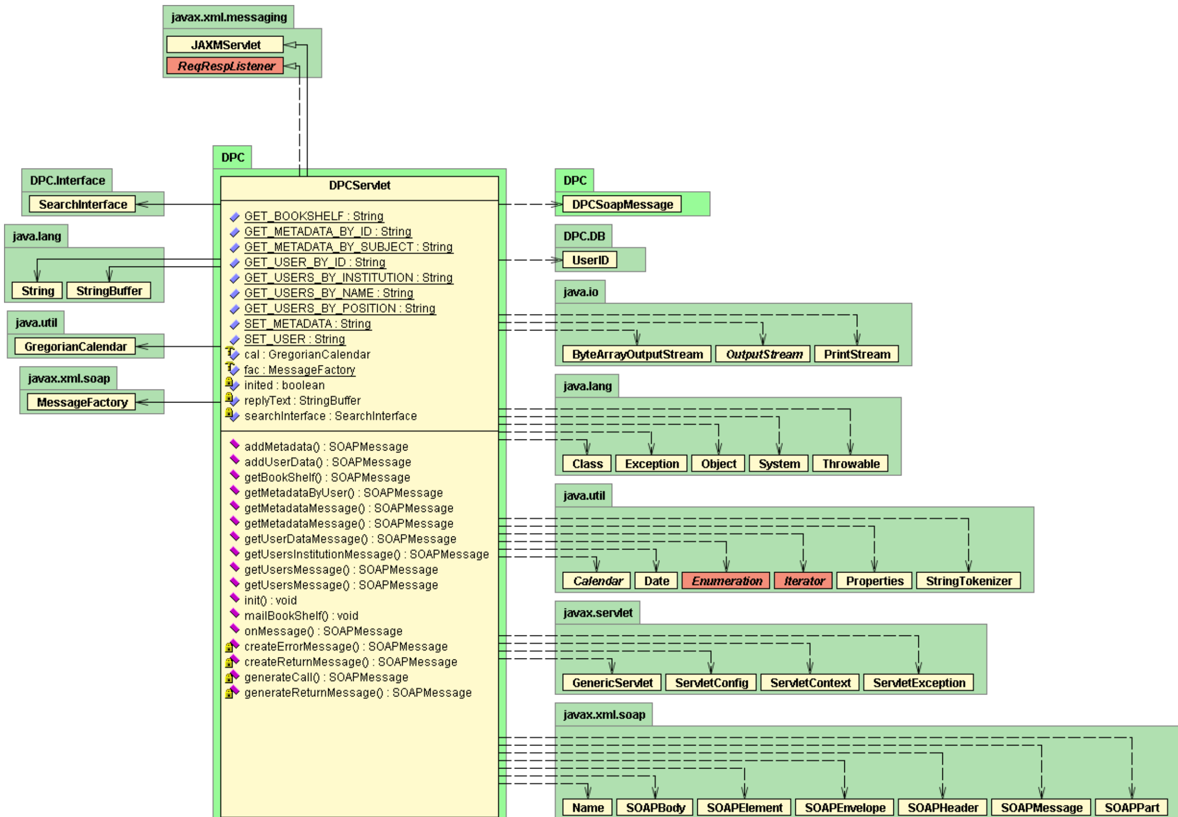
Alle andre søk til systemet går også gjennom denne komponenten. SearchInterface-klasse har koblinger til de forskjellige kommunikasjonskomponentene, slik at SearchInterface-klassen kan bruke dem for å hente ut de data den trenger. Dette er vist i Figur 6–3 på side 94. Dette er vist som en kobling fra Sear-



Figur 6–3: UML diagram for SearchInterface klassen.

chInterface til UserDBInterface, CollectionDBInterface og MetadataDBInterface.

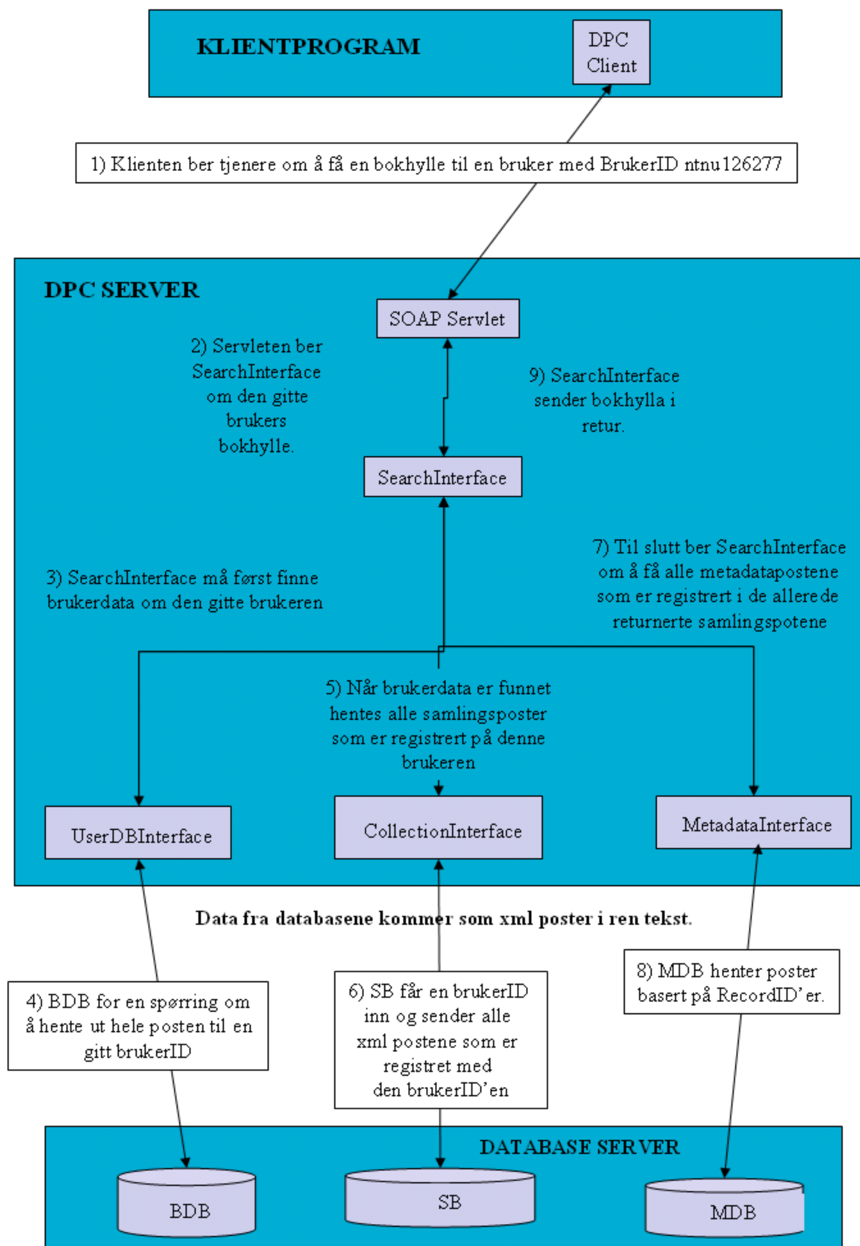
Over det ovenfor nevnte laget i applikasjonen finnes en servlet som håndterer kommunikasjon mot søketjenestene, som f.eks. digitale bibliotekstjenester. Denne komponenten mottar SOAP-meldinger fra de eksterne tjenestene, for deretter å konvertere disse til interne datastrukturer og metodekall i søkegrensesnittet (SearchInterface). Når søkegrensesnittet returnerer et svar, blir dette pakket inn i en SOAP-melding og sendt tilbake til brukeren. Servleten er implementert i klassen DPCServlet. Denne klassen har kun tilgang til SearchInterface-klassen, og har dermed ikke direkte kontakt med databasen. Dette gjør det lettere å løsrive applikasjonen fra databasene. Se Figur 6–4 på side 95 der klassediagrammet viser at DPCServlet ikke har tilgang til databasene i det hele tatt.



Figur 6–4: Klassediagram for DPCServlet.

Et scenario for hvordan systemet fungerer er beskrevet i systemarkitekturen i kapittel 5, og programkomponentene vil håndtere dette som vist i Figur 6–5. I denne figuren ser man at brukeren starter med å bruke et klientprogram som har en programkomponent som kan kommunisere med den personlige samlingstjenesten. Denne komponenten lager en SOAPmelding som inneholder en kommando og et parameter. Disse verdiene blir så sendt til DPC Serveren. SOAPServleten i DPC Serveren henter så ut dataene fra SOAPmeldinger og finner ut hva klienten ber om. SearchInterface-komponenten blir så bedt om å hente ut den gitte brukerens bokhylle. SearchInterace'et setter deretter i gang en rekke kall:

- UserDBInterface blir bedt om å hente brukerdata for den angitte BrukerID'en.
- UserInterface'et ber deretter Brukerdatabasen om de spesifiserte data, og brukerdatabasen returnerer dette.
- CollectionInterface blir deretter bedt om å hente fram de postene som angår den spesifiserte BrukerID'en.
- CollectionInterface ber deretter Samligsbasen om å få alle postene som angår den gitte brukeren, og Samlingsbasen returnerer disse postene.
- SearchInterface henter ut de MetadatPostID'ene som finnes i data returnert fra Samlingsbasen.

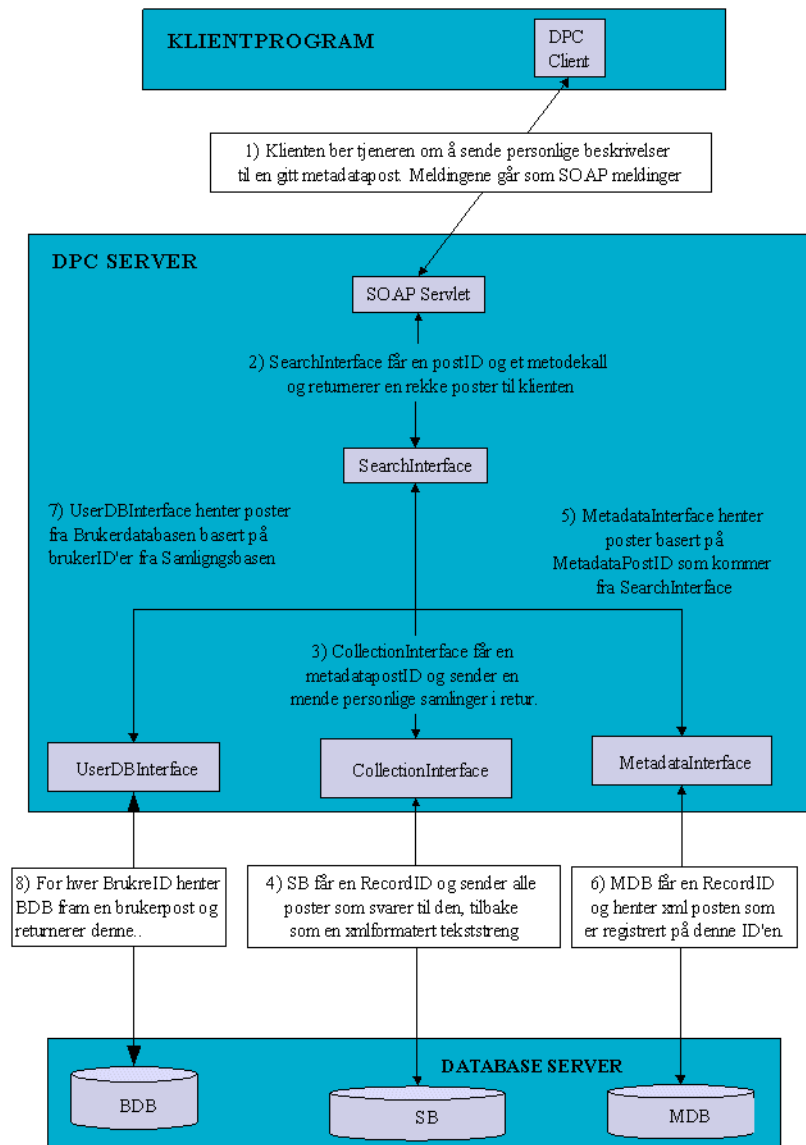


Figur 6–5: Programsystemets håndtering i forbindelse med framhenting av en personlige bokhylle/samling.

- MetadataInterface blir bedt om å hente fram data om de angitte MetadataPostID'ene.
- MetadataInterface ber deretter Metadatabasen om de gitte postene, og returnerer dem tilbake til SearchInterface.

SearchInterface har nå alle data om den gitte brukeren og om de postene som befinner seg i dennes personlige samling. SearchInterface bygger nå opp en post av disse dataene og sender dem tilbake til SOAPServleten, som igjen sender dem tilbake til klienten som presenterer dem for brukeren.

I et annet scenario ber den klientprogrammet om personlig metadata for en gitt post. Det som skjer da er følgende:



Figur 6–6: Programsystemets håndtering i forbindelse med framhenting av personlige metadataposter for en gitt objektiv metadatapost.

- Klientprogrammet sender en SOAPmelding med identifikator for den aktuelle metadataposten, og SOAPServleten undersøker meldingen finner fram data.
- SOAPServlet SearchInterface om å hente fram alle personlige metadataposter som har den gitt ID'en.
- SearchInterface ber CollectionInterface om en alle Bruker ID'er som har registrert den gitte MetadatapostID'en.
- SearchInterface ber UserDBInterface om å få alle brukere som har de overnevnte BrukerID'ene.

- SearchInterface ber MetadataInterface om å få alle poster som er registrert på den gitte MetadatapostID'en.
- MetadataInterface ber databasen om de riktige postene, og returnerer dem deretter til SearchInterface.
- SearchInterface setter deretter opp en post av de gitte metadatapostene og brukerpostene, og returnerer dem til SOAPServlet'en som sender dem som en melding tilbake til klientprogrammet.

6.3.3 Databasen

Utgangspunktet for databasevalget i samlingstjenesten er at det skal tas vare på tre typer data: Personlige Samlinger, Personlig Metadata, og Bruker-informasjon. Det er ingen krav til ytelse for databasen, og en liten pilot som denne har ingen store krav til antallet brukere. Det er flere måter å implementere en database på. Det kunne benyttes en sql-database som for eksempel MySQL, eller det kunne brukes en xml-database som for eksempel XIndice eller Exist. Fordelen med å bruke en xml-database er at man lagrer dokumenter i xml, og man skal i denne tjenesten også videresende xml-dokumenter. Dessuten er det interessant å se om en xml-database kan fungere i denne sammenhengen. Xml-databaser har ikke vært i bruk i så lang tid som SQL-databaser, og derfor er det interessant å se om nye xml-databaser fungerer tilfredsstillende for et system som en Personlig Samlingstjeneste. Valget falt dermed på en xml-database.

Det er forsøkt brukt to xml-databaser. XIndice fra Apache-prosjektet ble forsøkt benyttet først, men den viste seg å være vanskelig å få til å fungere korrekt. Problemene med Xindice var å få den til å fungere sammen med programkoden som kjører den Personlige Samlingstjenesten. Det var vanskelig å få kontakt med databasen, og det var vanskelig å finne informasjon som kunne hjelpe på problemet. Siden Xindice er gratis programvare, kan man ikke kreve å få støttetjenester til installasjon og lignende. I stedet for å bruke mye tid på å få Xindice til å fungere som tiltenkt, ble et alternativt databasesystem, Exist, prøvd ut. Dette er gratis tilgjengelig på samme måte som XIndice, men ikke en del av Apache-prosjektet. Exist-databasen kjører som en tjeneste tilgjengelig via nettverket, og kan aksesseres fra hvor som helst i verden hvis man har adressen til den. Fordi denne basen er tilgjengelig fra nettet, kan den også installeres hvor som helst, og være tilgjengelig for den personlige samlingstjenesten.

Exist har flere kommunikasjonsalternativer. Ett av disse er via JDBC. JDBC er et grensesnitt mellom Java-programmer og en database. Når databasen implementerer dette grensesnittet, er det lett å kommunisere med databasen fra et Java-program. JDBC gjør det mulig å lage spørringer i Java-systemet som deretter blir sendt via nettverket til databasen, og Exist svarer med å returnere et resultatsett som blir tilgjengelig i Java-programmet som et objekt som inneholder xml-postene fra databasen. Java kan så hente ut xml-postene fra dette objektet, og har dermed fått tilgang til hele xml-dokumentet som ble returnert.

6.4 Oppsummering av teknisk løsning

Den tekniske løsningen av samlingstjenesten baserer seg på noen valg som er tatt forut for implementasjon:

- Det skal lages en nett-tjeneste for samlingstjenesten.
- Det skal brukes WebServices som teknisk plattform.
- Det skal brukes en xml-database for lagring av data.
- Det skal brukes SOAP som kommunikasjon mot tjenesten.

Suksesskriteriene for implementasjonen i et teknisk perspektiv er derfor basert på disse utgangspunktene.

Det lot seg gjøre å lage en nett-tjeneste for en personlig samlingstjeneste, og løsningen er laget med WebServices som teknologisk plattform. Disse målene anses dermed som oppfylt. WebServices er en ny og interessant metode for å lage tjenester som baserer seg på kommunikasjon mellom systemer, og i forhold til kravene som stilles av en Personlig Samlingstjeneste, fyller WebServices rollen på en bra måte. Det er ikke funnet noen problemer ved WebServices som tyder på at dette ikke skal takle rollen i en biblioteksammenheng. SOAP som kommunikasjonsform har heller ikke skapt problemer. Det er imidlertid ikke klart definerte meldingsstrukturer å forholde seg til, så dette må avklares mellom sender og mottaker på forhånd. I den senere tid har Z39.50-protokollen[60] blitt videreutviklet, og Library of Congress har utviklet SRW (Search Retrieve WebService)[31] for å for å søke med. Denne protokollen bygger på Z39.50 og SOAP og WebServices for å gjøre det lettere å tilby interoperabilitet mellom databaser. SRW tilbyr riktignok ikke en update-funksjon slik Z39.50 gjør, så denne funksjonaliteten vil man måtte håndtere på en annen måte. Man kan konkludere med at WebServices som teknologisk plattform virker lovende, ikke bare i handel over Internett, men også som modell for digitale bibliotek. Systemer basert på WebServices er enda forholdsvis umodne, så man ser et forbedringpotensial enn så lenge. Dette ser man endrer seg med utviklingen av SRW.

Det største problemet i forhold til implementeringen av den Personlige Samlingstjenesten har vært xml-databasen. Som nevnt har to xml-databaser blitt forsøkt tatt i bruk. Dette skyldtes at det første databasesystemet (Xindice) ikke ville fungere som planlagt. Problemene med Xindice var at det ikke lot seg gjøre å koble seg til databasen fra de programmene som allerede var utviklet. Dette problemet kunne antakelig latt seg løse med større tålmodighet, men på grunn av manglende støttetjenester i forhold til produktet, ble det forsøkt å prøve en annen database-løsning (Exist). Når det viste seg at Exist fungerte tilfredsstillende uten store tilpasninger, ble det valgt å fortsette å benytte Exist av praktiske grunner.

7 KONKLUSJON

I forrige kapittel ble pilot-implementasjonen av den personlige samlingstjenesten gjennomgått, og det er konkludert med at den teknologiske plattformen systemet er implementert i, ser ut til å fungere som planlagt.

At den teknologiske plattformen fungerer som planlagt, betyr ikke at arkitekturen for personlige samlinger fungerer som antatt. Mål for implementasjonsarbeidet:

1. Teste arkitekturen for en personlig samlingstjeneste.
2. Undersøke WebServices som en mulig plattform å lage en personlig samlingstjeneste på.

Delmål for implementasjonsarbeidet:

3. Se hvordan en personlig samlingstjeneste kan oppføre seg, og hvordan en slik tjeneste kan kobles til andre systemer i praksis.
4. Undersøke bruken av xml-databaser i et digitalt bibliotek.

De teknologiske aspektene av disse målene er allerede diskutert i forrige kapittel, og arkitekturens egnethet i forhold til problemene informasjonssøkere står overfor i dag, er grunnlaget for dette kapitlet.

7.1 Arkitektur for en personlig samlingstjeneste

Hvorvidt arkitekturen fungerer etter intensjonen må evalueres opp mot premissene og kritereiene som er lagt til grunn for arkitekturen. Premissene for arkitekturen er basert på teoriene og erfaringene fra søkeprosessne. Det ble der funnet at systemet skal la informasjonssøkere ta vare på informasjonsobjekter i en personlige samling, men viktigst, at de skal kunne dele kommentarer med andre informasjonssøkere.

Den valgte arkitekturen beskriver et system som er avhengig av andre systemer, for å ha data å behandle. Den personlige samlingstjenesten er definert som en tilleggstjeneste til et tradisjonelt søkesystem som informasjonssøkeren allerede bruker og er kjent med. Ved å bruke allerede eksisterende søkesystemer, vil man ha tilgang til metadata, og dermed trenger man ikke å bygge opp et komplett digitalt bibliotek i tillegg til en personlig samling. Den personlige samlingstjenesten vil dermed bare håndtere informasjon om brukere, deres personlige samlinger, og de kommentarer brukerne har skrevet.

Fordelen med den valgte arkitekturen:

- Systemet er mindre enn om et komplett digitalt bibliotek skulle vært designet i tillegg. Dette gjør at det er enklere å lage en personlig samlingstjeneste.
- De personlige samlingene kan gjøres tilgjengelige i flere søketjenester. Dette fører til at brukerne trenger å forholde seg til bare ett sett data i en personlig samling.
- Systemet er mindre komplekst, og dermed vil også kostnadene med utvikling of vedlikehold være mindre.

For at man skal kunne dra nytte av de personlige samlingene i andre søkesystemer, er man avhengig av identifikatorer, og her møter man noen praktiske hindringer:

- Informasjonsobjekter må identifiseres på tvers av systemer.
- Brukere må identifiseres i søkesystemet så vel som i den personlige samlingstjenesten.

Problemet med identifikasjon er at entydig og persistent identifikasjon av informasjonsobjekter ennå ikke er fullt ut implementert hos informasjonstilbydere, selv om det finnes systemer som The Handle System og DOI. Uten en slik identifikasjon av informasjonsobjekter vil systemet kreve mye av søkesystemene som gir tilgang til informasjonsobjektene, og dette kan føre til at løsningen ikke er praktisk gjennomførbar. Identifikasjonsproblemet gjelder ikke bare informasjonsobjekter, men også brukerne. For at man skal kunne dra nytte av personlige samlinger, må informasjonssøkerne være identifisert på en slik måte at den personlige samlingstjenesten kan identifisere en bruker, og slik at den samme brukeren har tilgang til søkesystemet. En slik bruker-identifikasjon er ikke vanlig i dag. Hvert enkelt system har sin måte å identifisere brukere på.

Antar man at de overfor nevnte problemene er løst, fungerer arkitekturen i forhold til premissene. I en praktisk virkelighet er det ennå et stykke igjen til man kommer i mål med identifikasjonsproblemene, men i dag jobbes det med å utvikle en felles elektronisk identitet i UH sektoren, FEIDE[16] , og dette er et skritt i riktig retning.

7.2 Personlige samlinger i andre systemer

For at en personlig samlingstjeneste skal bli et verdifullt bidrag for informasjonssøkere, må de kunne få tak i sin personlige samling i det søkesystemet de bruker. Fordelen med arkitekturen er at den tillater at flere systemer tilbyr personlige samlinger fra det samme personlige samlingssystemet. Brukerne trenger bare å ha én personlig samling, i stedet for en personlig samling i hvert søkesystem. Ulempen med denne måten å få tilgang til de personlige samlingene på, er at det er opp til de enkelte søkesystemene å implementere støtte for personlige samlinger. For å få til dette er det nødvendig at de som tilbyr søkesystemer, er enige om å ta i bruk et slikt system, og er enige om hvordan dette skal gjøres. Disse problemene kan ikke teknologien løse, men om man finner ut at dette er en god måte å forbedre tjenestene man tilbyr informasjonssøkerne, kan teknologien gjøre det lettere. Åpne standarder kan også hjelpe for å få aksept for løsningen ut over én institusjon.

7.3 Personlige samlinger og brukernes behov

Om personlige samlingstjenester støtter en informasjonssøkers behov, er et av kjernespørsmålene for denne avhandlingen.

Personlige samlinger som de er framstilt her, har mye til felles med bokanmeldelser i nettbokhandler. Populariteten til slike anmeldelser ser man blant annet i Amazon.co.uk, der man kan konkurrere om å skrive flest anmeldelser[2] . I den norske nettbokhandelen Mao.no tilbys man rabatt på neste kjøp om man er den første til å anmelde en bok. Dette viser at bokhandlene ser verdien av subjektive meninger om informasjonsobjekter. Et annet eksempel fra nettbokhandlene er muligheten man har til å bli anbefalt en bok basert på de bøkene andre kjøpere av den aktuelle boken, har kjøpt i tillegg til denne. Se Figur 7–1 nedenfor. Dette bildet er hentet fra Amazon.co.uk. Man ser at det viser fram en oversikt over andre bøker, som kanskje er aktuelle for kunden som er interessert i denne boken. Definisjonen av "aktuelle" i dette tilfellet er om andre kunder har kjøpt andre bøker i tillegg til denne, og i så fall hvilke bøker de kjøpte. Man må anta at Amazon gjør dette for å lokke kunden til å kjøpe bøker han ikke hadde tenkt til å kjøpe. Dette gjør de ved å fokusere på hva andre kunder har kjøpt og anbefalt i anmeldelser. De andre kundenes meninger om informasjonsobjekter er med på å gi et bredt beslutningsgrunnlag.

Utgangspunktet for modellen for personlige samlinger, er et ønske om å gi informasjonssøkeren et bredt beslutningsgrunnlag i informasjonssøkeprosessen. Ved å tilby funksjonalitet som personlige metadata eller annoteringer, vil brukeren få en annen beskrivelse av et informasjonsobjekt, som kan gi en bredere beskrivelse av hva objektet handler om.

Beslutningsgrunnlaget som dannes på grunnlag av personlige samlinger og personlige metadata-annoteringer, er ikke bedre enn kvaliteten på de annoteringer som informasjonssøkerne legger inn i databasen. I Mao.no kontrolleres anmeldelser av bokhandelen før den legges ut, for å sikre kvaliteten på anmel-

Figur 7–1: Anefalinger basert på andres kjøp.

delsene. Den personlige samlingstjenesten tar ikke høyde for dette, og dette er heller ikke undersøkt i forbindelse med arbeidet. Faren med å åpne for at alle kan si sin mening om et informasjonsobjekt, er at meningen ikke sjekkes for faktafeil eller lignende. Det er likevel en fordel: Brukernes synspunkter blir ikke sensurert. I en butikk vil positiv omtale lettere føre til salg, enn negativ omtale. Det er derfor ikke utenkelig at anmeldelser sensureres for å vise et positivt bilde av et informasjonsobjekt. Dette unngås i en personlig samlingstjeneste basert på arkitekturen som er presentert i denne avhandlingen.

Brukernes deltakelse i et system som dette er vesentlig. Uten brukernes deltakelse vil det ikke være noe datagrunnlag i systemet, og dette vil ikke fungere som planlagt. En indikasjon på at brukerne ikke tar seg tid til å skrive anmeldelser, er at Mao.no gir rabatt på neste kjøp til den første som anmelder en bok. Det er klart at økonomiske goder som dette kan føre til at folk skriver bokanmeldelser. I et digitalt bibliotek har man sjelden mulighet til å gi slike rabatter, og kanskje vil man ikke få folk til å skrive annoteringer på informasjonsobjekter.

7.4 Evaluering av arbeidet

Arbeidet med avhandlingen baserer seg på informasjonssøkeprosesser og personalisering. I hovedsak er arkitekturen basert på scenarier i informasjonssøkeprosessen til Marchionini, og arkitekturen søker å forenkle søkeprosessen ved å tilby informasjonssøkerne et bredere beslutningsgrunnlag i fasene der informasjonssøkerne normalt må gå til informasjonsobjektet. For å nyansere dette bildet hadde det vært ønskelig å analysere søkeprosessene i forhold til andre søkesprosess-teorier.

Søkeprosessen som er skildret i scenariene, er utført av forfatteren, men det hadde vært ønskelig å analysere søkeprosessene som virkelige brukere utfører. Dette kunne dannet et mer nyansert bilde av søkeprosessene. Kravene til arkitekturen for den personlige samlingstjenesten er basert på disse scenariene, og forfatterens erfaringer med søking og søkesystemer. Det hadde vært ønskelig å innhente disse kravene fra en større brukergruppe, og dermed fått et mer realistisk bilde av de faktiske kravene som stilles til en personlige samlingstjeneste.

Forut for implementasjonsarbeidet burde strukturerte systemutviklingsmetoder vært brukt i større grad for å unngå merarbeid om man ombestemmer seg underveis i arbeidet. Ved å utføre grundigere designarbeid ville grensesnittene mellom komponentene og dataflyten vært mer stabil, og implementasjonsarbeidet ville blitt utført raskere. Databasestrukturene burde også vært klarlagt grundigere, for å unngå programendringer som følge av databaseendringer. Rent teknisk burde programsystemene rundt SOAP vært studert grundigere, og den teknologiske plattformen kunne vært analysert grundigere. Da kunne antakelig allerede eksisterende komponenter blitt brukt som erstatning for SOAP-spesifikke deler av av systemet. Det finnes blant annet komponenter som lager SOAP-meldinger basert på WSDL-filer. Disse komponentene er ikke brukt i pilot-implementasjonen, og kunne nok spart en del arbeid i forbindelse med meldingshåndteringen.

Til slutt burde implementasjonen av arkitekturen vært fullstendig, slik at hele arkitekturen kunne blitt testet på sluttbrukere i en reell testfase. Da ville dette gitt sikrere svar på om den personlige samlingstjenesten fungerte i henhold til kravene stilt av reelle sluttbrukere. Med en fullstendig implementasjon kunne man også sagt mer om hvorvidt WebServices er egnet til å håndtere bruken en slik tjeneste ville blitt utsatt for. I en produksjonsfase måtte systemet tålt belastningen fra mange samtidige brukere, og dette vil man med dagens implementasjon ikke kunne si om systemet gjør.

Punktene som er nevnt over tar utgangspunkt i en komplett systemutviklingsprosess, og det er derfor mange ting som ikke har blitt utført grundig nok i dette arbeidet. Det er likevel ikke et mål i denne avhandligen å utføre en korrekt utviklingsprosess for å utvikle et produkt. Målene har fokusert på å vise at arkitekturen fungerer, og det rammeverket som er bygd opp, viser nettopp dette.

7.5 Videre arbeid

Denne avhandlingen har sett på flere aspekter ved innføring av en personaliseringstjeneste i et digitalt bibliotek. Da dette spenner over flere fagområder, som systemutvikling, digitale bibliotek og brukergrensesnittdesign, vil det være mange punkter å jobbe videre med.

Det bør i første omgang utarbeides en kravspesifikasjon basert på grundige brukerundersøkelser. Bare ved å finne ut hva virkelige informasjonssøkere ettespør, kan man danne krav til et system som er i samsvar med virkeligheten. Dette er et arbeid som krever mer enn det som har vært mulig i forbindelse med denne avhandlingen. Det bør også utredes spørsmål i forbindelse med personvern. Vil for eksempel brukere utlevere data om seg selv på den måten det legges opp til i personlige samlinger? Dette spørsmålet er ikke trivielt, og bør avklares før videre utvikling skjer.

Basert på krav kan man videreutvikle arkitekturen og pilot-implementasjonen av den personlige samlingstjenesten. Man kan vurdere å bytte ut deler av systemet med andre komponenter. Det er f.eks. mulig at man ønsker å benytte andre databaser i systemet, og man kan da teste andre databaseløsninger.

Det er også interessant å undersøke nye protokoller for kommunikasjon. Her kan det neves SRW som er en arvtager til Z39.50. Det kan være aktuelt å bruke denne for å hente informasjon fra den personlige samlingstjenesten, og bruke en annen protokoll for oppdatering av informasjon.

Det er også viktig at man utvikler en bedre prototyp av den personlige samlingstjenesten, og det er viktig å sette opp testkriterier for denne, slik at man kan analysere WebService-platformens egnethet og arkitekturens egnethet.

8 REFERANSER

1. AlltheWeb(2003). AlltheWeb, http://www.alltheweb.com. (Aksessert 19.11.2003)
2. Amazon.co.uk(2004), Amazon.co.uk: Top reviewers, http://www.amazon.co.uk/exec/obidos/tg/cm/top-reviewers-list/-/1//ref=cs_hd_lp_1/202-5614247-5841426
3. Arms, William Y. & Christophe Blanchi & Edward A. Overly(1997). An Architecture for Information in Digital Libraries. http://www.dlib.org/dlib/february97/cnri/02arms1.html.
4. Bainbridge, David & Buchanan, George & McPherson, John & Jones, Steve & Mahoui, Abedelaziz & Witten, Ian H. (2001) Greenstone: A Platfor for Distibuted Digital Library Applications. I: Lecture Notes in Computer Science no 2163, ss. 137-148.
5. BIBSYS (2004). Antall elektroniske tidsskrifter i BIBSYS Bibliotekbase. http://wgate.bibsys.no/gate1/FIND?bibl=BIBSYS&F0=&felt0=to&de=&fm=n&lang=N&base=TIDSSKR&sort=ts&type=S%F8k (Aksessert 06/01/2004)
6. Carroll, Sean (2003). How to find anything online, PC Magazine, Vol. 22, Number 9.
7. Chun Wei Choo & Detlor, Brian & Turnbull, Don (1998). A Behavioral Model of Information Seeking on the Web - Preliminary Results of a Study of How Managers and IT Specialists Use the Web. http://

choo.fis.utoronto.ca/FIS/ResPub/asis98/default.html. (Lesedato 22.03.04)

8. Chun Wei Choo & Detlor, Brian & Turnbull, Don (2000). Information Seeking on the Web, An Integrated Model of Browsing and Searching. http:/firstmoday.org/issues/issue5_2/choo/(Lesedato 28.04.2002)
9. Cohen, Suzanne & Fereira, John & Horne, Angela & Kibbee, & Mistlebauer, Holly & Smith, Adam(2000). MyLibrary. Personalized Electronic Services in the Cornell University Library. I: D-Lib Magazine, Volume 6 Number 4. http://www.dlib.org/dlib/april00/mistlebauer/04mistlebauer.html.
10. Dienst, Dienst Overview and Introduction, http://www.cs.cornell.edu/cdlrg/dienst/DienstOverview.htm
11. DOI, The DOI handbook, http://doi.org/hb.html
12. Dublin Core Metadata Element Set, http://dublincore.org/documents/dces/.
13. Ellis, David (1993). Modelling th Information-Seeking Patterns of Academic Researchers: A Grounded Theory Approach. I: The Library quarterly. 63, no. 4, s. 469-486. ISSN: 0024-2519
14. Ellis, David & Haugan Merethe (1997). Modelling the information seeking patterns of engineers and research scientists in an industrial environment . I: The Journal of documentation. 53, no. 4, s. 384-403. ISSN: 0022-0418.
15. European Conference on Research and Advanced Technologies for Digital Libraries(2003), http://www.ecdl2003.org.
16. Feide (2004), http://www.feide.no
17. French, James C. & Viles, Charles L. (1999). Personalized Information Envorionments. I: D-Lib Magazine, vol 5, no 6, URL:www.dlib.org/dlib/june99/french/06french.html.
18. French, James C. & Viles, Charles L. (1998). Personalized Information Envorionments. DARPA PI Meeting Presentation. URL:http://www.cs.virginia.edu/~cyberia/PIE/98pi.pdf
19. Ghaphery, James (2002). My Library at Virginia Commonwealth University. Third Year Evaluation. I: D-Lib Magazine, Volume 8 Number

7/8. http://www.dlib.org/dlib/july02/ghaphery/07ghaphery.html.

20. Google(2003). Google, http:// www.google.com

21. Graphery, james & Ream, Dan (2000). VCU's My Library: Librarians Love It. . . . Users? Well, Maybe. I: Information Technology and Libraries, Volume 19, Number 4. http://www.lita.org/ital/ ital1904.html#anchor158481.

22. Handle System, The(2003), http://www.ietf.org/internet-drafts/draft-sun-handle-system-protocol-05.txt. Se også http://ww.handle.net.

23. Haugen , Frank B. og van Nuys, Carol(2003). Dublin Core metadataelementer, versjon 1.1: Referansebeskrivelse, http://www.nb.no/ katkom/dublincore.html

24. Hill, LL & Janee, G & Dolin, R & Frew, J & Larsgaard, M (Nov 1999). Collection Metadata solutions for digital library applications. I: JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE, Vol. 50, Nr. 13, s. 1169-1181.

25. IEFT(1995), RFC 1807, A Format for Bibliographic Records, http:// www.ietf.org/rfc/rfc1807.txt?number=1807, Aksessert 2003-11-11.

26. Jones, D (1999) . Collection development in the digital library. I: SCIENCE & TECHNOLOGY LIBRARIES, Vol. 17, Nr. 3-4, s. 27-37.

27. Ketchell, Debra S. (2000). Too Many Channels: Making Sense out of Portals and Personalization. I: Information Technology and Libraries, Volume 19, Number 4. URL:http://www.lita.org/ital/1904_ketchell.html.

28. Kaarø, Jan Erik (2004). Farvel til trykte tidsskrifter. I: Universitetsavisa. http://www.universitetsavisa.no/ ua_lesmer.php?kategori=nyheter&dokid=3ffdaced7d8ed1.74055415.

29. Lee, HL (Okt 2000). What is a collection?. I: JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE, Vol. 51, Nr. 12, s. 1106-1113.

30. Library and Information Technology Association(1999). Technology and library users: LITA experts identify trends to watch. http://www.ala.org/ Content/NavigationMenu/LITA/LITA_Resources_and_Services/

Top_Technology_Trends/Midwinter_1999.htm.

31. Libarry of Congress (2004). SRW-Search/Retrieve WebService. http://www.loc.gov/z3950/agency/zing/srw/background.html
32. Logoze, Carl & Fielding, David (Nov 1989). Defining Collections in Distributed Digital Libraries. I: D-Lib Magazine. URL:http://www.dlib.org/dlib/november98/lagoze/11lagoze.html.
33. Mao.no(2004),Bokanmeldelser i mao.no, http://mao.no/nb_redaksjonen/nb_hjelp_tekst.en.htm#bokanmeldelser
34. Marc 21 Format for Bibliographic Data(2002), http://www.loc.gov/marc/bibliographic/
35. Marchionini, Gary (1998). Towards the digital library : the British Library’s Initiatives f;:1998 [Marchionini95]
36. Marchionini, Gary(1995), Information Seeking in Electronic Environments. Cambridge: Cambridge University Press.
37. Morgan, Eric Lease (1999). MyLibrary@NCState: The Implementation of a User-centered, Customizable Interface to a Library's Collection of Information Resources. http://www.ted.cmis.csiro.au/sigir99/morgan/
38. Microsoft (2004), .Net Home, http://www.microsoft.com/net/.
39. Ober, John (1999). The California Digital Library. I:D-Lib Magazine, Vol. 5 No. 3. http://www.dlib.org/dlib/march99/03ober.html.
40. Ryvarden, Einar (2003). Google tror på 104 milliarder. digi.no. http://digi.no/php/art.php?id=94995
41. Stanford Digital Library Technologies, http://www-diglib.stanford.edu/diglib/pub/index.shtml
42. Statistisk Sentralbyrå(2003), Internett-målinga, 4. kvartal 2002, http://www.ssb.no/emner/10/03/inet/
43. Statistisk Sentralbyrå(2003), Norsk mediebarometer, 2002, http://www.ssb.no/vis/07/02/30/medie/sa57/art-2003-03-31-01.html.
44. Sun (2004), Java technology, http://java.sun.com.
45. The Perl Fondation (2004), The Perl Directory - perl.org, http://

www.perl.org.

46. Tomcat (2004), Apache Software Foundation, http://jakarta.apache.org/tomcat/index.html.
47. Uddi.org (2004), About UDDI, http://www.uddi.org/about.html.
48. Universitetsbiblioteket i Trondheim(2003), http://www.ub.ntnu.no, Aksessert 2003-11-11.
49. W3c (2000), Extensible Markup Language (XML) 1.0 (Second Edition), http://www.w3.org/TR/REC-xml, Aksessert 2003-11-11.
50. W3C, HTTP - Hypertext Transfer Protocol Overview, http://www.w3.org/Protocols/, Aksessert 2003-11-11.
51. W3C (2000), Simple Object Access Protocol (SOAP) 1.1, http://www.w3.org/TR/SOAP/, Aksessert 2003-11-11
52. W3C, Web Services Activity, http://www.w3.org/2002/ws/, Aksessert 2003-11-11.
53. W3C, Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language,http://www.w3.org/TR/wsdl20/, Aksessert 2004-01-29.
54. W3C, World Wide Web Consortium, www.w3c.org, Aksessert 2003-11-11.
55. Witten, I.H. & Bainbridge, B. & and Boddie, S.J. (2001). Power to the People: End-User Building of Digital Library Collections. I:First ACM/IEE-CS Joint Conference on Digital Libraries, Roanoke, Virginia, 24-28 June, pp 94-103. http://www.cs.waikato.ac.nz/~nzdl/publications/2001/01IHW-DB-SB-Powertothepeop.pdf.
56. Witten,Ian H. & Bainbridge, David & Boddie, Stefan J. (2001). Greenstone: Open-Source Digital Library Software. I: D-Lib Magazine, Vol. 7, No. 10. http://www.dlib.org/dlib/october01/witten/10witten.html.
57. Witten, Ian H. & McNab, Rodger J. & Boddie, Stefan J. & Bainbridge, David (2000). Greenstone: a Comprehensive Open-Source Digital Library Software System. http://www.cs.waikato.ac.nz/~nzdl/publications/2000/00IHW-RJM-SJB-DB-Open-Source.pdf.
58. Witten, I.H. & Bainbridge, D. & and Boddie, S.J. (2001). Online

Information Review, vol 25, no 5, pp 288-198. http://www.cs.waikato.ac.nz/~nzdl/publications/2001/01IHW-DB-SB-Greenstoneopn.pdf.

59. Whatis.com(2003), Digital Library, http://whatis.techtarget.com/definition/0,,sid9_gci750204,00.html

60. Z39.50 Maintenance Agency(1995), InformationRetrieval(Z39.50): Application Service Definition andProtocol Specification, ftp://ftp.loc.gov/pub/z3950/official/part1.pdf, ftp://ftp.loc.gov/pub/z3950/official/part2.pdf, ftp://ftp.loc.gov/pub/z3950/official/part3.pdf, ftp://ftp.loc.gov/pub/z3950/official/part4.pdf

61. Aalberg, Trond & Hegna, Knut (2000). Arkitektur for digitale bibliotek. Trondheim.

APPENDIX A MARCHIONINIS SØKEPROSESS

A.1 Introduksjon

Gary Marchioninis søkeprosess er en ofte referert til teori. Marchionini beskriver søkeprosessen som en kognitiv prosess der han fokuserer på rasjonelle valg og begrunnelser som brukeren gjør når han befinner seg i søkeprosessen. Modellen tar likevel høyde for en situasjon der man også foretar ikke-rasjonelle valg.

Bakgrunn for Marchioninis søkeprosess er at brukeren har et informasjonsbehov. Dette setter i gang selve prosessen. Et informasjonsbehov er forskjellen mellom det brukeren vil vite og det brukeren allerede vet.

Marchionini deler modellen inn i de faktorer som er av betydning for informasjonssøkeprosessen og de delprosesser som utgjør selve søkeprosessen. Faktorene påvirker hvordan en informasjonssøker utfører prosessen og de resultater han oppnår. Først presenteres en forklaring av disse faktorene, og deretter en forklaring av søkeprosessen og de delprosessene den er satt sammen av.

A.1.1 Faktorer

Søkeprosessen som Marchionini beskriver, består av flere faktorer som spiller sammen.

Informasjonssøker
Informasjonssøkeren er den personen som har et informasjonsbehov. Denne faktoren interagerer med søkesystemet og kontrollerer oppgaven som skal utføres. Det er også denne informasjonssøkeren som avgjør hva som er et godt resultat.

Informasjonssøkeren har unike mentale modeller av verden som samspiller med de erfaringer, evner og preferanser han har. Alt dette påvirker personen i hele søkeprosessn.

Det finnes flere typer informasjonssøkere. Marchionini deler dem inn i to kategorier; megleren og domeneeksperten. Megleren fokuserer på spørringer og informasjonsstrukturer, mens domeneeksperten har et klarere bilde av svaret han ønsker å finne, og søker å indentifisere akkurat dette.

Andre viktige egenskaper ved informasjonssøkeren:

- Alder, spatial resonnering (Egan 1988 i [36]).
- Akademisk suksess, leseevne, studium, verbale kvantitative evner (Borgman, Marchionini 1988).

- Fysisk og psykisk helse påvirker personer og kan ikke ignoreres [36] .

Hvilket informasjonsbehov man har, er en meget viktig faktor som ikke er uavhengig av informasjonssøkeren. Marchionini gjengir fire nivåer etter Taylor, 1962. Det første er *emosjonelt nivå* (visceral level). Her er personen klar over en mangel på informasjon, men har ikke definert den kognitivt. På det *bevisste nivå* (conscious level) klarer personen å uttrykke mangelen, og har satt grenser for den, om enn ikke presist. Når informasjonsbehovet er på et *formalisert nivå,* kan det uttrykkes i klartekst. Til slutt kommer det kompromitterte nivået, og her kan problemet uttrykkes i et spørrespråk. Nivåene "emosjonelt" og "bevisst" korresponderer til Dervins "gap" og Belkins "anomalous state of knowledge" (Marchionini [36] , s 35). Marchionini beskriver et informasjonsbehov med begrepene "noumena" og "cloud". Noumena er minnespor og inntrykk som samlet i en cloud, danner et konsept eller en ide. Et informasjonsproblem får vi når det er for få felles noumen i en mengde skyer. Stabil kunnskap får vi når det er nok felles noumen i en mengde skyer. En mangel i antall noumena eller clouds er ofte et resultat av ytre påvirkning.

Oppgaven

Oppgaven er en realisering av informasjonssøkerens problem. Det er oppgaven som driver søkeprosessen framover, og den består i å uttrykke problemet og interagere med søkesystemet til problemet er løst. Uttrykket og artikulasjonen av problemet er vanligvis i form av et spørsmål, og interaksjonen med søkesystemet er både mental og fysisk. I tillegg må søkeren reflektere over resultatene. I utgangspunktet starter det med at brukeren har et mål for øyet, men etter hvert som søkeprosessen skrider fram, kan planene endres. Mer om dette kommer når selve prosessen skal beskrives.

Det er to måter å karakterisere oppgaven på; i forhold til kompleksiteten, eller i forhold til det forventede svaret.

Etter hvert som informasjonssøkere definerer problemet, kommer de fram til konsepter og relasjoner, og legger til termer til konseptene for å uttrykke det gitte problemet. Dette danner grunnlaget for kompleksiteten i oppgaven.

Oppgaver kan også uttrykkes med målet eller svarene man forventer. Her opererer man med 3 skalaer. Se tabell A–1 nedenfor. Disse er spesifisitet, kvantitet og beleilighet. Spesifisitet måler hvor "stort" informasjonsgapet er. Dette kan

Tabell A–1: Beskrivelse av en oppgave

Spesifisitet	Målene kan gå fra en enkelt fakta-opplysning til en forståelse eller en mening.
Kvantitet	Mengde målt i informasjonsmengde eller tidsforbruk.
Beleilighet	Forventet tid for fullførelse: umiddelbart (fakta-opplysninger som vi slår opp), eller sent (læring tar lengre tid). Tidsforbruket må samsvare med forventningene.

gå fra en enkelt fakta-opplysning til en mening. Kvantiteten er et mål på f.eks. hvor mange informasjonsenheter som trengs eller hvor lang tid man kan bruke

på oppgaven. Beleilighet måler i hvor stor grad tidsbruken samsvarer med det forventede tidsforbruket.

Søkesystem
Søkesystemet er kilden som representerer kunnskap, og som tilbyr verktøy og regler for tilgang og bruk av den kunnskapen [36] . Kunnskap som potensielt er tilgjengelig, kalles i denne sammenheng en database, det være seg en bok, et menneske eller et bibliotek. For å få tilgang til databasen må man gå via et grensesnitt. Grensesnittet er en representasjon av kunnskapen og verktøy samt regler og mekanismer for tilgang og manipulering. Samlet strukturerer disse komponentene kunnskapen.

Det finnes konseptuelle og fysiske komponenter av databasen og grensesnittet. For databasen er det innhold (content) som er den viktigste konseptuelle delen. Innhold peker her på hva slags kunnskap som finnes i databasen; om databasen inneholder primærdata (fulltekst), sekundærdata (bibliografi) eller tertiærdata (metabibliografisk). Et tredje aspekt er typen innhold (tekst, grafer, bilder etc). Størrelsen og kvaliteten på databasen er også viktig (en bok eller et bibliotek, og er det snakk om en seriøs forfatter eller et lite folkebibliotek?). Til slutt teller granulariteten til dataene i databasen inn (generell eller spesiell informasjon).

Den fysiske organiseringen av databasen gjør også sitt til at det påvirker søkingen. Den avgjør hvordan man skal gå fram for å finne det man er på jakt etter, og om man trenger spesielle verktøy. Grensesnittet er en megler mellom databasen og brukeren som søker. Hvordan grensesnittet er organisert, avgjør hvor lett det er å lære og bruke grensesnittet. Den fysiske delen av grensesnittet omhandler hvordan man forteller systemet noe (input), og hvordan systemet gir tilbakemeldinger (output). Input-innretninger kan f.eks. være blyant, tastatur eller stemme. Output-innretninger kan f.eks. være bøker, skjerm eller stemme. Interaksjonen med grensesnittet bestemmes i stor grad av databasens fysiske aspekter.

Den konseptuelle delen av grensesnittet omhandler regler og protokoller for informasjonsflyt. Et grensesnitts interaksjonsstil omhandler hvordan man kommuniserer med databasen, og hva som kreves av brukeren for å gjøre det. Grensesnittets representasjonsstil sier noe om organisering av informasjon og fysiske mekanismer for å manipulere strukturen. Marchionini nevner tre typer struktur: lineær, hierarkisk og nettverksstruktur. Dette åpner for at de forskjellige behovene kan initiere forskjellige representasjoner og dermed støtte informasjonssøkeren i større grad. Søkemekanismer definerer og begrenser søkene. Bøker har f.eks. indekser, og elektroniske systemer har i tillegg bl.a. boolsk søking.

Domenet
Domenet representerer kunnskapsområdet eller fagfeltet (f.eks. informatikk eller psykologi). Domenet består av entiteter og relasjoner mellom disse. Det varierer i kompleksitet og påvirker informasjonsøkingsprosessen og delprosessene. Domenet avgjør i stor grad hvilken database man må velge for å få dekket informasjonsbehovet. Noen vitenskaper har informasjon som "går ut

på dato" mens andre domener har mer persistent informasjon. Dette må man forholde seg til som informasjonssøker, og det påvirker de valg man må ta i søkeprosessen.

Rammer
Rammefaktorer angir rammene som begrenser en søkeprosess på ulike måter. Marchionini deler dem inn i to grupper: fysiske og konseptuelle. De fysiske rammene legger begrensninger i form av tiden man kan bruke på et søk, fysisk tilgjengelighet av materiale, komfort, distraksjoner og kostnader. Dette skiller søk som foregår på søkerens kontor fra søk som foregår på det offentlige biblioteket, eller på en maskin du må betale for å bruke samtidig som det er lang kø av andre som venter på tur. De konseptuelle rammene handler om psykologiske og sosiale faktorer som påvirker søkeren i både positiv og negativ retning. Psykologiske faktorer kan være kognitive evner etc. Sosiale faktorer kan være sosial status og sosial interaksjon.

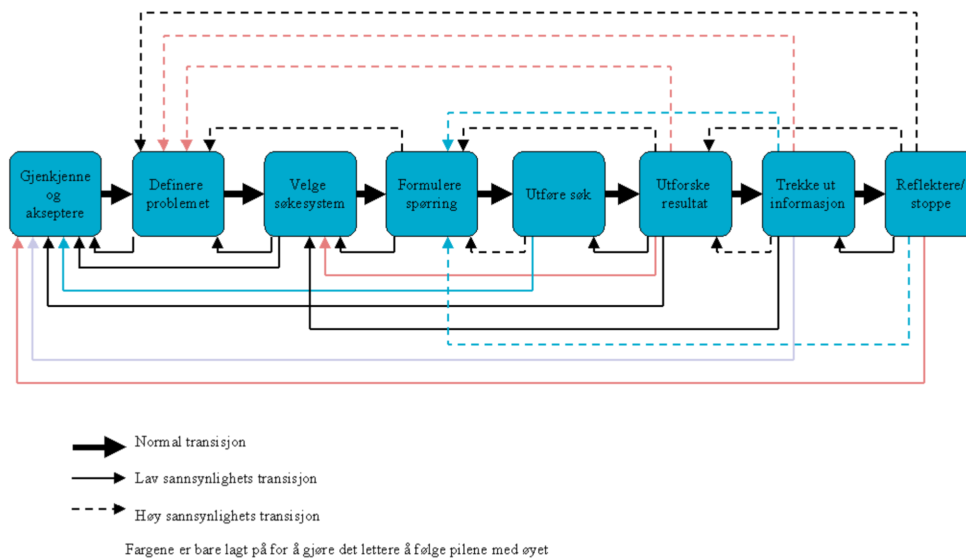
Søkeresultat
Søkeresultater er ikke bare selve resultatet av søk, men også prosessen som leder til resultatene. Søkeresultatene er selvsagt viktige, men selve prosessen leder også med seg resultater, og man lærer ting man tar med seg videre i prosessen. Man kan evaluere systemet man søker i, og seg selv, og dette kan man bruke for å forbedre søkeprosessen.

A.1.2 Søkeprosessen

Innrømme og akseptere et informasjonsproblem
Første skritt i søkeprosessen er å innrømme og akseptere at man har et informasjonsproblem. Dette kan være internt eller eksternt motivert. Informasjonsproblem manifesteres som et ressursbehov i kognitive systemer. Dette er nevnt tidligere i kapittel A.1.1. Problemet kan aksepteres eller undertrykkes. En undertrykking er påvirket av rammene: kostnader kan ikke forsvares etc. En aksept er påvirket av kunnskap om domenet, kunnskap om søkesystemer, rammene og andre personlig årsaker.

Definere og forstå problemet
Etter at søkeren har innsett en mangel på informasjon, må han definere og forstå problemet. Denne prosessen er aktiv så lenge søkeprosessen pågår. Det er også en stor mulighet for at søkeren kommer tilbake til denne prosessen etter hvert som han går gjennom søkeprosessen. Den eneste delprosessen som ikke har en mulighet for å returneres tilbake til prosessen, er "formulere spørring". At denne delprosessen er viktig i søkeprosessen er det derfor liten tvil om, men denne delprosessen blir ofte tatt for lett på av sluttbrukere, og dette fører til frustrasjon og feil [36] .

Definisjonen av problemet må begrense og klassifisere det. I tillegg må man bestemme seg for en form på svaret; om man vil ha svaret som fakta eller økt forståelse. Dette setter begrensninger på resultatene, som igjen vil påvirke søket. Måten å angripe et problem på vil variere om det man skal ha tak i er et foto eller en idé.



Figur A–1: Informasjonsøkeprosessen etter Marchionini

Velge søkesystem

Når man velger seg et søkesystem, gjør man dette på bakgrunn av den erfaringen man tidligere har i domenet og ut fra personlig infrastruktur. Domenekunnskap setter sterke føringer for hvilket søkesystem man velger. Personlig infrastruktur avhenger av tidligere erfaring med problemløsning generelt samt søkers kognitive ferdigheter. En tredje faktor som påvirker valg av søkesystem er tidligere erfaring med søkesystemer.

Informasjonssøkere tilpasser søkeoppgaven til ett eller flere søkesystemer. Vanligvis brukes flere, og med framveksten av flere elektroniske søkesystemer er det mer og mer vanlig å bruke søkesystemer for å finne et egnet søkesystem.

Formulere søk

Prosessen med å formulere et søk handler om å tilpasse søkerens forståelse av problemet til det søkesystemet man valgte i forrige fase. Dette innebærer at man må overføre søkerens vokabular for å uttrykke problemet til søkesystemets vokabular (*sematic mapping*). Deretter må man tilpasse søkestrategier og søke-taktikk til reglene for søkesystemet (*action mapping*).

Utføre søk

Denne prosessen drives av søkerens mentale modell av søkesystemet, og er basert på formuleringen av spørringen som ble utviklet i forrige fase. Selve prosessen kan dreie seg om alt fra å skrive noen ord på tastatur til å si noe til en annen person.

Undersøke resultat

For at søkeren skal kunne måle framgang mot avslutning av søkeprosessen, må søkeresultatene undersøkes. Dette baseres på kvantitet, type og format på re-

sultatet, og søkeren må bedømme relevansen på treffene. Problemet og søkerens personlige informasjonsinfrastruktur bestemmer dennes forventninger til resultatet og antallet treff, og kvaliteten på disse, som er nødvendig for å avslutte søkeprosessen. Representasjonen av søkeresultatene er viktig for hvordan søkeren bearbeider dem. Jo større resultatsettet er, jo viktigere er organisering. Elektroniske systemers tendens til å produsere store resultatsett, gjør denne oppgaven vanskeligere.

Resultatenes relevans må bedømmes i forhold til den definerte oppgaven. Fra søkers praktiske perspektiv kan relevans sees på som avgjørelsen om hva som er neste handling i søkeprosessen. Søkeren har flere muligheter:

- Avslutte søket fordi målet er nådd.
- Undersøke dokumentet grundigere, nå eller senere (ta vare på informasjon om dette).
- Forfølge dokumentets implikasjoner for den videre søkingen.
- Fortsette å undersøke flere resultat i settet.
- Formulere en ny spørring.
- Redefinere problemet.
- Forkaste dokumentet og fortsette å undersøke resultatsettet.
- Forkaste dokumentet og stoppe søkeprosessen uten å fullføre den.

Valget påvirkes av typen resultat og av mengden. Ved små sett kan søkeren kikke fort gjennom dem eller undersøke dem systematisk og/eller grundig. For store resultatsett må man kanskje redusere settet med en ny spørring. Man kan også bruke semantiske surrogat (metadata) for å finne en mer utfyllende relevansvurdering. Undersøkelser viser at brukere er villige til å skanne betydelige resultatsett hvis de har riktige verktøy[35] .

Trekke ut informasjon

Å trekke ut informasjon fra resultatene henger tett sammen med å undersøke settene. Hvis søkeren finner et relevant dokument, kan han fortsette bearbeiding av det og trekke ut mer informasjon, eller vente med videre bearbeiding og forsette undersøkelsen av resultatsettet.

For å trekke ut informasjon må søkeren bruke kunnskaper om lesing, skanning, høring, klassifikasjon, kopiering og lagring av informasjon. Når informasjon er trukket ut, vil denne informasjonen legges til i søkerens kunnskap om domenet og dermed endre grunnlaget for søkeprosessen. (Se «Informasjonssøker» på side 113.s)

Reflektere eller stoppe

For å oppnå ønsket resultat, trenger man ofte flere iterasjoner i denne prosessen. Det første resultatsettet fungerer ofte som en form for input til formulering av videre spørringer og søk. Hvis man skal gå en runde til i “sirkelen”, kreves det en vurdering av selve søkeprosessen og om denne svarte til forventningene.

Spørsmålet om hvorvidt man skal stoppe eller ikke, kan avhenge av flere ting som søkesystemet, rammene, oppgaven, domenekunnskapen eller informasjonssøkingsevne.

APPENDIX B SØKEPROSESSEN

B.1 Søkeprosessen beskrevet i Marchioninis termer

Innrømme og akseptere problemet
Før prosessen starter må man innrømme at man har et informasjonsbehov. I dette spesielle tilfellet er denne prosessen både internt og eksternt motivert. Den interne faktoren dreier seg i hovedsak om at denne avhandlingen skal skrives, og at det trengs mer kunnskap om domenet avhandlingen befinner seg i. Den eksterne påvirkningen kommer fra veilederen som ba forfatteren finne 15 referanser som er relevante for problemstillingen i oppgaven.

Problemstillingen handler om "personlige samlinger i distribuerte digitale bibliotek". Dette inkluderer emner som "digitale bibliotek", "samlinger i digitale bibliotek", og "personlige digitale bibliotek".

Definere problemet
Problemet er delvis definert ovenfor: å finne 15 referanser for å øke kunnskapen om emnet "personlige samlinger i distribuerte digitale bibliotek". De 15 referansene er bare en måte å få mulighet til å øke kunnskapen på. Med andre ord er det primære målet å øke kunnskapen om domenet og sekundært finne 15 referanser.

Formen på resultatet av søkeprosessen er vanskelig å definere. For det sekundære problemet er det forholdsvis greit da man lett kan telle antall dokumenter som ble funnet. Det blir verre når relevansen til disse skal vurderes, og det blir ikke lettere å måle den økte kunnskapen, som er det primære målet. Det er derfor definert noen klare evalueringskriterier for søket. På grunn av liten kompetanse innen området er det vanskelig å si om et funn er relevant, og i så tilfelle om funnet er verdt å ta vare på som en av de 15 referansene, som det sekundære målet krever.

Velge søkesystem
BibSøk-Nett fra BIBSYS ble raskt valgt som første søkesystem. Det er flere grunner til dette. Kjennskap til systemet og det faktum at det er "kvalitetssikret informasjon" tilgjengelig er viktige faktorer. Det er også relativt lett å få tak i eventuelle dokumenter. Andre mulige søkesystem er personer som har arbeidet i dette kunnskapsdomenet tidligere, og andre søkesystem som ACM og D-Lib.

Definere problem
I tillegg til å finne de tidligere nevnte artikler, ønsket vi å få generell kunnskap om domenet og det valgte søkesystemet.

Formulere spørring
Det ble valgte å søke i fritekst da dette gir bredest treff-flate for søket. Spørringen "digital library" ble ansett som bred nok til å omfatte det meste innen domenet, og ga større oversikt.

Utføre søk
Søket ble sendt til systemet ved å skrive inn termene i søkesystemets dialog og trykke "send".

Utforske resultat
Resultatsettet inneholdt ikke så mange treff som forventet, og de var ikke organisert på noen måte, men det inneholdt flere relevante treff. Treffene gir en liten oversikt over domenet, men det var for få treff til at man kunne gjøre seg opp en mening om dette var de 15 riktige dokumentene. Noen av de mest relevante treffene ble tatt vare på for senere undersøkelse, og noen ble en del av de 15 etterspurte dokumentene.

Trekke ut informasjon
De interessante treffene ble grundig sjekket, og de ga noe informasjon som la seg som en del av domenekunnskapen.

Reflektere/stoppe
Ettersom 15 relevante treff ikke er funnet ennå, må søket fortsette. Hvorvidt dette eventuelt er relevante treff er også et usikkerhetsmoment.

Formulere spørring
Spørringen ble omformulert til "electronic library" da dette og "digital library" er begreper som kan brukes om hverandre i mer eller mindre overlappende grad, som man kan se av resultatene i Kapittel 3.8.

Utføre søk
Søket ble sendt på samme måte som tidligere.

Utforske resultat
Antallet treff på siste søk var høyere enn på termen "digital library". Flere av treffene anses som relevante. Målet med 15 treff er ennå ikke nådd, og litteratur nok til å mette kunnskapsbehovet er ennå ikke funnet. Noen av treffene ble tatt vare på for framtidig sammenligning med andre ressurser.

Trekke ut informasjon
De interessante treffene ble grundigere sjekket, og de ga noe informasjon som la seg som en del av domenekunnskapen.

Reflektere/stoppe
15 relevante treff var ennå ikke funnet, så søket fortsatte. Det var dessuten et stort usikkerhetsmoment om dette var relevante treff. De to bredeste og mest brukte termer som er kjent for domenet, var brukt uten at de hadde gitt nok resulater, så et annet søkesystem måtte velges for å få flere relevante treff.

Velge søkesystem
BIBSYS ISI-Søk ble nå valgt. Ut fra tidligere kjennskap til databasen var det større tiltro til at denne basen kunne gi mer interessante treff. Dette er dessuten en artikkelbase, så det var også derfor større muligheter for mer relevante treff i denne, da artikler er den foretrukne publiseringmåten i dette fagfeltet.

Definere problem
Problemet var her det samme som tidligere, men i tillegg er det et nytt søkesystem å gjøre seg kjent med.

Formulere spørring
Som tidligere ble det søkt bredest mulig her også, for å sammenligne med BibSøk Nett-søket som var gjort tidligere. Spørringen er ellers som i første søk.

Utføre søk
Søkesystemet er veldig likt BibSøk-Nett, så framgangsmåten er den samme.

Utforske resultat
Antallet treff i ISI-Søk var flere enn det er hensiktsmessig å bla gjennom. Resultatene var sortert på år, men ikke organisert på annen måte. Disse treffene ga god oversikt over området, men det var for mange treff til at de kunne brukes til noe annet enn å konstatere at dette søkesystemet hadde mange flere artikler om samme tema enn BibSøk Nett. Det var derfor behov for å redusere resultatsettet.

Formulere spørring
Resultatsettet ble avgrenset til artikler publisert de siste fem år.

Utføre søk
Avgrensingen ble utført med å velge avgrensingsfunksjonen og skrive inn et årstall å avgrense fra.

Utforske resultat
Resultatsettet var fortsatt for stort til at det med letthet kunne gjennomgås. Derfor ble søket ytterligere avgrenset.

Formulere spørring
Søket ble nå formulert enda smalere for å få en overkommelig treffmengde. Ordet “collection” ble lagt til spørringen i håp om at det ga et noe mindre resultatsett. Dette søket passet også antakelig bedre inn i problemstillingen som var definert til å begynne med.

Utføre søk
Søket ble sendt som tidligere.

Utforske resultat
Resultatet ga en overkommelig treffmengde, og bearbeiding av resultatene ga mer kunnskap om domenet og flere relevante treff. På bakgrunn av dette ble det dannet evalueringskriterier: Artiklene må gi en oversikt over samlinger i di-

gitale bibliotek, og må ha informasjon om personlige samlinger og distribuerte digitale bibliotek generelt. I tillegg må det være 15 referanser.

Trekke ut informasjon
I resultatsettet ble det funnet mange artikler som kunne være av interesse for den definerte problemstillingen.

Reflektere/Stoppe
Det ble funnet godt over 15 relevante referanser. De som ble funnet ble også vurdert som relevante nok i forhold til problemet. Informasjonssøkeprosessen kunne derfor avsluttes.

APPENDIX C SKJERMBILDER

Nedenfor er det vist noen skjermbilder fra et fiktivt system for personlige samlinger. Skjermbildene baserer seg på at personlige samlinger er integrert i BibSøk Nett fra BIBSYS.

Normal visning av en post i databasen

BIBSYS Post 031887961 - BibSøk Nett -

Enkel søk | Avansert søk | Tidsskrifter | Veiledning

Tittel: **The ultimate digital library : where the new information players meet / Andrew K. Pace.**
Forfatter: Pace, Andrew K.
Trykt: Chicago : American Library Association, 2003.
Sidetall: XVII, 168 s. : ill.
ISBN: 0-8389-0844-6 (h.)
Eiere: HIS NBO UBB UBIT

Tilleggsinformasjon:

- Fullstendig eksemplarliste .
- Fullstendig liste over emneord og klassifikasjonsnummer .

Bestill lån eller artikkelkopi.
Eksporter til format MARC

Andre brukere mener:

Jeg synes boka var interessant, og vil på det sterkeste anbefale den for andre som er interessete i digitale bibliotek. Boka beskriver der ultimate digitale biblioteket der informasjonssøkere møtes. Boka er sikkert ikke egnet som pensum for uinteresserte, men for profesjonelle, er den et sikkerstikk.
Professor Olsen.

Jeg synes boka er noe av det kjedeligste jeg har lest, og synes bare at den er i veien i skolesekken min. Dessuten er det ikke et eneste fargebilde, og det er en plage å lese den. Stakkars studenter som må gjennom denne boka.
Grunnfagsstudent Nilsen

Figur 3–1: Skjermbilde fra postvisning i et søkesystem med personlige samlinger.

Skjermbildet (Figur 3–1) viser en post i databasen når det er lagt inn kommentarer fra andre brukere. På denne posten ser man at det er lagt inn to kommentarer fra brukere, hvorav en er fra en professor og en er fra en student[11]. En informasjonssøker vil her se hva andre mener om dette informasjonsobjektet, og kan basere sine egne meninger på dette.

Visning av personopplysninger

Ved at en bruker klikker på navnene under kommentarene kan de få se informasjon om de brukerne som har lagt inn kommentarer på posten. Om brukeren klikker på lenken det står “Professor Olsen” kan brukeren få se informasjon om Ole Olsen. Dette er vist i Figur 3–2.

11. Merk at kommentarene på posten er fiktive og ikke er reflekterer innholdet i boka som presenteres.

BIBSYS - BibSøk Nett -

Enkel søk | Avansert søk | Tidsskrifter | Veiledning

Personopplysninger

Navn:	Ole Olsen
Stilling	Professor
Grad:	Dr.
Institusjon:	Universitet
Arbeids-, studie-sted	UniversitetET

Se personlig samling

Figur 3–2: Skjermbilde fra visning av personopplysninger i et søkesystem med personlige samlinger integrert.

Visning av en personlig samling

BIBSYS - BibSøk Nett -

Enkel søk | Avansert søk | Tidsskrifter | Veiledning

Personlig samling

Navn:	Ole Olsen
Stilling	Professor
Grad:	Dr.
Institusjon:	Universitet
Arbeids-, studie-sted	UniversitetET

Dokumenter

1	**Tittel**	The ultimate digital library : where the new information players meet
	Identifikator	031887961
	Personlig Kommentar	Jeg synes boka var interessant, og vil på det sterkeste anbefale den for andre som er interessete i digitale bibliotek. Boka beskriver der ultimate digitale biblioteket der informasjonssøkere møtes. Boka er sikkert ikke egnet som pensum for uinteresserte, men for profesjonelle, er den et sikkerstikk.
2	**Tittel**	Information seeking in electronic environments
	Identifikator	95234677x
	Personlig Kommentar	Boka beskriver på en meget god måte hvordan vi leter etter informasjon i elektroniske systemer, og det beskrives bl.a. en modell for hvordan dette skjer. Denne modellen er meget interessant. Boka enger seg for alle som har ønske om å sette seg inn i informasjonssøking.

Figur 3–3: Visning av en personlig samling for en bruker i et søkesystem med personlige samlinger integrert.

Om en bruker ønsker å se den personlige samlingen til en bruker kan skjermbildet se ut som i Figur 3–3. Her vises personopplysninger samt de informasjonsobjektene denne brukeren har kommentert og valgt å ha som en del av sin personlige samling. Denne samlingen kan brukes av "eieren" for å organisere

informasjon han trenger som del av et forskningsprosjekt, eller den kan brukes av andre informasjonssøkere som et sted for å finne informasjon som kan være relevant (antagelsen er at informasjonssøkeren har samme interessefelt som eieren av den personlige samlingen, og derfor vil finne relevent informasjon i den personlige samlingen).

APPENDIX D TEKNOLOGIFORKLARINGER

D.1 SOAP

SOAP (Simple Object Access Protocol) er en måte for et program som kjører under et operativsystem å kommunisere med et annet program, som kjører under samme eller et annet operativsystem, ved å bruker Hypertext Transfer Protocol (HTTP) og Extensible Markup Language (XML) som informasjonsutvekslings-mekanisme. Fordi HTTP og XML er tilgjengelig i alle store operativsystemer er denne teknologien lett tilgjengelig for å løse problemet med kommunikasjon på tvers av operativsystem i nettverk. SOAP spesifiserer hvordan et program kan kode en XML-fil og HTTP-headeren slik at et program kan sende informasjon til et annet. SOAP spesifiserer også hvordan responsen skal håndteres.

SOAP ble utvikled av Microsoft, DevelopMentor and Userland Software, og har blitt foreslått som et standardgrensesnitt til Internet Engineering Task Force (IETF). Det er noenlunde likt Internet Inter-ORG Protocol (IIOP), som er en del av Common Object Request Broker Architecture (CORBA). Sun Microsystems' Remote Method Invocation (RMI) er en tilsvarende løsning for program skrevet i Java.

En fordel med SOAP er at forespørsler har større sannsynlighet for å komme seg forbi brannvegg-programmer som utelukker trafikk på andre enn kjente porter. Siden HTTP-forespørsler normalt er tillatt gjennomen brannvegg vil programmer som bruker SOAP kunne kommunisere med andre programmer hvor som helst.

SOAP meldinger består av tre deler: Header, Body og Attachment. Headeren beskriver meldingen med tegnsett etc. Body-delen er selve SOAP-meldingen. Attachment-delen kan brukes for å sende vedlegg til meldingen. Vedleggene trenger ikke være xml, men kan være andre typer filer, f.eks. musikkfiler eller bilder e.l. Anvendelsesområdet for SOAP-meldinger øker naturligvis av dette.

D.2 Z39.50

Z39.50 er en standard kommunikasjonsprotokoll for søking og gjenfinning av bibliografisk informasjon i nettversktilkoblete databaser. z39.50 brukes til å søke i bibliotekskataloger. Ved å bruke Z39.50 kan man lage "virtuelle" bibliotekskataloger som gjør at en informasjonssøker kan søk i noe som kan se ut som en database, mens søket i virkeligheten går mot flere databaser ved hjelp av Z39.50. Z39.50 er en ANSI/NISO standard.

D.3 The Handle System

The handle system er et system for persistent identifikasjon av informasjonsobjekter (for eksempel nettsider). En Handle er bygd opp så samme måte som en DOI, da DOI bare er en implementasjon av The Handle System. Se avsnitt D.4 DOI nedenfor.

D.4 DOI

En DOI (digital object identifier) er en permanent identifikator på et nettdokument, som gjør at man kan finne igjen dokumentet selv om Internett-adressen skulle endres. Man sender en DOI til en sentralt styrt katalog og bruker deretter adressen til katalogen og DOI'en sammen i stedet for en vanlig Internett-adresse. DOI systemet drives i dag en the International DOI Foundation, men ble utviklet av The Association of American Publishers i samarbeid med the Corporation for National Research Initiatives.

For å finne et dokumentet en DOI peker på, sender man en URL til en DOI-resolver (løser), som deretter sender tilbake en URL til selve dokumentet. en DOI kan se ut som følger: 10.1038/nm744. Tallene før skråstreken angir en katalog som man ser for seg kan vedlikeholdes av andre aktører enn DOI-stiftelsen. Tallen etter skråstreken angir identifikasjon av dokumentet. For å finne dokumentet som identifiseres av en DOI må man angi hvilken resolver man vil bruke, og dette vil gi en DOI som ser slik ut:

http://www.doi.org/10.1038/nm744

D.5 Dublin Core

Dublin Core er et initiativ for å lage et digitalt "bibliotekskatalogkort" for Internett. Dublin Core består av 15 metadataelementer (fra [23]):

- Title: Et navn gitt ressursen.
- Creator: En enhet som er hovedansvarlig for ressursens intellektuelle innhold.
- Subject: Et uttrykk for ressursens emnemessige innhold.
- Description: En redegjørelse for ressursens innhold.
- Publisher: En enhet som er ansvarlig for å ha gjort ressursen tilgjengelig.
- Contributor: En enhet som har bidratt til ressursens innhold.
- Date: En dato knyttet til en hendelse i levetiden til en ressurs.
- Type: En egenskap ved ressursen eller en sjangerbetegnelse som beskriver ressursens innhold.

- Format: En ressurs' fysiske eller digitale form.
- Identifier: En entydig henvisning til ressursen i en gitt kontekst.
- Source: En referanse til den ressursen som foreliggende ressurs er utledet av.
- Language: Språket som ressursens intellektuelle innhold er skrevet på.
- Relation: En henvisning til en beslektet ressurs.
- Coverage: Utstrekningen til eller omfanget av ressursens innhold.
- Rights: Informasjon om rettigheter knyttet til ressursen.

Det finnes to typer Dublin Core: Enkel og kvalifisert. Enkel type bruker bare tekst i metadata-elementene for å beskrive et objekt, mens kvalifisert i tillegg beskriver hvordan inkodingsskjema som er brukt i beskrivelsen.

Dublin Core er ment for å benyttes av mange. Derfor er det fokusert på enkelhet i antallet og beskrivelsen av metadataelementene. Man trenger derfor ikke være bibliotekar for å kunne beskrive et informasjonsobjekt ved hjelp av Dublin Core.

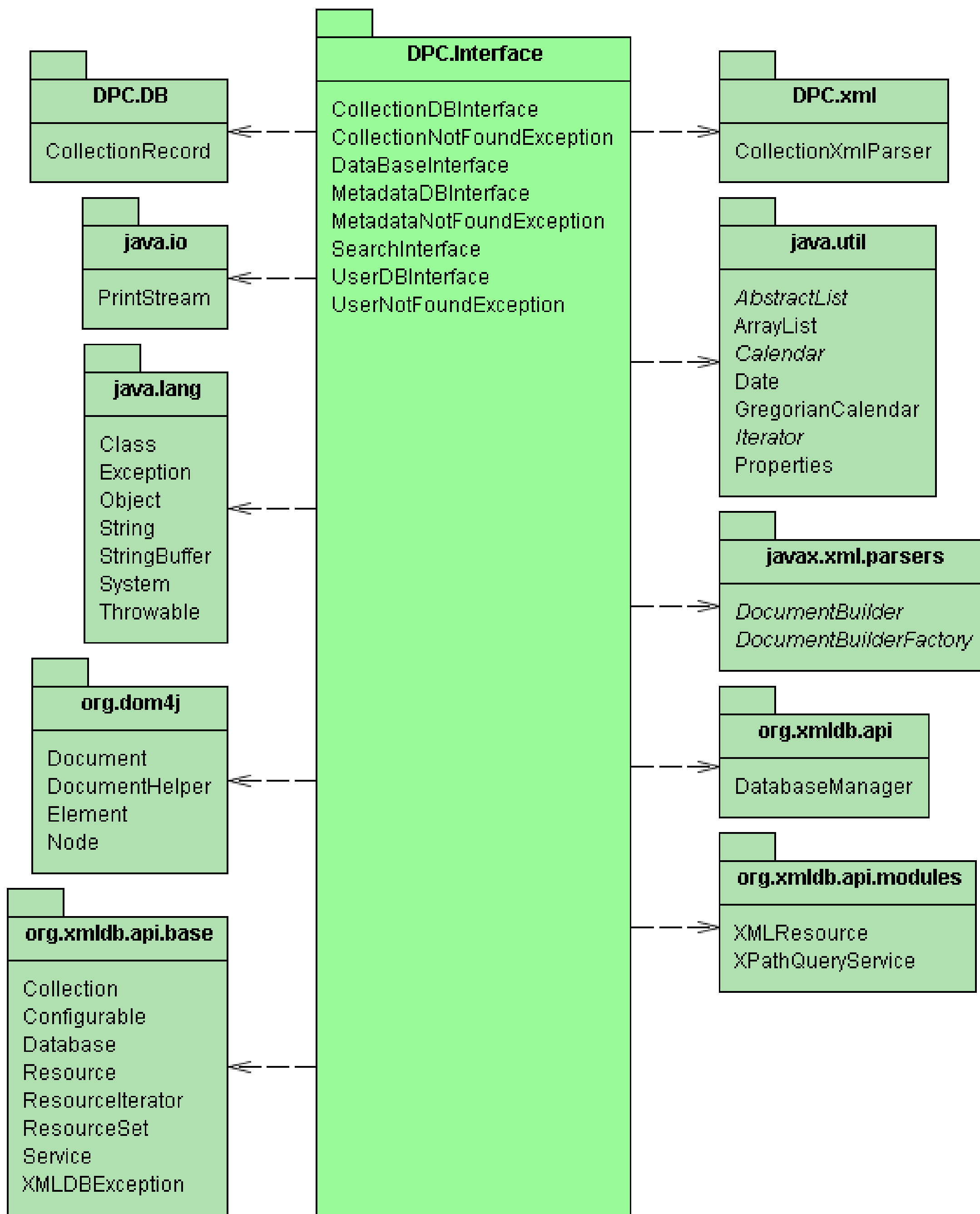
D.6 WebServices

WebServices eller Nett-tjenester er en standard som lar programmer kommunisere med hverandre over Internett. WebServices er ikke noe nytt, og standarder som EDI (Electronic Data Interchange) har eksistert lenge. EDI brukes i dag av bedrifter til å utveksle dokumenter, bestillinger og fakturaer etc. WebServices er en plattform som kan avløse EDI.

WebServices beskrives ved hjelp av WebServices Description Language (WSDL), som uttrykkes i XML. Ved at en WebService beskriver seg selv i en WSDL-fil, kan andre tjenester benytte denne beskrivelsen til å lage meldinger som kan forstås av WebService'en. Slike meldinger mellom WebServices sendes med via SOAP (Se «SOAP» på side 129.).

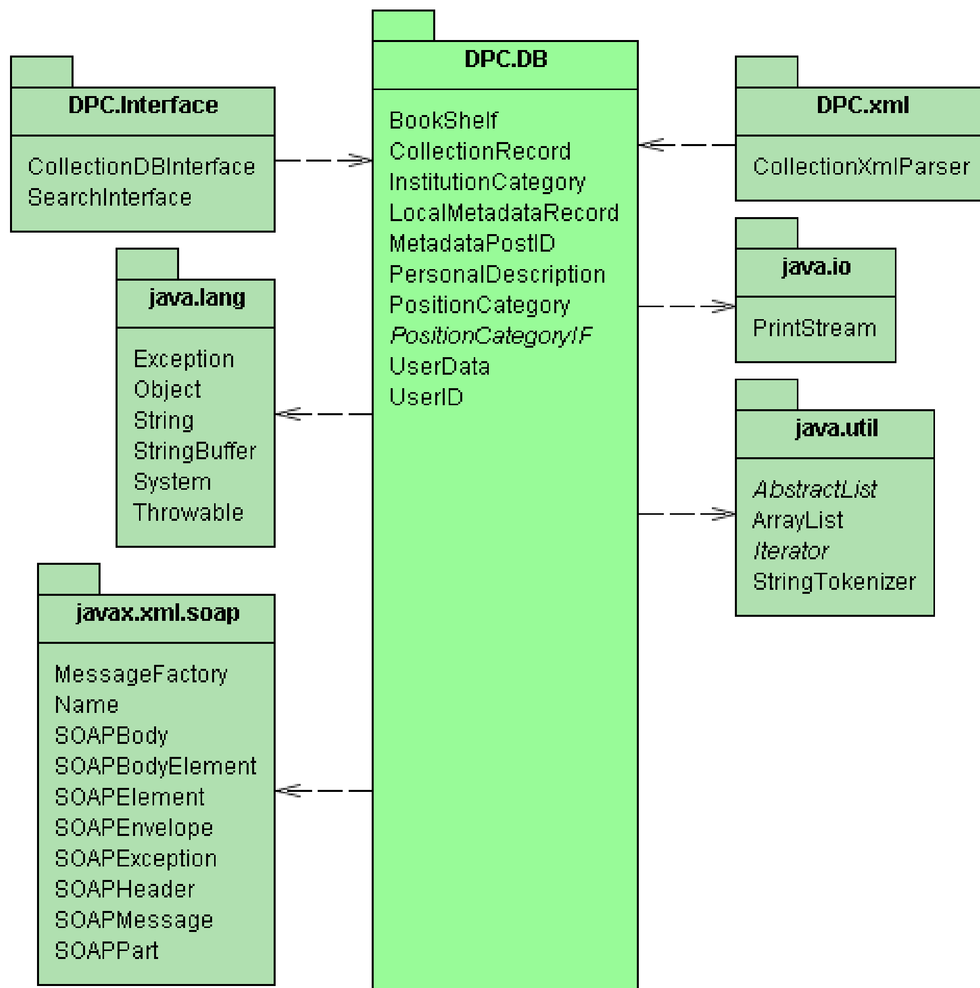
APPENDIX E KLASSEDIAGRAMMER

E.1 DPC.Interface



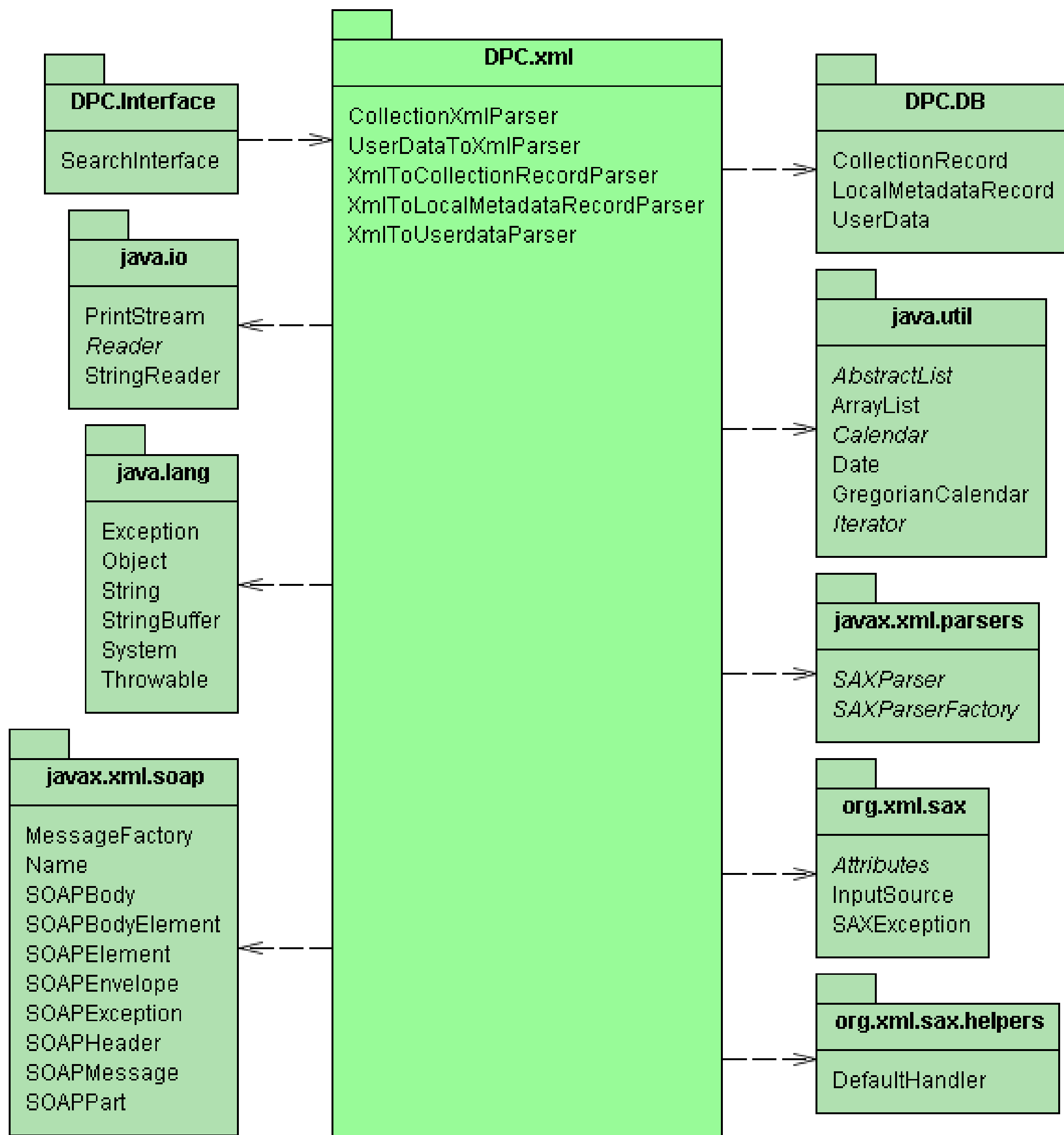
Figur E–1: Umldiagramfor pakken DPC.Interface

E.2 DPC.DB



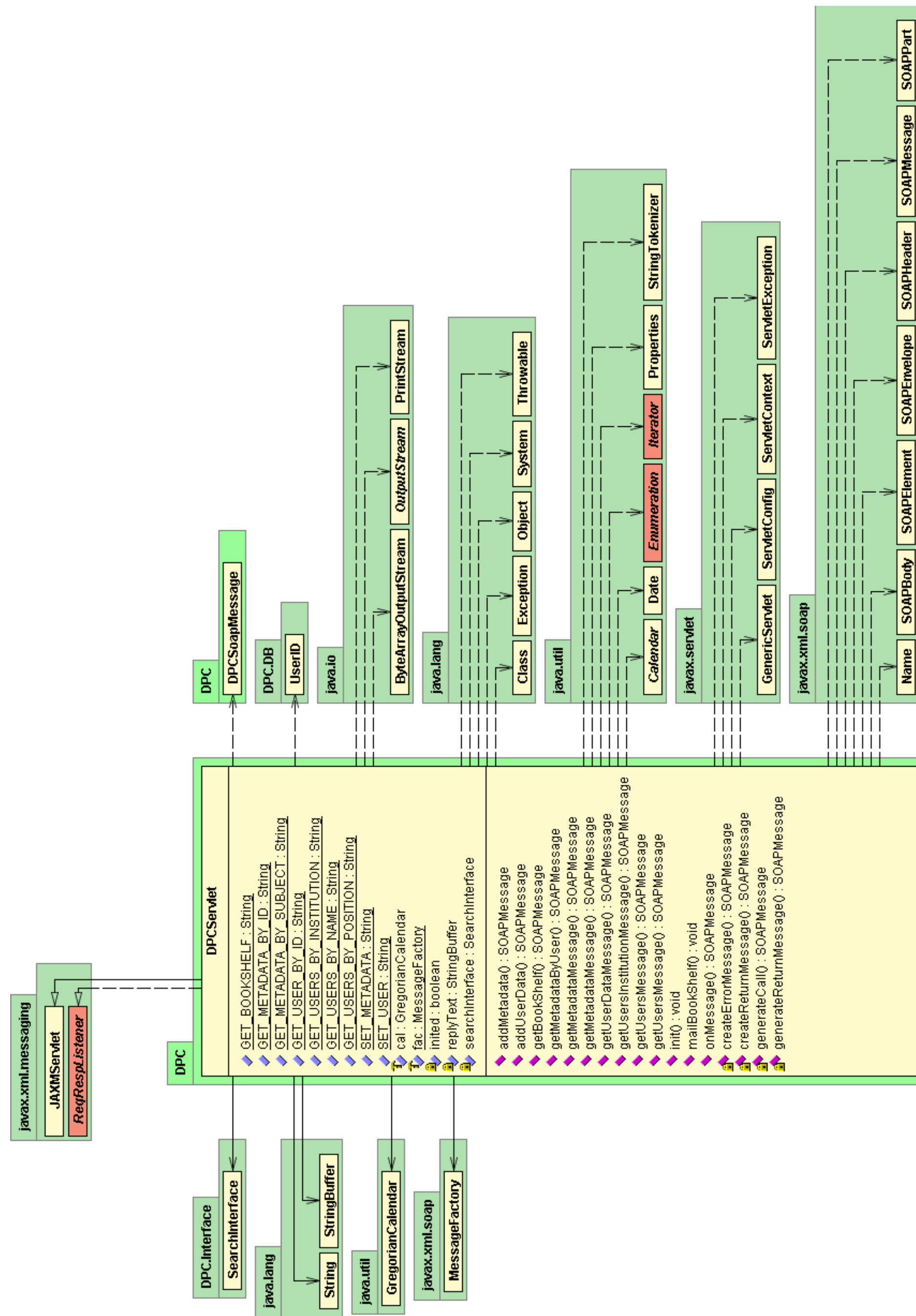
Figur E–2: Uml-diagram for pakken DPC.DB

E.3 DPC.xml



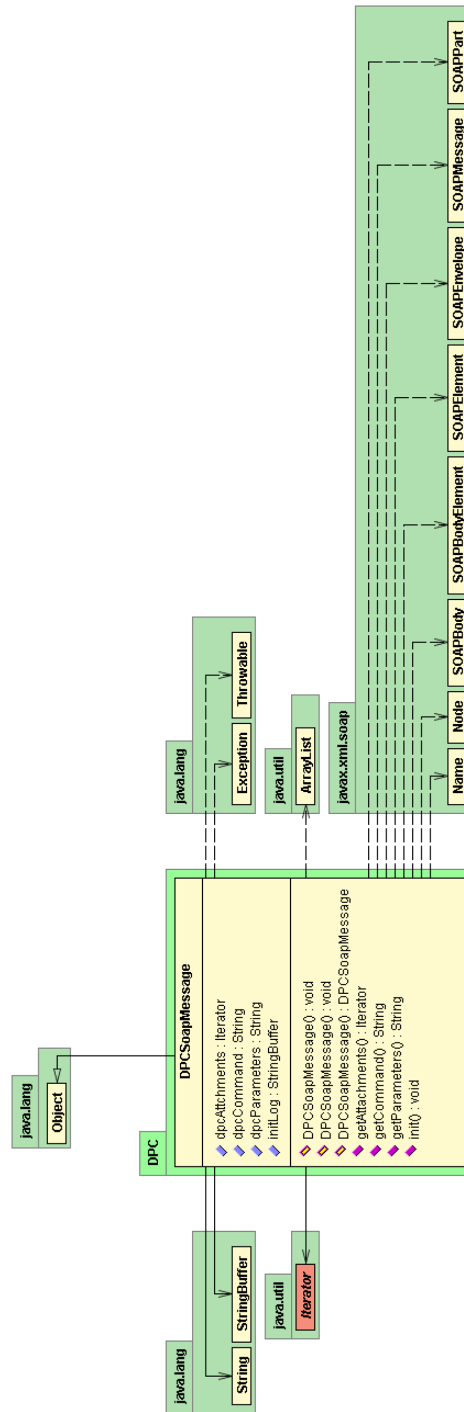
Figur E–3: Uml-diagram for pakken DPC.xml

E.4 DPC.DPCServlet



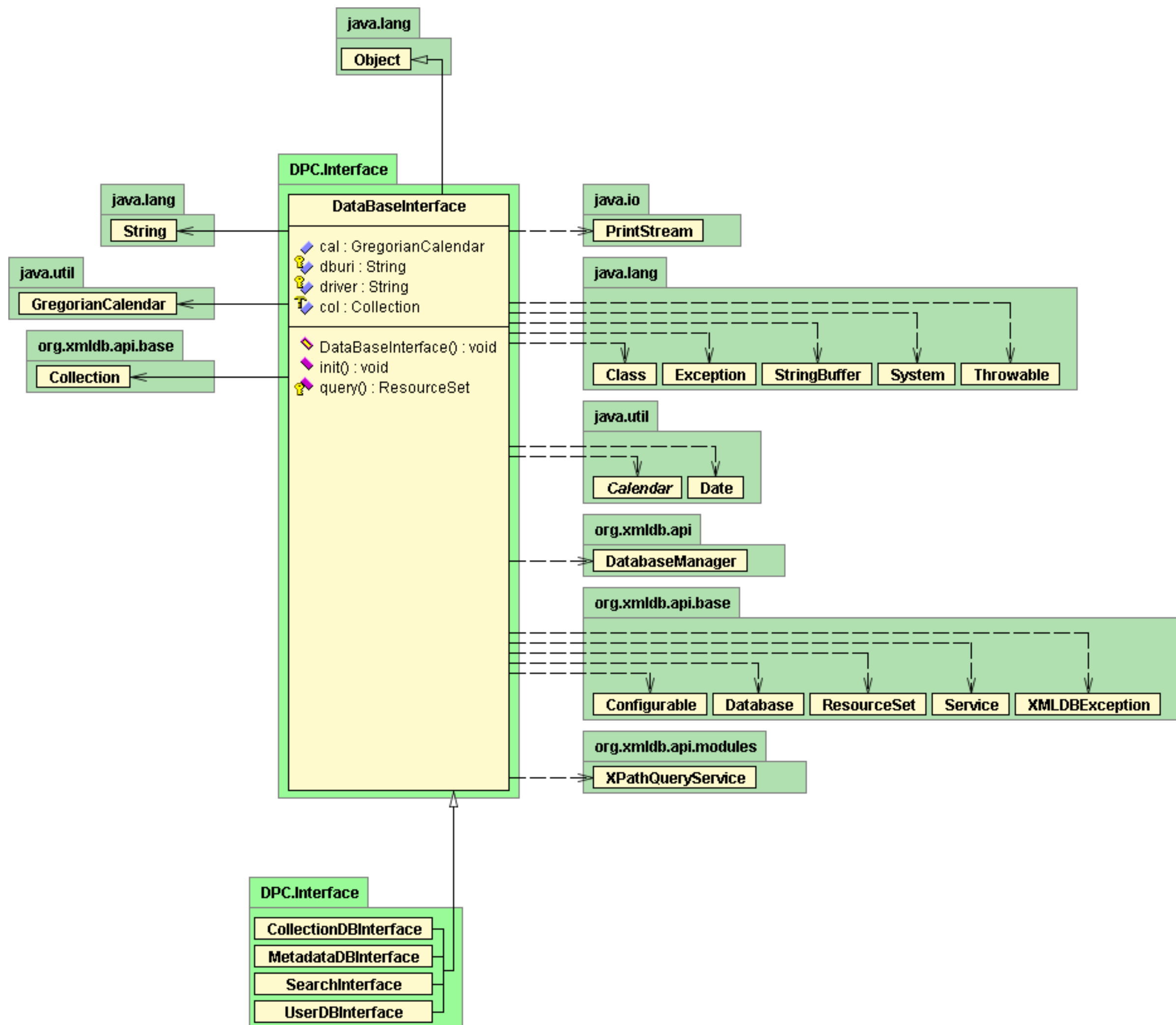
Figur E–4: Klassediagram for DPCServlet

E.5 DPC.DPCSoapMessage



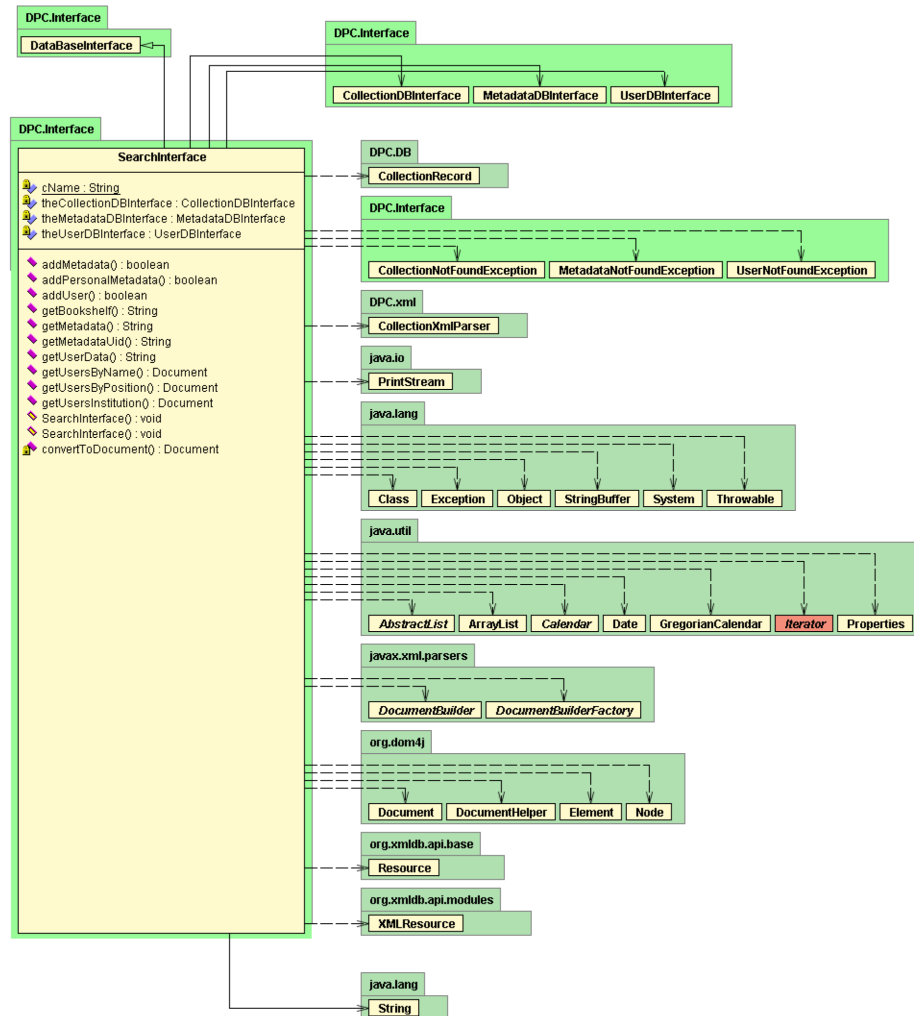
Figur E–5: Klassediagram for DPCSoapMessage

E.6 DPC.Interface.DatabaseInterface



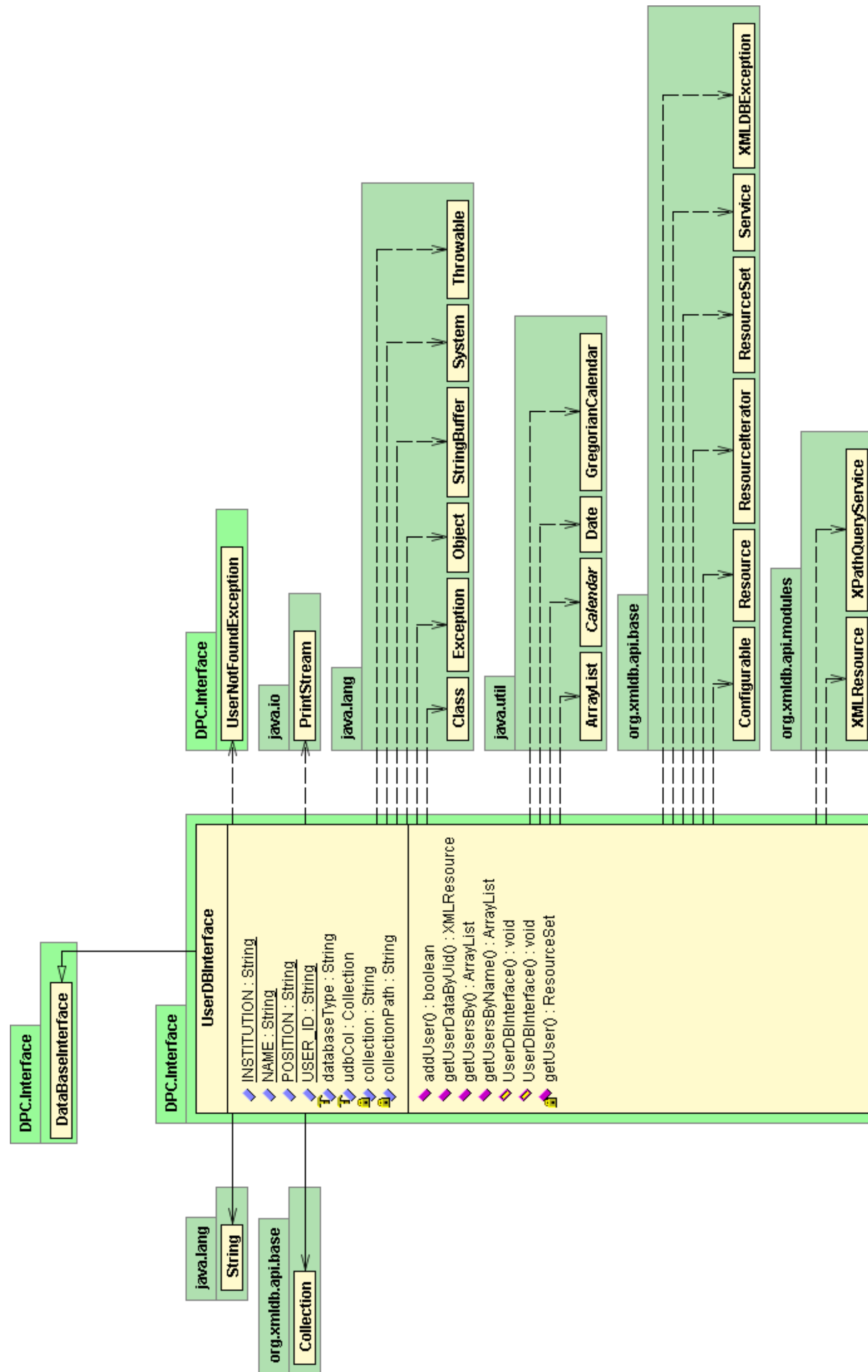
Figur E–6: Klassediagram for DatabaseInterface

E.7 DPC.Interface.SearchInterface



Figur E–7: Klassediagram for SearchInterface

E.8 DPC.Interface.UserDBInterface

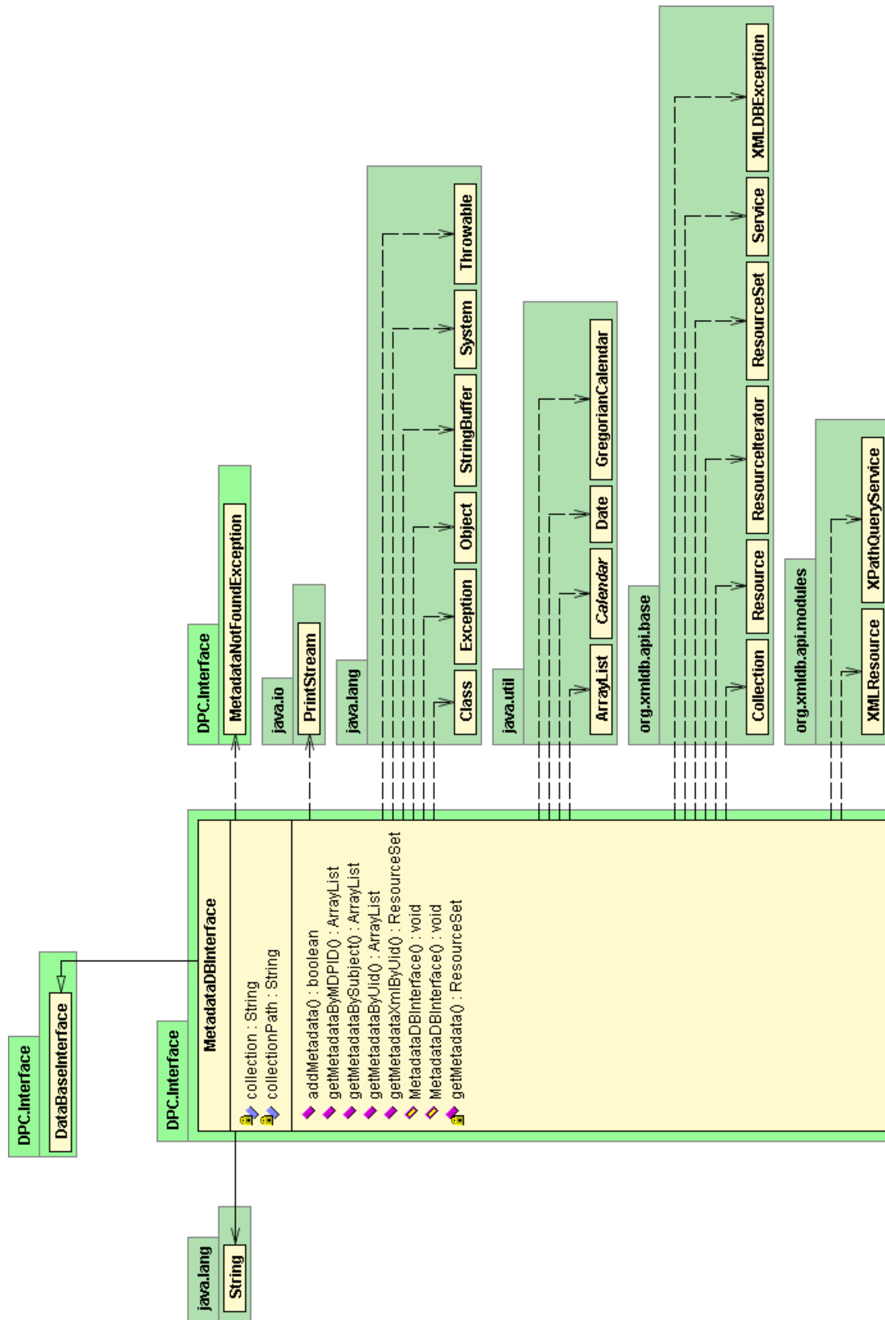


Figur E–8: Klassediagram for UserDBInteracе

E.9 DPC.Interface.CollectionDBInterface

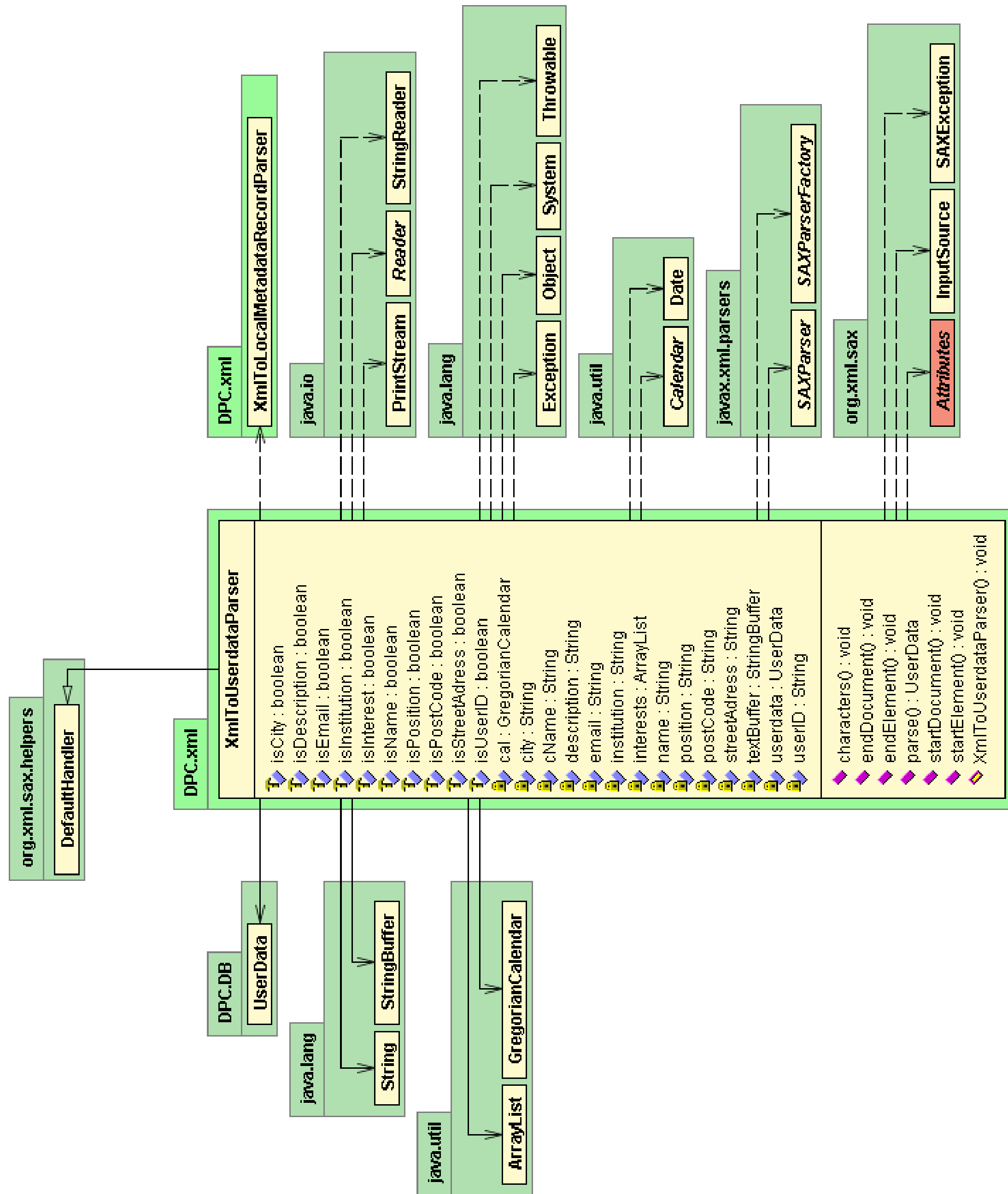
Figur E–9: Klassediagram for CollectionDBInterface

E.10 DPC.Interface.MetadataDBInterface



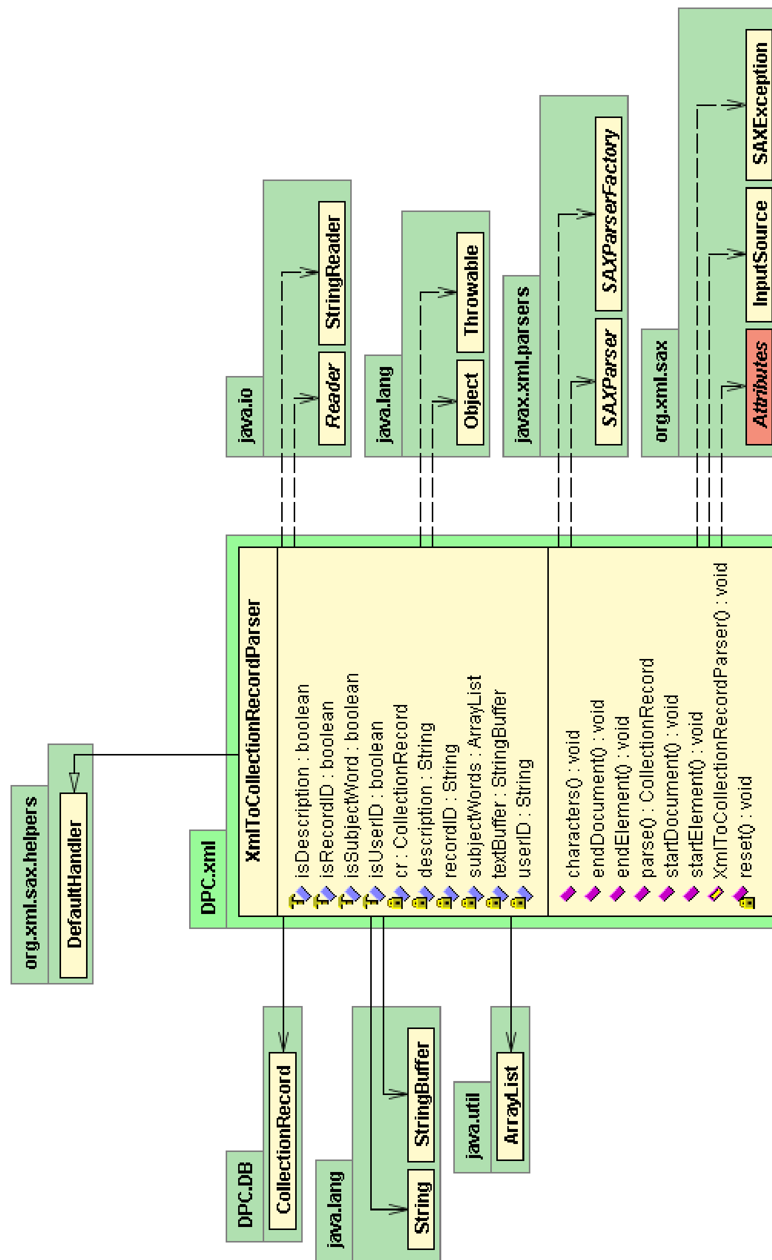
Figur E–10: Klassediagram for MetadataDBInterface

E.11 DPC.xml.XmlToUserDataParser



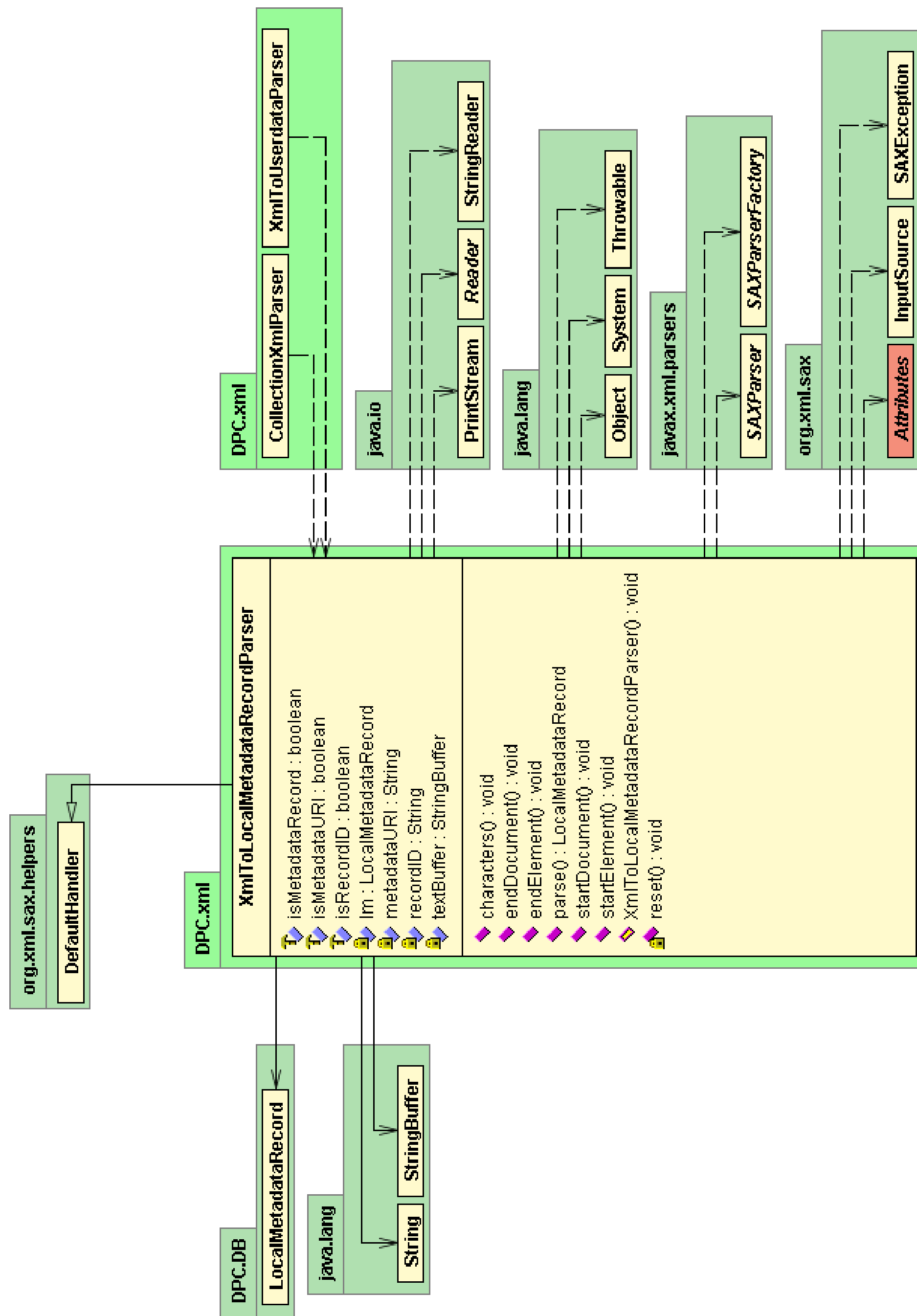
Figur E–11: Klassediagram for XmlToUserDataParser

E.12 DPC.xml.XmlToCollectionRecordParser



Figur E–12: Klassediagram for XmlToCollectionRecordParser

E.13 DPC.xml.XmlToLocalMetadataRecordParser

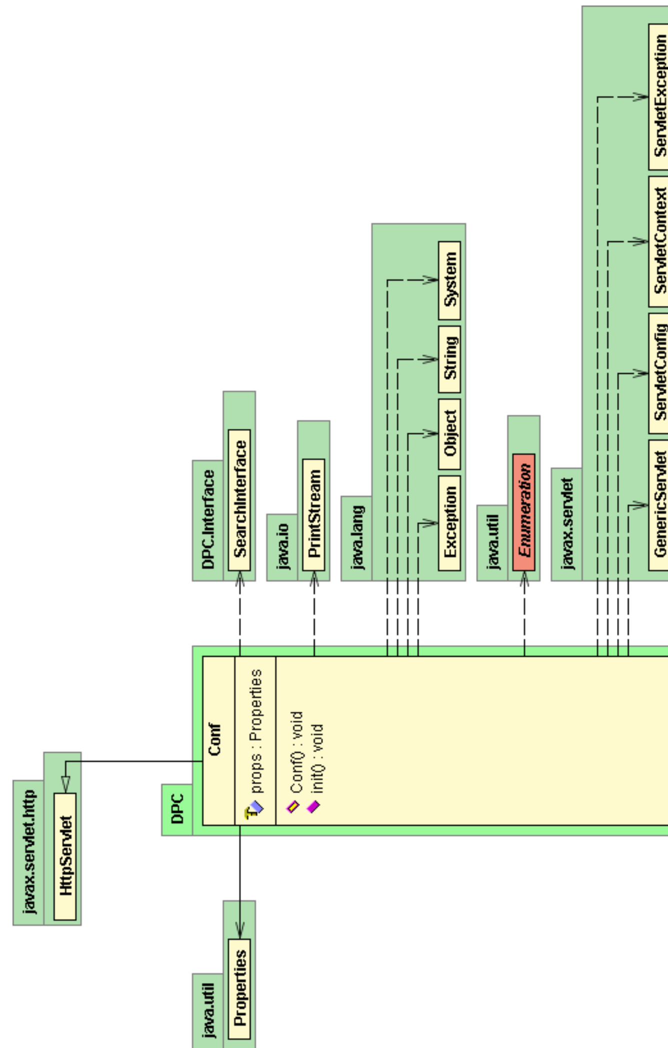


Figur E–13: Klassediagram for XmlToLocalMetadataRecordParser

E.14 DPC.xml.UserDataToXmlParser

Figur E–14: Klassediagram for UserDataToXmlParser

E.15 DPC.Conf



Figur E–15: Klassediagram for Conf

APPENDIX F PROGRAM DOKUMENTASJON

F.1 DPC.DPCSOAPMessage

DPC

Class DPCSoapMessage

public class **DPCSoapMessage**

Title: PDC WebService

Description: DPC Servlet/WebService

This is an class for the representation of an internal DPC Message. This message had information about what command and parameters the client sent, and has a structure to contain the attachements sent in a SOAPMessage.

Version:

1.0

Author:

Sverre Joki

Field Detail

dpcCommand

`public java.lang.String` **`dpcCommand`**

The command in ths SOAPMessage

---

dpcParameters

`public java.lang.String` **`dpcParameters`**

The parameters in the SOAP message

---

initLog

```
public java.lang.StringBuffer initLog
```

This attribute is used to logging.

---

dpcAttchments

```
public java.util.Iterator dpcAttchments
```

This attribute is used to iterate over all attachments in the

---

Constructor Detail

DPCSoapMessage

```
public DPCSoapMessage()
```

Constructs an empty DPCSoapMessage

DPCSoapMessage

```
public DPCSoapMessage(java.lang.String command,
                      java.lang.String parameter,
                      java.util.Arrays$ArrayList attachments)
```

Constructs a DPCSoapMessage with the data supplied in the parameters. This means that the object will be initialized with a command, a parameter and attachments.

Parameters:

`String` - command the command that the SOAPMessage wanted

`String` - parameter the parameters to the given command

`ArrayList` - attachments the list of attachements that cam along the SOAP message

---

DPCSoapMessage

```
public DPCSoapMessage(javax.xml.soap.SOAPMessage msg)
```

This contructor uses an incoming SOAPMessage to create a DPCSoapMessage object. The SOAPMessage will be parsed to identify the command, parameters and attachments, and the attributes of the object will be set accordingly.

Parameters:

`SOAPMessage` - the SOAPMessage you want to create a DPCSoapMessage from

---

Method Detail

init

```
public void init(javax.xml.soap.SOAPMessage msg)
```

This method uses an incoming SOAPMessage to initialize this DPCSoapMessage object. The SOAPMessage will be parsed to identify the command, parameters and attachments, and the attributes of the object will be set accordingly.

Parameters:

`SOAPMessage` - the SOAPMessage you want to create a DPCSoapMessage from

Returns:

void

---

getCommand

```
public java.lang.String
getCommand(javax.xml.soap.SOAPMessage msg)
```

This method fetches the command from a SOAPMessage.

Parameters:

`SOAPMessage` - the SOAPMessage you want to fetch the command from

Returns:

String the string value of the command

---

getParameters

```
public java.lang.String
getParameters(javax.xml.soap.SOAPMessage msg,
                                          java.lang.String cmd)
```

Fetches the parameters from the input. The parametes are returned in a String containing all parameters as they appear in the soap message. That is, as space separated words.

Parameters:

`SOAPMessage` - the SOAPMessage you want to fetch the parameters from

`String` - the command that has the parameters you want

Returns:

String the string value of the parameters.

---

getAttachments

```
public java.util.Iterator getAttachments()
```

This method gives you an Iterator of all attachments in this object.

F.2 DPC.DPCServlet

DPC

Class DPCServlet

public class **DPCServlet**

extends javax.xml.messaging.JAXMServlet

implements javax.xml.messaging.ReqRespListener

This is the main server part of the DPC system. The DPCServlet handles all resuests to the DPC system. The system is activaed by a call to the onMessage method. This method handles the recieveing and sending of messages.

Author:

Sverre Joki

Field Detail

GET_BOOKSHELF

`public static java.lang.String` **`GET_BOOKSHELF`**

String constant identifying this command

GET_USER_BY_ID

`public static java.lang.String` **`GET_USER_BY_ID`**

String constant identifying this command

GET_USERS_BY_NAME

`public static java.lang.String` **`GET_USERS_BY_NAME`**

String constant identifying this command

GET_USERS_BY_INSTITUTION

`public static java.lang.String` **`GET_USERS_BY_INSTITUTION`**

String constant identifying this command

GET_USERS_BY_POSITION

`public static java.lang.String` **`GET_USERS_BY_POSITION`**

String constant identifying this command

GET_METADATA_BY_ID

`public static java.lang.String` **`GET_METADATA_BY_ID`**

String constant identifying this command

GET_METADATA_BY_SUBJECT

`public static java.lang.String` **`GET_METADATA_BY_SUBJECT`**

String constant identifying this command

SET_USER

`public static java.lang.String` **`SET_USER`**

String constant identifying this command

SET_METADATA

`public static java.lang.String` **`SET_METADATA`**

String constant identifying this command

---

searchInterface

`private DPC.Interface.SearchInterface` **`searchInterface`**

This attribute is the SearchInterface object that connects the DPCServlet to the databases.

---

replyText

`private java.lang.StringBuffer` **`replyText`**

stringbuffer to keep reply text

---

inited

`private boolean` **`inited`**

True if the servlet is initialized

---

fac

`static javax.xml.soap.MessageFactory` **`fac`**

The object responsible for creating new empty messages

---

cal

`java.util.GregorianCalendar` **`cal`**

Kalenderobjeckt som brukes i debugsammenheng

Method Detail

init

`public void` **`init`**`(javax.servlet.ServletConfig servletConfig)`

This method initializes the serlvet, and sets up the searchInterface class to handle requests from this servlet.

Parameters:

`ServletConfig` -

Throws:

`ServletExpetion` - if anything goes very wrong

Author:

Sverre Joki

---

onMessage

`public javax.xml.soap.SOAPMessage`
`onMessage``(javax.xml.soap.SOAPMessage message)`

This is the application code for handling the message. Once the message is received the application can retrieve the soap part, the attachment part if there are any, or any other information from the message. This method converts a SOAP message to an internal format that can be handled by the DPC server.

Parameters:

`SOAPMessage` - the SOAP message to process

Returns:

SOAPMessage the returned SOAP message with the results from the processing

Author:

Sverre Joki

---

generateReturnMessage

```
private javax.xml.soap.SOAPMessage generateReturnMessage()
```

Based on generateCall and subsequent calls, this method generates a SOAPMessage to return to the caller

Returns:

SoapMessage the soapMessage to be returned, null if errors occur

Author:

Sverre Joki

---

generateCall

```
private javax.xml.soap.SOAPMessage
generateCall(DPC.DPCSoapMessage msg)
```

This method generate a call to the corrert method for subsequent handling of the request based on the DPCSoapMessage object.

Parameters:

`dpcSoapMessage` - the DPCSoapMessage object generated in the omMessage method

Returns:

SOAPMessge the message to be returned to the user

Author:

Sverre Joki

---

getUserDataMessage

```
public javax.xml.soap.SOAPMessage
getUserDataMessage(java.lang.String uid)
```

Generates a soapmessage containing the userdata for the given userid. Usually uses getUserData(String uid) to fetch the userdata.

Parameters:

`uid` - the userId of the person in question

Returns:

SOAPMessage the soapmessage contianing the requested userInfo

---

getUsersMessage

```
public javax.xml.soap.SOAPMessage
getUsersMessage(java.lang.String userName)
```

This method returns a message containg users with the given name

Parameters:

`String` - the user name you want to be found

Author:

Sverre Joki

---

getUsersInstitutionMessage

```
public javax.xml.soap.SOAPMessage
getUsersInstitutionMessage(java.lang.String institution)
```

This method finds all useres which belongs to a give institution.

Parameters:

`String` - the institution in question

Returns:

SOAPMessage containing the matchin users in the given institution

Author:

Sverre Joki

---

getUsersMessage

```
public javax.xml.soap.SOAPMessage
getUsersMessage(DPC.DB.PositionCategory positionCategory)
```

This method finds all usres with a given position within any instituion.

Parameters:

`PositionCategory` - The position categoy in question

Returns:

ArrayList containing the matching users with the given positions

Author:

Sverre Joki

---

addUserData

```
public javax.xml.soap.SOAPMessage
addUserData(DPC.DPCSoapMessage dpcMsg)
```

This method adds an usre to the database. It then generates a return message telling whoever want to read it, that all went well. If the update doesn't go well, the returnmessage tells you that.

Parameters:

`DPCMessage` - the dpc message containing

Returns:

SOAPMessage

Author:

Sverre Joki

---

getMetadataMessage

`public javax.xml.soap.SOAPMessage`
`getMetadataMessage``(java.lang.String mdpid)`

Creates a SOAPMessage containing the LocalMetadataREcord matching the MetadataPostID input.

Parameters:

`String` - The id of the metadatarecord in question

Returns:

SOAPMessage a message containing the result from the search

Author:

Sverre Joki

---

getMetadataByUser

`public javax.xml.soap.SOAPMessage`
`getMetadataByUser``(java.lang.String userid)`

Creates a SOAPMessage containing the LocalMetadataREcord matching the UserID input.

Parameters:

`userID` - the userId you want to search for in the metadatabase.

Returns:

SOAPMessage contining the metadata record for the given userID

---

getMetadataMessage

`public javax.xml.soap.SOAPMessage`
`getMetadataMessage``(java.util.Arrays$ArrayList subjectWords)`

Generates a SOAPMessage containing records whith corresponding subjectwords

Parameters:

`String` - the subject words in question

Returns:

SOAPMessage a message containing the result from the search

Author:

Sverre Joki

---

addMetadata

```
public javax.xml.soap.SOAPMessage
addMetadata(DPC.DPCSoapMessage dpcMsg)
```

This method adds a record to the metadata-database in the DPC system.

Parameters:

`dpcMsg` - the DPCSoapMessage containing the metadatarecord you want to add to the database

Returns:

SOAPMessage indicating the success of the request

Author:

Sverre Joki

---

getBookShelf

```
public javax.xml.soap.SOAPMessage
getBookShelf(java.lang.String uid)
```

Gets the bookshelf of the user indicated by the user id. - get the userdata from udb - get the metadata from the

Parameters:

`UserID` -

Returns:

void

Throws:

---

mailBookShelf

`public void` **`mailBookShelf`**`(DPC.DB.UserID uid)`

THIS METHOD sends an bookshelf by to the user that is specified by the userID. Not in use.

Parameters:

`UserID` -

Returns:

void

Throws:

---

createReturnMessage

`private javax.xml.soap.SOAPMessage` **`createReturnMessage`**`(java.lang.String message)`

This method creates a simple SOAPmessage containing some text. The metod takes as argument a string which represents the message that is to be returned to the SOAP message sender.

Parameters:

`String` - the message text to be included in the message

Returns:

SOAPMessage the soap message that is generated by this method

Author:

Sverre Joki

---

createErrorMessage

```
private javax.xml.soap.SOAPMessage
createErrorMessage(java.lang.String message)
```

This method creates a simple SOAPmessage indicating errors. The metod takes as argument a string which represents the errormessage that is to be returned to the SOAP message sender.

Parameters:

`String` - the message text to be included in the errormessage

Returns:

SOAPMessage the soap message that is generated by this method

Author:

Sverre Joki

F.3 DPC.Conf

DPC

Class Conf

public class **Conf**
extends javax.servlet.http.HttpServlet

Title: PDC WebService Server

Description: DPC Servlet/WebService

Copyright: Copyright (c) 2002

Company:

Version:
1.0

Denne klassen tar seg av initialisering av DPC servleten.

systemet har flere klasser som bruker samme konfig, og denne klassen sørger for å laste denne konfigen inn i servletkonteksten.

Author:
Sverre Joki

Field Detail

props

`java.util.Properties props`

Dette er objektet som vi lagrer alle init-parametrene fra web.xml fila i. Dette objektet lagres i servlet konteksten og er dermed tilgjengeli i alle servletene som kjører i denne applikasjonen. Dette åpner for at vi kan lagre mange verdier her om vi trenger.

Constructor Detail

Conf

`public Conf()`

Tom konstruktør for et konfigurasjonsobjekt. Denne klassen kalles når servlet serveren (tomcat) starter. Dette er satt opp i web.xml fila somligger i WEB-INF katalogen for DPC applikasjonen. Når Conf objektet kallesvil det først kalle init metoden som er en del av Servlet strukturen. Dette skjer automatisk så lenge load-on-startup er satt i web.xml.

Method Detail

init

`public void init(ServletConfigconf)`

Setter opp conigen til servletene. Laster inn parametrene fra web.xmlfila, og setter dem inn i et properies objekt som igjen gjøres tilgjengelig i servletcontexten til denne servlet maskina. For å bruke dette objektet må man kalle

```
ServletContext ctx = conf.getServletContext();
Propserties p = (Properties)ctx.getAttribute("conf");
p.getProperty("dpc.serv.url");
```

Parameters:

`conf`- the ServletConfig

Returns:

void

Throws:

`ServletException`- in case of errors

F.4 DPC.DB

Package DPC.DB

Class Summary

BookShelf
CollectionRecord
LocalMetadataRecord
MetadataPostID
PersonalDescription
UserData

F.5 DPC.DB.BookShelf

DPC.DB

Class BookShelf

public class **BookShelf**

Title: PDC WebService

Description: DPC Servlet/WebService

This class represent a BookShelf(personal collection) for a single user.

Version:

1.0

Author:

Sverre Joki

Field Detail

userData

`private DPC.DB.UserData **userData**`

This attribte contains the user data for the given bookshelf

---

personalDescriptions

`private java.util.ArrayList **personalDescriptions**`

This attribute contains the personal desctiptions for the given bookshelf

---

mf

`private javax.xml.soap.MessageFactory **mf**`

Object used to create a SOAPMessage.

---

msg

`private javax.xml.soap.SOAPMessage **msg**`

A SOAPMessage that is returned inthe methods

---

sp

`private javax.xml.soap.SOAPPart **sp**`

Part of a SOAPMessage that is used when parsing the UserData object, and creating the SOAPMessage

---

envelope

```
private javax.xml.soap.SOAPEnvelope envelope
```

Part of a SOAPMessage that is used when parsing the UserData object, and creating the SOAPMessage

---

hdr

```
private javax.xml.soap.SOAPHeader hdr
```

Part of a SOAPMessage that is used when parsing the UserData object, and creating the SOAPMessage

---

bdy

```
private javax.xml.soap.SOAPBody bdy
```

Part of a SOAPMessage that is used when parsing the UserData object, and creating the SOAPMessage

---

bodyElement

```
private javax.xml.soap.SOAPBodyElement bodyElement
```

Part of a SOAPMessage that is used when parsing the UserData object, and creating the SOAPMessage

---

Constructor Detail

BookShelf

```
public BookShelf()
```

This contructor creates an BookShelf object, but does not add any data to the object.

Method Detail

getPersonalDescriptions

`public java.util.ArrayList getPersonalDescriptions()`

This method retrieves an ArrayList of personal descriptions in this bookshelf.

Returns:

ArrayList an array of personal desctiption (PersonaDescription objects)

---

addPersonalDescription

`public void addPersonalDescription(DPC.DB.PersonalDescription desc)`

This method adds a Local MetadataRecord to the ArrayList containg all PersonalDescription objects.

Parameters:

`desc` - the PersonalDescription object to add to tha ArrayList

Returns:

void

---

setPersonaDescriptions

`public void setPersonaDescriptions(java.util.ArrayList list)`

This method sets the objects in the ArrayList, list, into the PersonalDescription ArrayList in this object.

Parameters:

`list` - the ArrayList containing the objects to add to the PersonalDescription ArrayList

Returns:

void

---

getUserData

`public DPC.DB.UserData **getUserData**()`

Returns the userdata in this bookshelf

Returns:

UserData a object describing the user

---

setUserData

`public void **setUserData**(DPC.DB.UserData innUserData)`

Sets the user in this bookshelf.

Parameters:

`innUserData` - the user data to use as user in this bookshelf

Returns:

void

---

sendByMail

`public boolean **sendByMail**()`

This method send this bookshelf by mail to the user already registered in the bookshelf. Returns a boolean indicating success.

NOT IMPLEMENTED YET!

Returns:

boolean true if the sending goes ok, false if not.

---

convertToXml

`public java.lang.String **convertToXml**()`

This method converts this bookshelf to an xml document that is returned in a pure text string.

Returns:

String the xml document string

Throws:

`Exception` - if something goes wrong

---

getBookShelfXmlStart

```
public java.lang.String getBookShelfXmlStart()
```

This method creates a x ml document header for a Bookshelf xml document.

Returns:

String the xml document header text.

F.6 DPC.DB.CollectionRecord

DPC.DB

Class CollectionRecord

public class **CollectionRecord**

Title: PDC WebService

Description: DPC Servlet/WebService

This class is an internal representation of a record in the collection database. This object keeps track of the personal description, user id, and subject word of this record.

Version:

1.0

Author:

Sverre Joki

Field Detail

recordID

```
private java.lang.String recordID
```

This attribute holds the value for the reocrd Id of this record in the collections database

---

userID

```
private java.lang.String userID
```

This attribute has the value for the userID of the user that has written the personal desctiption

---

subjectWords

```
private java.util.ArrayList subjectWords
```

This attribute has the subject words that the given user has used to describe the personal desription

---

personalDescription

```
private java.lang.String personalDescription
```

This attribute holds the personal descripton or personal metadata that a user has added to a metadata record.

Constructor Detail

CollectionRecord

```
public CollectionRecord()
```

Constructs an empty CollectionRecord object

CollectionRecord

```
public CollectionRecord(java.lang.String uid,
                        java.lang.String recordID,
                        java.util.ArrayList subjectWordsIn,
                        java.lang.String description)
```

This constructor creates a CollectionRecord object using the given paramters.

Parameters:

`uid` - the userId to use for the CollectionRecord

`recordID` - the recordId to use for the CollectionRecord

`subjectWordsIn` - the subject words to use for the CollectionRecord

`description` - the description to use for the CollectionRecord

Method Detail

reset

```
public void reset()
```

Tis method resets all attributes within this object to null.

Returns:

void

---

setUserID

```
public void setUserID(java.lang.String uid)
```

This method sets the user id of this object to the value specified in the parameter uid.

Parameters:

`uid` - the user id to use as userID in this object

Returns:

void

---

setRecordID

`public void setRecordID(java.lang.String recordID)`

This method insterts the recordID string into the recordID attribute of this object.

Parameters:

`recordId` - the value to add to the recordID of this object

Returns:

void

---

setDescription

`public void setDescription(java.lang.String description)`

This method sets the description of this object to the value specified in the parameter description.

Parameters:

`description` - the description to use as description in this object

Returns:

void

---

addsubjectWord

`public void addsubjectWord(java.lang.String word)`

This method adds a subject word to the list of subject words in this object. The word to be added is supplied in a string parameter.

Parameters:

`word` - the word to add to the subjectWords in this object

Returns:

void

---

setSubjectWords

`public void **setSubjectWords**(java.util.ArrayList words)`

This method sets the subject words to use in this object. The subject words to be used is supplied in a ArrayList parameter.

Parameters:

`words` - the ArrayList of words to use as the subjectWords in this object

Returns:

void

---

setSubjectWords

`public void **setSubjectWords**(java.lang.String[] words)`

This method sets the subject words to use in this object. The subject words to be used is supplied in a string array parameter.

Parameters:

`words` - the string array of words to use as the subjectWords in this object

Returns:

void

---

getUserID

`public java.lang.String **getUserID**()`

This method retrievs the userID used in this object

Returns:

String the userId of this object

---

getDescription

`public java.lang.String getDescription()`

This method retrievs the personalDescription used in this object

Returns:

String the personaDescription of this object

---

getsubjectWords

`public java.util.ArrayList getsubjectWords()`

This method retrievs the subjectWords used in this object

Returns:

ArrayList the subjectwords of this object

---

getRecordID

`public java.lang.String getRecordID()`

This method retrieves the record ID for this object.

Returns:

String the record id for this object

---

convertToXml

`public java.lang.String convertToXml()`

This method converts this object to a xml document and also adds a xml document header.

Returns:

String this object converted to a well formed xml document

F.7 DB.DPC.LocalMetadataRecord

DPC.DB

Class LocalMetadataRecord

public class **LocalMetadataRecord**

This class is an internal representation of a local metadata record. This class has attribuites to keep a list of personal desriptions and a list of users. There is also an URI attribute, to link descriptions and users to a metadata record.

Author:

Field Detail

localRecordID

`private java.lang.String` **`localRecordID`**

This attribute is the id for the loca metadata record

---

descriptions

`private java.util.ArrayList` **`descriptions`**

This attribute id an ArryList containing PersonalDescripton objects

---

users

`private java.util.ArrayList` **`users`**

This attribute id an ArryList containing userIDs

---

metadataURI

`private java.lang.String` **`metadataURI`**

This attribute id an ArryList containing the URI to the metadata record

---

Constructor Detail

LocalMetadataRecord

```
public LocalMetadataRecord()
```

Constructs an empty LocalMetadata object

Method Detail

get

```
public DPC.DB.LocalMetadataRecord get()
```

This method returns the LocalMetadataRecord object.

Returns:

the LocalMetadataRecord object.

---

setRecordID

```
public void setRecordID(java.lang.String recordID)
```

This method insterts the recordID string into the localRecordID attribute of this object.

Parameters:

`recordId` - the value to add to the localRecordID of this object

Returns:

void

---

getRecordID

```
public java.lang.String getRecordID()
```

This method returns the localRecordID of this object.

Returns:

String the localRecordID of this object

---

setmetadataURI

`public void setmetadataURI(java.lang.String metadataURI)`

This method insterts the metadataURI string into the metadataURI attribute of this object.

Parameters:

`metadataURI` - the value to inset as the metadataURI of this object

Returns:

void

---

getmetadataURI

`public java.lang.String getmetadataURI()`

This method returs the metadataURI of this object

Returns:

the metadataURI of this object

---

setUsers

`public void setUsers(java.util.ArrayList users)`

This method sets the ArrayList of users of this object. The ArrayList must contain userIDs for eash user.

Parameters:

`users` - the ArrayList of users to set for this object

Returns:

void

---

getUsers

`public java.util.ArrayList getUsers()`

This method returns the ArrayList of users for this object.

Returns:

the ArrayList of users for this object.

---

addUser

`public void addUser(java.lang.String uid)`

This method adds a single userID to the ArrayList of usres in this object.

Parameters:

`uid` - the user ID to add to the ArrayList of users in this object

Returns:

void

---

convertToSOAP

`public javax.xml.soap.SOAPMessage convertToSOAP(java.util.ArrayList metaDataArray)`

This is a utility method to convert this LocalMetadataRecord intto a soapmessage ready to be sent to a client.

Parameters:

`metaDataArray` - this is an ArrayList containing LocalMetadataRecord objects that is to be converted into a SOAPMessage

Returns:

SOAPMessage a SOAPMessage of the LocalMetadataRecord objects in the parameter

---

convertToSOAP

`public javax.xml.soap.SOAPMessage` **`convertToSOAP`**`()`

Genereates a SOAPMessage based on the LocalMetadataRecord object in this object.

This method creates one SOAPMessage based on ONE userdata object

Returns:

SOAPMessage a SOAPMessage of this object

Throws:

`SOAPException` - if something goes wrong during the convertion.

---

createMetaDataElement

`private javax.xml.soap.SOAPElement`
`createMetaDataElement``(DPC.DB.LocalMetadataRecord lm)`

This method converts the given LocalMetadataRecord into a SOAPElement. This canthe be added to a SOAPMessage object. This method has to be used from within this class.

Parameters:

`lm` - the LocalMetadataRecord to convert to a SOAPElement

Returns:

SOAPElement the converted LocalMetadataRecord

Throws:

`SOAPException` - if something goes wrong during convertion

---

convertToXml

`public java.lang.String` **`convertToXml`**`()`

This method converst this object into a xml document (string).

Returns:

String the xml document of this object

F.8 DPC.DB.PersonalDescription

DPC.DB

Class PersonalDescription

public class **PersonalDescription**

Title: PDC WebService

Description: DPC Servlet/WebService

This class is an internal representation of a record in the metadatabase. This class has attributes for a userID, persona description and a string of subject words that describe the personal metadata in the desription attribute.

Version:

1.0

Author:

Sverre Joki

Field Detail

userID

`private java.lang.String userID`

This attribute has the value for the userID of the user that has written the personal desctiption

---

subjectWords

`private java.util.ArrayList subjectWords`

This attribute has the subject words that the given user has used to describe the personal desription

---

personalDescription

`private java.lang.String personalDescription`

This attribute holds the personal descripton or personal metadata that a user has added to a metadata record.

Constructor Detail

PersonalDescription

```
public PersonalDescription()
```

Constructs an empty PersonalDescription object

PersonalDescription

```
public PersonalDescription(java.lang.String uid,
                           java.lang.String[] subjectWordsIn,
                           java.lang.String description)
```

Constructs a PersonalDescripton object using the given parameters for userID, subject words and personal desctiption.

Parameters:

`uid` - the userID to use for the PersonalDescription object

`subjectWordsIn` - a string array containing the subject words to add to this object

`descrption` - a string containing the personal description

Method Detail

reset

```
public void reset()
```

This methos resets all attributes within this object to null.

Returns:

void

---

setUserID

`public void setUserID(java.lang.String uid)`

This method sets the user id of this object to the value specified in the parameter uid.

Parameters:

`uid` - the user id to use as userID in this object

Returns:

void

---

setDescription

`public void setDescription(java.lang.String description)`

This method sets the description of this object to the value specified in the parameter description.

Parameters:

`description` - the description to use as personalDescription in this object

Returns:

void

---

addsubjectWord

`public void addsubjectWord(java.lang.String word)`

This method adds a subject word to the list of usbject words in this object. The word to be added is supplied in a string parameter.

Parameters:

`word` - the word to add to the subjectWords in this object

Returns:

void

---

setSubjectWords

```
public void setSubjectWords(java.util.ArrayList words)
```

This method sets the subject words to use in this object. The ssubject words to be used is supplied in a ArrayList parameter.

Parameters:

`words` - the ArrayList of words to use as the subjectWords in this object

Returns:

void

---

setSubjectWords

```
public void setSubjectWords(java.lang.String[] words)
```

This method sets the subject words to use in this object. The subject words to be used is supplied in a string array parameter.

Parameters:

`words` - the string array of words to use as the subjectWords in this object

Returns:

void

---

getUserID

```
public java.lang.String getUserID()
```

This method retrievs the userID used in this object

Returns:

String the userId of this object

---

getDescription

`public java.lang.String getDescription()`

This method retrievs the personalDescription used in this object

Returns:

String the personaDescription of this object

---

getsubjectWords

`public java.util.ArrayList getsubjectWords()`

This method retrievs the subjectWords used in this object

Returns:

ArrayList the subjectwords of this object

---

convertToXmlDocument

`public java.lang.String convertToXmlDocument()`

This method converts this object to a xml document and also adds a xml document header.

Returns:

String this object converted to a well formed xml document

---

getPersonalDescriptionAsXml

`public java.lang.String getPersonalDescriptionAsXml()`

This method returns the xml elements of the attributes of this object. This inclueds PersonalDescription, UserID, and SubjectWords.

Returns:

String the xml elements of this object.

---

getPersonalDescriptionXmlStart

`public java.lang.String getPersonalDescriptionXmlStart()`

This method returns a string of the document header for this object.

Returns:

String the xm ldocument header to make the xml document well formed.

F.9 DPC.DB.UserData

DPC.DB

Class UserData

public class **UserData**

This class is a java implementation of the data stored about a user in the UserDataBase. The data about the user if contained in the attributes of this class.

Author:

Sverre Joki

Field Detail

name

`public java.lang.String name`

This object is used to store the name of this particular user.

---

userID

`public java.lang.String userID`

This object is used to store the userID of this particular user.

---

adress

`public java.lang.String **adress**`

This object is used to store the adress of this particular user.

---

postCode

`public java.lang.String **postCode**`

This object is used to store the post code of this particular user.

---

city

`public java.lang.String **city**`

This object is used to store the city of this particular user.

---

email

`public java.lang.String **email**`

This object is used to store the email-adress of this particular user.

---

position

`public java.lang.String **position**`

This object is used to store the position of this particular user.

---

institution

`public java.lang.String **institution**`

This object is used to store the instituion this particular use belongs to

---

interests

```
public java.util.ArrayList interests
```

This object is used to store the interestes of this particular user.

---

description

```
public java.lang.String description
```

This object is used to store a decrition of this particular user.

Constructor Detail

UserData

```
public UserData(java.lang.String userName,
                java.lang.String userID,
                java.lang.String adress,
                java.lang.String postCode,
                java.lang.String city,
                java.lang.String email,
                java.lang.String interest,
                java.lang.String position,
                java.lang.String instituion,
                java.lang.String description,
                java.lang.String xmlString)
```

Creates a new UserData object with the parameteres given. All parameters are given in strings, but the interests. Interest is a comma separated string containing the insterest for the particular user.

Parameters:

`userName` - the name of the user you want to create an userobject for

`userID` - the userID of the user you want to create an userobject for

`adress` - the adress of the user you want to create an userobject for

`postCode` - the post Code of the user you want to create an userobject for

`city` - the city of the user you want to create an userobject for

`email` - the email of the user you want to create an userobject for

`interest` - the interest of the user you want to create an userobject for

`position` - the position of the user you want to create an userobject for

`instituion` - the instituion of the user you want to create an userobject for

`description` - the description of the user you want to create an userobject for

`xmlString` - an xml string

Method Detail

convertToSOAP

```
public javax.xml.soap.SOAPMessage
convertToSOAP(java.util.ArrayList userDataArray)
```

This method ocnverts an ArrayList containing several user objects, to a SOAPMessage

Parameters:

`userDataArray` - an array containg the userdata object from. Each userdata object is then created into an xml-structure and inserted into a SOAPMessage.

Returns:

SOAPMessage that can be sent to the sender. This message contains all users in the userDataArray

---

convertToSOAP

```
public javax.xml.soap.SOAPMessage convertToSOAP()
```

Genereates a SOAPMessage based on the this object. This method creates one SOAPMessage based on ONE userdata object

Returns:

SOAPMessage based on the data in this object.

---

convertToSOAP

`public javax.xml.soap.SOAPMessage`
`convertToSOAP``(DPC.DB.UserData userData)`

Genereates a SOAPMessga based on the Userdata object given as input. This method creates one SOAPMessage based on ONE userdata object

Parameters:

`userData` - a UserData object to convert into a SOAPMessage

Returns:

a SOAPMessage is returned

---

createUserDataElement

`private javax.xml.soap.SOAPElement`
`createUserDataElement``(DPC.DB.UserData ud)`

This method converts a UserData object info a SOAPElement. This method can be used if you vwant to convert a lot of UserData object into a SOAPMessage. Each UserData object can then be added to a SOAPMessage as a SOAPElement

Parameters:

`ud` - the UserData object you want to convert to a SOAPElement

Returns:

SOAPElement the SOAPElement after the convertion of the UserData object.

Throws:

`SOAPException` - if something goes wrong

---

convertToXML

`public java.lang.String` **`convertToXML`**`()`

This method ocnverts this object into a xml-document in a string object. This xml document can then be sent to a database for storage. It can also be added to a SOAPMessage and sent.

Returns:

String the xml document of `this` object.

---

getUserDataXml

`public java.lang.String getUserDataXml()`

This method is used to create a xml string from the data contained in this UserData object.

Returns:

String the xml elements of this UserData object

---

getXmlElementTag

`public java.lang.String getXmlElementTag()`

This method is used to return the xml document header for this UserData object as a xml document in a string.

Returns:

String the xml document header

F.10 DPC.Interface

Package DPC.Interface

Class Summary

CollectionDBInterface
CollectionNotFoundException
DataBaseInterface
MetadataDBInterface
MetadataNotFoundException
SearchInterface
UserDBInterface
UserNotFoundException

F.11 DPC.Interface.CollectionDBInterface

DPC.Interface

Class CollectionDBInterface

public class **CollectionDBInterface**
extends DPC.Interface.DataBaseInterface

This class is the communication class that connects the SearchInterface to the Collection Database containing all personal colletion added by the users.

Author:
Sverre Joki

Field Detail

USERID

`public static java.lang.String USERID`

String constant identifiying this parameter in the collection database

RECORDID

`public static java.lang.String` **`RECORDID`**

String constant identifiying this parameter in the collection database

collection

`private java.lang.String` **`collection`**

The path to the collection in the database to use for the UserDBInterface

collectionPath

`private java.lang.String` **`collectionPath`**

The path to the collection to use for the database

Constructor Detail

CollectionDBInterface

`public` **`CollectionDBInterface`**`()`

This contructor creates a standard CollectionDBInterface object, and initializes the connetcion to the database using the attributes specified above as `driver` and `collectionPath`.

Throws:

`Exception`- If anything goes wrong during initialization, the contructor writes the exception to the error log, and throws an Exception indication a failure.

CollectionDBInterface

```
public CollectionDBInterface(java.lang.StringdbDriver,
                             java.lang.StringcollectionPath)
```

This contructor creates a standard CollectionDBInterface object, and ini-

tializes the connetcion to the database using the parameteres specified as `dbDriver` and `collectionPath`.

Parameters:

`dbDriver`- the driver class used as database connection driver
`collectionPath`- the path to the correct collection to use for the MCollection Database

Throws:

`Exception`- If anything goes wrong during initialization, the contructor writes the exception to the error log, and throws an Exception indicating a failure.

Method Detail

init

```
public void init(java.lang.Stringdriver,
                 java.lang.StringcollectionPath)
```

This method initializes the connetcion to the database using the parameteres specified as `dbDriver` and `collectionPath`

Parameters:

`dbDriver`- the driver class used as database connection driver
`collectionPath`- the path to the correct collection to use for the MetadataDatabase

Throws:

`XMLDBException`- If anything goes wrong during initialization, the contructor writes the exception to the error log, and throws an XMLDBException indicating a failure.

addCollectionRecord

```
public boolean addCollectionRecord(java.lang.StringuserID,
                                   java.lang.StringrecordID,
                                   java.lang.Stringdescription,
                                java.util.ArrayListsubjectWords)
```

This method creates a collectionrecord from the data supplied as parameters and adds it to the database. The method uses the collection as set up in the init method or contructor.

Parameters:

`userID`- the userid of the user you want to add
`recordID`- the ID of the record you want to add a description about
`description`- the description that the given user has added to the metadata-record
`subjectWords`- an arraylist of subjectwords that is used to describe the description given.

addCollectionRecord

`public boolean` **`addCollectionRecord`**`(DPC.DB.CollectionRecordrec)`

This method adds a collectionrecord to the data database. The method uses the collection database as set up in the init method or contructor.

Parameters:

`CollectionRecord`- rec the CollectionRecord object that is to be added to the database.

Returns:

boolean indicating if the database update went ok.

getCollectionUser

`public org.xmldb.api.modules.XMLResource` **`getCollection-User`**`(java.lang.Stringuserid)`

This method get the collection record from the database. The parameter to specify is the userid of the user you want to get the record for. The return object is a array of xmlresources

Parameters:

`userid`- the user id to look for

Returns:

XMLResource the collection record

getCollectionRecord

`public org.xmldb.api.modules.XMLResource` **`getCollection-Record`**`(java.lang.Stringrecid)`

Deprecated.
This method get the collection record from the database. The parameter to specify is the userid of the user you want to get the record for.

getCollection

```
private org.xmldb.api.base.ResourceSet getCollec-
tion(java.lang.Stringparam,

java.lang.Stringvalue)
```

This method is used to fetch data from the database based on a parameter and a value. This method connects to the database using the driver and path specified in the attribuets in this class.

Parameters:
`param`- the database-field to search for data in
`value`- the value to search for in the specified parameter
Returns:
ResourceSet the ResourceSet object containing the found records
Throws:
`XMLDBException`- if anything goes wrong during the connection to the database

getUserID

```
protected java.util.ArrayList getUserID(java.lang.Stringrecor-
dID)
```

This method retrieves a number of userIDs from the collection database for the record identified by the given recordID.

Parameters:
`recordID`- the record ID, for which you will get all user IDs
Returns:
ArrayLlist If records for the given recordID are found, the userIDs are added to an ArrayLlist, and returned. If the ArrayList is empty, there was no users found for the given recordID
Throws:
`Exception`- if something goes wrong when conncting to the database, the Exception is written to the errorlog, and an exception is thrown, indicating failure.

F.12 DPC.Interface.CollectionNotFoundException

DPC.Interface

Class CollectionNotFoundException

public class **CollectionNotFoundException**

extends java.lang.Exception

Title: PDC WebService

Description: DPC Servlet/WebService

This Excpetion indicates that the collection was not found in the database

Version:

1.0

Author:

Sverre Joki

Constructor Detail

CollectionNotFoundException

`public` **`CollectionNotFoundException`**`()`

Creates an CollectionNotFoundException that has no message

CollectionNotFoundException

`public` **`CollectionNotFoundException`**`(java.lang.String s)`

Creates an CollectionNotFoundException that has a message. The mesasge is given in the string parameter

Parameters:

`s` - the message string

F.13 DPC.Interface.DatabaseInterface

DPC.Interface

Class DataBaseInterface

public class **DataBaseInterface**

This class isthe superclass for all the database interfaces. This class has data about where det databases are located, and what drier the databases needs. This informastion is available in the driver and dburi attributes.

Author:

Field Detail

driver

```
protected java.lang.String driver
```

This is the attribute specifying what database driver that is needed to connect to the databases. This spiecifies if it si an Exist or XIndice database

dburi

```
protected java.lang.String dburi
```

This attribute specifies the default url to the databases

Method Detail

init

```
public void init(java.lang.Stringdriver,
                 java.lang.StringcollectionPath)
```

This method initializes the connetcion to the database using the parameteres specified as `driver` and `collectionPath`

Parameters:

`dbDriver`- the driver class used as database connection driver

`collectionPath`- the path to the correct collection to use for the MetadataDatabase

Throws:

`Exception`- If anything goes wrong during initialization, the contructor writes the exception to the error log, and throws an Exception indicating a failure.

query

```
protected org.xmldb.api.base.ResourceSet
query(java.lang.Stringxpath)
```

This method is used to send a quesry to a collection.

Parameters:

`xpath`- the xpath expression to send

Returns:

ResourceSet the resourceSet returned form the database

F.14 DPC.Interface.MetadataDBInterface

DPC.Interface

Class MetadataDBInterface

public class **MetadataDBInterface**
extends DPC.Interface.DataBaseInterface

This class is the communication class that connects the SearchInterface to the Metadatabase containing all personal metadata added by the users.

Author:

Sverre Joki

Field Detail

collection

```
private java.lang.String collection
```

The path to the collection in the database to use for the UserDBInterface

collectionPath

```
private java.lang.String collectionPath
```

The path to the collection to use for the database

Constructor Detail

MetadataDBInterface

```
public MetadataDBInterface()
```

This contructor creates a standard MetadataDBInterface object, and initializes the connetcion to the database using the attributes specified in the super-class.

Throws:

`Exception`- If anything goes wrong during initialization, the contructor writes the exception to the error log, and throws an Exception indication a failure.

MetadataDBInterface

```
public MetadataDBInterface(java.lang.StringdbDriver,
                           java.lang.StringcollectionPath)
```

This contructor creates a standard MetadataDBInterface object, and initializes the connetcion to the database using the parameteres specified as `dbDriver` and `collectionPath`

Parameters:

`dbDriver`- the driver class used as database connection driver
`collectionPath`- the path to the correct collection to use for the MCollection Database

Throws:

`Exception`- If anything goes wrong during initialization, the contructor writes the exception to the error log, and throws an Exception indicating a failure.

Method Detail

init

```
public void init(java.lang.Stringdriver,
                 java.lang.StringcollectionPath)
```

Deprecated. *use super.init instead*
This method initializes the connetcion to the database using the parameteres specified as `dbDriver` and `collectionPath`.

Parameters:

`dbDriver`- the driver class used as database connection driver
`collectionPath`- the path to the correct collection to use for the MetadataDatabase

Throws:

`XMLDBException`- If anything goes wrong during initialization, the contructor writes the exception to the error log, and throws an XMLDBException indicating a failure.

F.15 DPC.Interface.MetadataNotFoundException

DPC.Interface

Class MetadataNotFoundException

public class **MetadataNotFoundException**

extends java.lang.Exception

Title: PDC WebService

Description: DPC Servlet/WebService

This Excpetion indicates that the metadatarecord was not found in the database

Version:

1.0

Author:

Sverre Joki

Constructor Detail

MetadataNotFoundException

`public MetadataNotFoundException()`

Creates an UserNotFoundException that has no message

MetadataNotFoundException

`public MetadataNotFoundException(java.lang.String s)`

Creates an CollectionNotFoundException that has a message. The mesasge is given in the string parameter

Parameters:

`s` - the message string

F.16 DPC.Interface.SearchInterface

DPC.Interface

Class SearchInterface

public class **SearchInterface**
extends DPC.Interface.DataBaseInterface

Title: PDC WebService

Description: DPC Servlet/WebService

This class is used to perform searches in the DPC system. This class har connections to the 3 different databases in the DPC system, and builds records that the methods specify.

Version:
1.0
Author:
Sverre Joki

Field Detail

cName

`private static java.lang.String **cName**`

This is the attribute that holds this class' name. Used for debugging purposes.

theUserDBInterface

`private DPC.Interface.UserDBInterface **theUserDBInterface**`

This is the UserDbInterfae connection class

theMetadataDBInterface

`private DPC.Interface.MetadataDBInterface **theMetadataDBInterface**`

This is the object that conntect the SearchInterface to the metadata database interface

theCollectionDBInterface

`private DPC.Interface.CollectionDBInterface **theCollectionDBInterface**`

This is the object that conntect the SearchInterface to the collections database interface

Constructor Detail

SearchInterface

`public SearchInterface(java.util.Propertiesp)`

This contructor creates a SearchInterface object using de properties specified in the parameters. The properties needed are:

- collection.db.driver
- collection.db.url
- localmetadata.db.driver
- localmetadata.db.url
- users.db.driver
- users.db.url

Parameters:

`p`- proterties that has the correct values

Throws:

`Exception`- if anything goes wrong

SearchInterface

`public SearchInterface()`

This contructor creates a SearchInterface object using the default contructors of the separate interfaces

Throws:

`Exception`- if anything goes wrong

Method Detail

getMetadata

`public java.lang.String getMetadata(java.lang.Stringmdpid)`

Creates a text document object matching the MetadataPostID input.

Parameters:
`mdpid`- the metadata record id
Returns:
Document the xml document containing the record matching the given input
Throws:

getMetadataUid

`public java.lang.String **getMetadataUid**(java.lang.Stringuid)`

Creates a Document object containing all records matching the userid input.

Parameters:
`uid`- UserId to the user in question
Returns:
Document containing all records this user has registered in the metadata db
Throws:

getUsersByPosition

`public org.dom4j.Document **getUsersByPosition**(java.lang.Stringpo-sitionCategory)`

Parameters:
`PositionCategory`-
Returns:
ArrayList containing the matching user with the given positions
Throws:

getUsersInstitution

`public org.dom4j.Document **getUsersInstitution**(java.lang.String-institution)`

Parameters:
`String`- institution
Returns:
Vector containing the matchin users in the given institution
Throws:

getUsersByName

```
public org.dom4j.Document getUsersByName(java.lang.Stringuser-
Name)
```

This method returns an ArrayList containing all matching users in the UserDB. The list contains UserData objects.

Parameters:
`String`-
Returns:
ArrayList of users matching the input userName
Throws:

getUserData

```
public java.lang.String getUserData(java.lang.Stringuid)
```

This method gets the userData for a given user

Parameters:
`String`- uid the userId of the person in question
Returns:
UserData the userData object contianing the requested userInfo
Throws:

getBookshelf

```
public java.lang.String getBookshelf(java.lang.StringuserID)
```

this method retrieves the bookshelf to the user with the given userId. The returnValue is a ArrayList object that can be converted later. The arrayList contains a userata document from the database and all metadataobjects that

addUser

`public boolean **addUser**(java.lang.StringxmlRecord)`

This method add a user wrapped in an xmlrecord to the database

addPersonalMetadata

`public boolean **addPersonalMetadata**(java.lang.StringxmlRecord)`

Add personal metadata to the collections database.

Parameters:
`String`- the xml record. Must validate against the collection DTD.

convertToDocument

```
private org.dom4j.Document convertToDocu-
ment(java.util.Arrays$ArrayListlist)
```

Gives you a ArrayList containg all LocalMetadata object matching the given subjectword

Parameters:
`ArrayList`- subjectWords
Returns:
void
Throws:

F.17 DPC.Interface.UserDBInterface

DPC.Interface

Class UserDBInterface

public class **UserDBInterface**
extends DPC.Interface.DataBaseInterface

This class/object is the gateway to the userdatabase. This class is created to allow for easy switching of databases in the dpc servlet

This interface "contains" a normal databaseconnection to the database server but this can be switched eith another collection.

UserDBInterface also has methos that the DPCServlet needs to contanct the DataBase in question.

Author:
Sverre Joki

Field Detail

USER_ID

`public static java.lang.String USER_ID`

String constant identifiying this parameter in the SOAP request

INSTITUTION

`public static java.lang.String INSTITUTION`

String constant identifiying this parameter in the SOAP request

POSITION

`public static java.lang.String POSITION`

String constant identifiying this parameter in the SOAP request

NAME

`public static java.lang.String` **`NAME`**

String constant identifiying this parameter in the SOAP request

databaseType

`java.lang.String` **`databaseType`**

The database type to use. xindice or exist

collection

`private java.lang.String` **`collection`**

The path to the collection in the database to use for the UserDBInterface

collectionPath

`private java.lang.String` **`collectionPath`**

The path to the collection to use for the database

Constructor Detail

UserDBInterface

`public` **`UserDBInterface`**`()`

This constructor creates a new UserDBInterface instance, and initializes it with the attributes specified in the super class.

Throws:

`Exception`- if initializatoin goes wrong, or something else happens.

UserDBInterface

```
public UserDBInterface(java.lang.Stringdriver,
                       java.lang.StringcollectionPath)
```

This constructor creates a new UserDBInterface instance, and initializes it with the data specified in the parameters.

Parameters:

`driver`- the driver class to use when initalizing the connection to the database
`collectionPath`- the path to the collection in the database.

Throws:

`Exception`- if initializatoin goes wrong, or something else happens.

Method Detail

init

```
public void init(java.lang.Stringdriver,
                 java.lang.StringcollectionPath)
```

Deprecated. *use super.init instead*
This method initializes the connetcion to the database using the parameteres specified as `dbDriver` and `collectionPath`.

Parameters:

`dbDriver`- the driver class used as database connection driver
`collectionPath`- the path to the correct collection to use for the User DB

Throws:

`XMLDBException`- If anything goes wrong during initialization, the contructor writes the exception to the error log, and throws an XMLDBException indicating a failure.

getUsersBy

```
public java.util.ArrayList getUsersBy(java.lang.String-
searchType,
                                      java.lang.StringsearchPa-
rameter)
```

Searches the userdatabase for users with a corresponding value. If the user is found it returns a ArrayList containing all XMLResource objects matching the criteria. If the searchparameter is not found for the given searchType, it throws an exception. This is only to tell the calling object that a user was not found. It will throw an exception in any error situations.

Parameters:
`searchType`- the type of search to perform.
Returns:
ArrayList containing the user data from the search
Throws:
`Exception`- if something goes wrong during the search

getUserDataByUid

```
public org.xmldb.api.modules.XMLResource getUser-
DataByUid(java.lang.Stringuid)
```

Searches the userdatabase for users with a corresponding uid. If the user is found it returns a UserData object. If the uid is not found, it throws an exception. This is only to tell the calling object that an user was not found.It will throw an exception in any error situations.

Parameters:
`uid`- the user id to use for the search
Returns:
XMLResource the XMLResource object as returned by the database
Throws:
`UserNotFoundException`- if the user specified does not exist in the database.

getUsersByName

```
public java.util.ArrayList getUsersByName(java.lang.Stringuser-
Name)
```

Searches the userdatabase for users with a corresponding name. If the user is found it returns a ArrayList containing all UserData objects matching the criteria. If the name is not found, it throws an exception. This is only to tell the calling object that an user was not found.It will throw an excep-

tion in any error situations.

Parameters:
`userName`- the user name to search for
Returns:
ArrayList containing the userData rfam the database
Throws:
`UserNotFoundException`- if ther are no users foung that matches the given user name, this exception will be thrown

addUser

```
public boolean addUser(java.lang.StringxmlRecord)
```

This method updates or adds the UserData object given as parameter to the database

Parameters:
`xmlRecord`- the userdata as an xml strig, to save in the database.
Returns:
boolean indicating a success or failure
Throws:
`Exception`- if something goes wrong

getUser

```
private org.xmldb.api.base.ResourceSet getUser(java.lang.String-
param,

java.lang.Stringvalue)
```

Defalut get method to get Userinfo from the UserDB. The parameter specifies what type of data to search(userId, name etc.), and the value specifies the value to match

Parameters:
`param`- what type of data to search(userId, name etc.)
`value`- the value to search for in the give databasefield specified by the param
Returns:
ResourceSet a ResourceSet with the records from the database

Throws:

`XMLDBException`- if something goes wrong during the retrieval of data from the database.

F.18 DPC.Interface.UserNotFoundException

DPC.Interface

Class UserNotFoundException

public class **UserNotFoundException**

extends java.lang.Exception

Title: PDC WebService

Description: DPC Servlet/WebService

This Excpetion indicates that the user was not found in the database

Version:

1.0

Author:

Sverre Joki

Constructor Detail

UserNotFoundException

`public UserNotFoundException()`

Creates an UserNotFoundException that has no message

UserNotFoundException

`public UserNotFoundException(java.lang.String s)`

Creates an UserNotFoundException that has a message. The mesasge is given in the string parameter

Parameters:

`s` - the message string

F.19 DPC.xml

Package DPC.xml

Class Summary

CollectionXmlParser
UserDataToXmlParser
XmlToCollectionRecordParser
XmlToLocalMetadataRecordParser
XmlToUserdataParser

F.20 DPC.xml.UserDataToXmlParser

DPC.xml

Class UserDataToXmlParser

public class **UserDataToXmlParser**

Title: PDC WebService

Description: DPC Servlet/WebService
This class is an utility class to parse user data. The class has methods to convert a UserData object into a SOAPMessage object, and convert and ArrayList of UserDataobject into a SOAPMessage.

Version:

1.0

Author:

Sverre Joki

Field Detail

mf

`private javax.xml.soap.MessageFactory **mf**`

Object used to create a SOAPMessage.

---

msg

`private javax.xml.soap.SOAPMessage **msg**`

A SOAPMessage that is returned inthe methods

---

sp

`private javax.xml.soap.SOAPPart **sp**`

Part of a SOAPMessage that is used when parsing the UserData object, and creating the SOAPMessage

---

envelope

`private javax.xml.soap.SOAPEnvelope **envelope**`

Part of a SOAPMessage that is used when parsing the UserData object, and creating the SOAPMessage

---

hdr

`private javax.xml.soap.SOAPHeader **hdr**`

Part of a SOAPMessage that is used when parsing the UserData object, and creating the SOAPMessage

---

bdy

`private javax.xml.soap.SOAPBody **bdy**`

Part of a SOAPMessage that is used when parsing the UserData object, and creating the SOAPMessage

---

bodyElement

`private javax.xml.soap.SOAPBodyElement` **`bodyElement`**

Part of a SOAPMessage that is used when parsing the UserData object, and creating the SOAPMessage

---

Constructor Detail

UserDataToXmlParser

`public` **`UserDataToXmlParser`**`()`

This constructor creates an empty UserDdataToXmlParser object.

Method Detail

convertToSOAP

`public javax.xml.soap.SOAPMessage`
`convertToSOAP``(java.util.Arrays$ArrayList userDataArray)`

This method converts an group of UserData objects into a SOAPMessage. The UserData object must be given in a ArrayList object. The SOAPMessage can be sendt to a client useing the DPC system.

Parameters:

`ArrayList` - the ArrayList that contains the UserData object that will be converted into a single SOAPMessage

Returns:

SOAPMessage containing the SOAP formatted UserData

Throws:

`SOAPException` - if some part of the convertion goes wrong, a SOAPExceptoin with the message "Error converting Userdata to SOAP"

---

convertToSOAP

```
public javax.xml.soap.SOAPMessage
convertToSOAP(DPC.DB.UserData userData)
```

This method genereates a SOAPMessage based on the Userdata object given as input. The SOAPMessage contains only this user.

This method creates one SOAPMessage based on ONE userdata object

Parameters:

`userData` - the UserData object that will be converted into a single SOAPMessage

Returns:

SOAPMessage containing the SOAP formatted UserData

Throws:

`SOAPException` - if some part of the convertion goes wrong, a SOAPExceptoin with the message "Error converting Userdata to SOAP"

---

createUserDataElement

```
private javax.xml.soap.SOAPElement
createUserDataElement(DPC.DB.UserData ud)
```

This method is used to create a SOAPElement of a single UserData object. This methos is used by the other methods of this class to create user data elements to add to a SOAPMessage.

The method creates child elements like this structure:

```
UserData
- UserID
- Name
- StreetAdress
- PostCode
- City
- Email
```

```
- Position
- Institution
- Interest +
- Descpription
```

Parameters:

`ud` - the UserData object to convert into a SOAPElement

Returns:

SOAPElement the SOAPElement that is created of the UserData object

Throws:

`SOAPException` - if some of the operations on the SOAPElement goes wrong

---

F.21 DPC.xml.XmlToCollectionRecordParser

DPC.xml

Class XmlToCollectionRecordParser

public class **XmlToCollectionRecordParser**

extends org.xml.sax.helpers.DefaultHandler

Title: XmlToCollectionRecordParser

Description: XmlToCollectionRecordParser
This class is a utility class to convert a xml document into a CollectionRcord document. This CollectionRcord objcet is used to handle the collection record internally in the DPC system.

Version:

1.0

Author:

Sverre Joki

Field Detail

textBuffer

`private java.lang.StringBuffer` **`textBuffer`**

This attribute is used during the convertion of the xml structure to store the text of the parsed xml document elements.

---

cr

`private DPC.DB.CollectionRecord` **`cr`**

This attribute is used during the convertion of the xml structure to keep the CollectionRecord object.

---

recordID

`private java.lang.String` **`recordID`**

This attribute is used during the convertion of the xml structure. recordID keeps the id for the record.

---

userID

`private java.lang.String` **`userID`**

This attribute is used during the convertion of the xml structure. userId keeps the id for the user.

---

subjectWords

`private java.util.ArrayList` **`subjectWords`**

This attribute is used during the convertion of the xml structure. subjectWords keeps the list of subjectword in this collection record.

---

description

`private java.lang.String` **`description`**

This attribute is used during the convertion of the xml structure. description keeps the description in the collection record

---

isRecordID

`boolean` **`isRecordID`**

This attribute is used during the convertion of the xml structure. True of the investigated element contains a record id

---

isSubjectWord

`boolean` **`isSubjectWord`**

This attribute is used during the convertion of the xml structure True of the investigated element contains a subject word

---

isDescription

`boolean` **`isDescription`**

This attribute is used during the convertion of the xml structure True of the investigated elemen contains a description

---

isUserID

`boolean` **`isUserID`**

This attribute is used during the convertion of the xml structure True of the investigated elemen contains a user id

---

Constructor Detail

XmlToCollectionRecordParser

`public XmlToCollectionRecordParser()`

This constructor creates a empty XmlToCollectionRecordParser object. This object is used to parse a collection record from xml structure into an internal datastructure used by the DPC system.

Method Detail

parse

`public DPC.DB.CollectionRecord parse(java.lang.String xmlInput)`

This method is used to parse a collection record from xml structure into an internal ldatastructure used by the DPC system. The xml structure must be deliverd as a string object containig the xml document.

Parameters:

`xmlInput` - the String containing the xml document

Returns:

CollectionRecord the new object created from the xml document

Throws:

`SAXException` - if something goes wrong during parsing

---

startDocument

`public void startDocument()`

This method is performed when the SaxParser finds the start of the xml document. This is only done once for a collection xml document. The method instasiates a CollectionRecord object.

Returns:

void

Throws:

`SAXExcpetion` - if something unexpected happens

---

endDocument

```
public void endDocument()
```

This method is called when the SaxParser finds the end of the xml document. This method adds the values found by the parser to the CollectionRecord object.

Returns:

void

Throws:

`SAXExcpetion` - if something unexpected happens

---

startElement

```
public void startElement(java.lang.String namespaceURI,
                         java.lang.String sName,
                         java.lang.String qName,
                         org.xml.sax.Attributes attrs)
```

This method is performed when the SaxParser finds any element in the xml document. Because this method is called for very different elements, this method checks the element name, and set the values of the `isRecord`, `isDescription`, `isUserId` and `isSubjectWord` accordingly.

This method is only trigged at the start of the element. The endElement method is used to add the data in the element, when the end of the element is found. The text in the element are collected in the characters method.

Parameters:

`namespaceURI` - the namespaceURI

`sName` - the simple name of the element

`qName` - the qualified name of the element

`attrs` - the attributes to the element

Returns:

void

Throws:

`SAXExcpetion` - if something unexpected happens

---

endElement

```
public void endElement(java.lang.String namespaceURI,
                       java.lang.String sName,
                       java.lang.String qName)
```

This method is called whe the end of an element is found. Based on the values in the `isRecord`, `isDescription`, `isUserId` and `isSubjectWord`. The method collects the characters in the element, and adds theese to the correct attribute in the CollectionRecord object.

The textBuffer is reset at the end, to avoid appending new data to old.

Parameters:

`namespaceURI` - the namespaceURI

`sName` - the simple name of the element

`qName` - the qualified name of the element

`attrs` - the attributes to the element

Returns:

void

Throws:

`SAXExcpetion` - if something unexpected happens

---

characters

```
public void characters(char[] buf,
                       int offset,
                       int len)
```

Thie method is called each time the SaxParser find a charachter in the xml document. This method appends the found characters to the `textBuffer` attribute in this class. This data is available to the endElement method.

Parameters:

`buf[]` - a char array

`offset` - Where to start

`len` - How long to collect

Returns:

void

Throws:

`SAXExcpetion` - if something unexpected happens

---

reset

`private void **reset**()`

This method is used to reset the attributes of this class between elements.

F.22 DPC.xml.XmlToLocalMetadataRecordParser

DPC.xml

Class XmlToLocalMetadataRecordParser

public class **XmlToLocalMetadataRecordParser**

extends org.xml.sax.helpers.DefaultHandler

Title: PDC WebService

Description: DPC Servlet/WebService

This class is a utility class to convert a xml document into a LocalMetadataRecord object. This LocalMetadataRecord objcet is used to handle the metadta record internally in the DPC system.

Version:

1.0

Author:

Sverre Joki

Field Detail

textBuffer

`private java.lang.StringBuffer` **`textBuffer`**

This attribute is used during the convertion of the xml structure to store the text of the parsed xml document elements.

---

lm

`private DPC.DB.LocalMetadataRecord` **`lm`**

This attribute is used during the convertion of the xml structure to keep the CollectionRecord object.

---

recordID

`private java.lang.String` **`recordID`**

This attribute is used during the convertion of the xml structure. recordID keeps the id for the record.

---

metadataURI

`private java.lang.String` **`metadataURI`**

This attribute is used during the convertion of the xml structure. metadataURI keeps the uri for the metadata record.

---

isMetadataRecord

`boolean` **`isMetadataRecord`**

This attribute is used during the convertion of the xml structure. True of the investigated element contains a metadata record

---

isRecordID

`boolean` **`isRecordID`**

This attribute is used during the convertion of the xml structure. True of the investigated element contains a record id

---

isMetadataURI

`boolean` **`isMetadataURI`**

This attribute is used during the convertion of the xml structure. True of the investigated element contains a metadata uri

---

Constructor Detail

XmlToLocalMetadataRecordParser

`public` **`XmlToLocalMetadataRecordParser`**`()`

This constructor creates a empty XmlToLocalMetadataRecordParser object. This object is used to parse a local metadata record from a xml structure into an internal ldatastructure used by the DPC system.

Method Detail

parse

`public DPC.DB.LocalMetadataRecord`
`parse``(java.lang.String xmlInput)`

This method is used to parse a local metadata record from a xml structure into an internal ldatastructure used by the DPC system. The xml structure must be deliverd as a string object containig the xml document.

Parameters:

`xmlInput` - the String containing the xml document

Returns:

LocalMetadataRecord the new object created from the xml document

Throws:

`SAXException` - if something goes wrong during parsing

---

startDocument

`public void startDocument()`

This method is performed when the SaxParser finds the start of the xml document. This is only done once for a collection xml document. The method instasiates a LocalMetadataRecord object.

Returns:

void

Throws:

`SAXExcpetion` - if something unexpected happens

---

endDocument

`public void endDocument()`

This method is called when the SaxParser finds the end of the xml document. This method adds the values found by the parser to the LocalMetadataRedcord object.

Returns:

void

Throws:

`SAXExcpetion` - if something unexpected happens

---

startElement

```
public void startElement(java.lang.String namespaceURI,
                         java.lang.String sName,
                         java.lang.String qName,
                         org.xml.sax.Attributes attrs)
```

This method is performed when the SaxParser finds any element in the xml document. Because this method is called for very different elements, this method checks the element name, and set the values of the `isRecord` and `isMetadataURI` accordingly.

This method is only trigged at the start of the element. The endElement method is used to add the data in the element, when the end of the element is found. The text in the element are collected in the characters method.

Parameters:

`namespaceURI` - the namespaceURI

`sName` - the simple name of the element

`qName` - the qualified name of the element

`attrs` - the attributes to the element

Returns:

void

Throws:

`SAXExcpetion` - if something unexpected happens

---

endElement

```
public void endElement(java.lang.String namespaceURI,
                       java.lang.String sName,
                       java.lang.String qName)
```

This method is called whe the end of an element is found. Based on the values in `isRecord` and `isMetadataURI`. The method collects the characters in the element, and adds theese to the correct attribute in the CollectionRecord object.

The textBuffer is reset at the end, to avoid appending new data to old.

Parameters:

`namespaceURI` - the namespaceURI

`sName` - the simple name of the element

`qName` - the qualified name of the element

`attrs` - the attributes to the element

Returns:

void

Throws:

`SAXExcpetion` - if something unexpected happens

---

characters

```
public void characters(char[] buf,
                       int offset,
                       int len)
```

Thie method is called each time the SaxParser find a character in the xml document. This method appends the found characters to the `text-Buffer` attribute in this class. This data is available to the endElement method.

Parameters:

`buf[]` - a char array

`offset` - Where to start

`len` - How long to collect

Returns:

void

Throws:

`SAXExcpetion` - if something unexpected happens

reset

`private void reset()`

This method is used to reset the attributes of this class between elements.

F.23 DPC.xml.XmlToUserdataParser

DPC.xml

Class XmlToUserdataParser

public class **XmlToUserdataParser**

extends org.xml.sax.helpers.DefaultHandler

Title: XmlToUserdataParser

Description: XmlToUserdataParser

This class is a utility class to convert a xml document into a UserData object This UserData objcet is used to handle the user data internally in the DPC system.

Version:

1.0

Author:

Sverre Joki

Field Detail

textBuffer

`private java.lang.StringBuffer textBuffer`

This attribute is used during the convertion of the xml structure to store the text of the parsed xml document elements.

---

userdata

`private DPC.DB.UserData userdata`

This attribute is used during the convertion of the xml structure to keep the UserData object.

---

userID

```
private java.lang.String userID
```

This attribute is used during the convertion of the xml structure. userID keeps the id for the user.

---

name

```
private java.lang.String name
```

This attribute is used during the convertion of the xml structure. name keeps the user name in the UserData object

---

streetAdress

```
private java.lang.String streetAdress
```

This attribute is used during the convertion of the xml structure. stree-tAdress keeps the street adress in the UserData object

---

city

```
private java.lang.String city
```

This attribute is used during the convertion of the xml structure. city keeps the user's city in the UserData object

---

postCode

```
private java.lang.String postCode
```

This attribute is used during the convertion of the xml structure. post-Code keeps the post code in the UserData object

---

email

`private java.lang.String **email**`

This attribute is used during the convertion of the xml structure. email keeps the user's email adress in the UserData object

---

position

`private java.lang.String **position**`

This attribute is used during the convertion of the xml structure. position keeps the user's work-position in the UserData object

---

institution

`private java.lang.String **institution**`

This attribute is used during the convertion of the xml structure. institution keeps the user's institution in the UserData object

---

interests

`private java.util.ArrayList **interests**`

This attribute is used during the convertion of the xml structure. interests keeps the user's interests in the UserData object

---

description

`private java.lang.String **description**`

This attribute is used during the convertion of the xml structure. description keeps the description in the collection record

---

isUserID

`boolean **isUserID**`

This attribute is used during the convertion of the xml structure. True of the investigated element contains a user id

---

isName

`boolean` **`isName`**

This attribute is used during the convertion of the xml structure. True of the investigated element contains a name

---

isStreetAdress

`boolean` **`isStreetAdress`**

This attribute is used during the convertion of the xml structure. True of the investigated element contains a Street adress

---

isCity

`boolean` **`isCity`**

This attribute is used during the convertion of the xml structure. True of the investigated element contains a city

---

isPostCode

`boolean` **`isPostCode`**

This attribute is used during the convertion of the xml structure. True of the investigated element contains a post code

---

isEmail

`boolean` **`isEmail`**

This attribute is used during the convertion of the xml structure. True of the investigated element contains a email adress

---

isPosition

`boolean` **`isPosition`**

This attribute is used during the convertion of the xml structure. True of the investigated element contains a postition

---

isInstitution

`boolean` **`isInstitution`**

This attribute is used during the convertion of the xml structure. True of the investigated element contains a institution

---

isInterest

`boolean` **`isInterest`**

This attribute is used during the convertion of the xml structure. True of the investigated element contains a interst

---

isDescription

`boolean` **`isDescription`**

This attribute is used during the convertion of the xml structure. True of the investigated element contains a description

---

Constructor Detail

XmlToUserdataParser

`public` **`XmlToUserdataParser`**`()`

This constructor creates a empty XmlToUserdataParser object. This object is used to parse a user record from xml structure into an internal datastructure used by the DPC system.

Method Detail

parse

```
public DPC.DB.UserData parse(java.lang.String xmlInput)
```

This method is used to parse a user data record from a xml structure into an internal datastructure used by the DPC system. The xml structure must be deliverd as a string object containig the xml document.

Parameters:

`xmlInput` - the String containing the xml document

Returns:

UserData the new object created from the xml document

Throws:

`SAXException` - if something goes wrong during parsing

---

startDocument

```
public void startDocument()
```

This method is performed when the SaxParser finds the start of the xml document. This is only done once for a collection xml document.

Returns:

void

Throws:

`SAXExcpetion` - if something unexpected happens

---

endDocument

```
public void endDocument()
```

This method is called when the SaxParser finds the end of the xml document. This method adds the values found by the parser to the UserData object.

Returns:

void

Throws:

`SAXExcpetion` - if something unexpected happens

---

startElement

```
public void startElement(java.lang.String namespaceURI,
                         java.lang.String sName,
                         java.lang.String qName,
                         org.xml.sax.Attributes attrs)
```

This method is performed when the SaxParser finds any element in the xml document. Because this method is called for very different elements, this method checks the element name, and set the values of the attributes indicating which element is found accordingly.

This method is only trigged at the start of the element. The endElement method is used to add the data in the element, when the end of the element is found. The text in the element are collected in the characters method.

Parameters:

`namespaceURI` - the namespaceURI

`sName` - the simple name of the element

`qName` - the qualified name of the element

`attrs` - the attributes to the element

Returns:

void

Throws:

`SAXExcpetion` - if something unexpected happens

---

endElement

```
public void endElement(java.lang.String namespaceURI,
```

```
                         java.lang.String sName,
                         java.lang.String qName)
```

This method is called whe the end of an element is found. Based on the values in

The textBuffer is reset at the end, to avoid appending new data to old.

Parameters:

`namespaceURI` - the namespaceURI

`sName` - the simple name of the element

`qName` - the qualified name of the element

`attrs` - the attributes to the element

Returns:

void

Throws:

`SAXExcpetion` - if something unexpected happens

---

characters

```
public void characters(char[] buf,
                       int offset,
                       int len)
```

Thie method is called each time the SaxParser find a charachter in the xml document. This method appends the found characters to the `text-Buffer` attribute in this class. This data is available to the endElement method.

Parameters:

`buf[]` - a char array

`offset` - Where to start

`len` - How long to collect

Returns:

void

Throws:

`SAXExcpetion` - if something unexpected happens

---

reset

```
private void reset()
```

This method is used to reset the attributes of this class between elements.

APPENDIX G PROGRAMKODE

G.1 DPC.DPCServlet

```
package DPC;

import DPC.*;
import DPC.DB.*;
import DPC.Interface.*;
import DPC.xml.*;

import java.util.*;
import java.io.*;
import javax.activation.*;

import org.w3c.dom.*;

//soap stuff
import javax.xml.soap.*;
import javax.xml.messaging.*;

//servletstuff
import javax.servlet.*;
import javax.servlet.http.*;

import javax.xml.transform.*;


/**
 * This is the main server part of the DPC system. The
 * DPCServlet handles all resuests to the DPC system.
 * The system is activaed by a call to the onMessage method.
 * This method handles the recieveing and sending of messages.
 * @author Sverre Joki
 */
public class DPCServlet
    extends JAXMServlet
    implements ReqRespListener {

  /***
   * String constant identifying this command
   */
  public static String GET_BOOKSHELF =
      "GetBookshelf";

  /***
   * String constant identifying this command
   */
  public static String GET_USER_BY_ID =
      "GetUserByUid";
```

```
/***
 * String constant identifying this command
 */
public static String GET_USERS_BY_NAME =
    "GetUsersByName";

/***
 * String constant identifying this command
 */
public static String GET_USERS_BY_INSTITUTION =
    "GetUsersByInstitution";

/***
 * String constant identifying this command
 */
public static String GET_USERS_BY_POSITION =
    "GetUsersByPosition";

/***
 * String constant identifying this command
 */
public static String GET_METADATA_BY_ID =
    "GetMetadataById";

/***
 * String constant identifying this command
 */
public static String GET_METADATA_BY_SUBJECT =
    "GetMetadataBySubject";

/***
 * String constant identifying this command
 */
public static String SET_USER = "SetUser";

/***
 * String constant identifying this command
 */
public static String SET_METADATA =
    "SetMetadata";

/***
 * This attribute is the SearchInterface object that
 * connects the DPCServlet to the databases.
 */
private SearchInterface searchInterface = null;

/***
 * stringbuffer to keep reply text
 */
private StringBuffer replyText = new
    StringBuffer();

/***
 * True if the servlet is initialized
```

```
   */
  private boolean inited = false;

  /***
   * The object responsible for creating new empty messages
   */
  static MessageFactory fac = null;

  static {
    try {
      fac = MessageFactory.newInstance();
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
  };

  /***
   * Kalenderobjeckt som brukes i debugsammenheng
   */
  GregorianCalendar cal = new GregorianCalendar();

  /***
   * This method initializes the serlvet, and sets up the
   * searchInterface class to handle requests from this servlet.
   * @author Sverre Joki
   * @param ServletConfig
   * @throws ServletExpetion if anything goes very wrong
   */
public void init(ServletConfig
                      servletConfig) throws
      ServletException {
    super.init(servletConfig);
    // Klargjør resten av basene man kan søke i etc.
    try {
      //først henter vi parametre fra contexten
      System.err.println(
          "\n\n\nSTARING SERVLET\n"
          + cal.getTime().toString()
          + ": Servlet init");
      ServletContext context =
          servletConfig.
          getServletContext();
      System.err.println(
          "Ferdig å sette opp searchinterface");
      Properties props = new Properties();
      int i = 0;
      for (Enumeration e = context.
           getInitParameterNames();
           e.hasMoreElements(); ) {
        String name, value;
        name = (String) e.nextElement();
        System.err.println("name=" + name);
        value = context.getInitParameter(
            name);
        System.err.println("value="
```

```
                          + value);
          props.setProperty(name, value);
          i++;
          System.err.println("Etter " + i
                          +
                          ". gjennomgang i lista over parametre");
        }

        context.setAttribute("conf", props);

        System.err.println(
            "Forsøker å lage et searchInterafcea objekt");
        SearchInterface si = new
            SearchInterface(props);

        inited = true;
        System.err.println(cal.getTime().
                        toString()
                        + ": Servlet init OK!");
      }
      catch (Exception e) {
        System.err.println(cal.getTime().
                        toString()
                        +
                        ": Servlet init NOT OK!");
        inited = false;
        e.printStackTrace();
        searchInterface = null;
      }
    }

    /***
     * This is the application code for handling the message.
     * Once the message is received the application can
     * retrieve the soap part, the attachment part if there
     * are any, or any other information from the message.
     * This method converts a SOAP message to an internal
     * format that can be handled by the DPC server.
     * @author Sverre Joki
     * @param SOAPMessage the SOAP message to process
     * @return SOAPMessage the returned SOAP message with the
     *                     results from the processing
     */
    public SOAPMessage onMessage(SOAPMessage
        message) {
      System.err.print(
          "\n\nRecieving message\n\n"
          + cal.getTime().toString()
          + ": DPCServlet.onMessage");
      System.err.println(
          "Dette er soapmeldingen som kom inn:");
      try {
        ByteArrayOutputStream bs = new
            ByteArrayOutputStream();
        message.writeTo(bs);
        System.err.println(bs.toString()
```

```
                        + "\n");
    }
    catch (Exception e) {
      //do nothing
    }
    if (inited) {
      try {
        System.err.print(
            "Getting the DPCSoapMessage object");
        DPCSoapMessage dpcm = new
            DPCSoapMessage(message);
        replyText.append(dpcm.initLog);
        return generateCall(dpcm);
      }
      catch (Exception e) {
        System.err.println(
            "onMessage caught an exception:");
        e.printStackTrace(System.err);
        replyText.append(
            "Error in getting content from soap message");
        return generateReturnMessage();
      }
    }
    else {
      return createErrorMessage(
          "An error occured during initalization.");
    }
  }

  /**
   * Based on generateCall and subsequent calls, this method
   * generates a SOAPMessage to return to the caller
   * @return SoapMessage the soapMessage to be returned,
   *                      null if errors occur
   * @author Sverre Joki
   */
  private SOAPMessage generateReturnMessage() {
    try {
      SOAPMessage msg = fac.createMessage();
      SOAPEnvelope envelope = msg.
          getSOAPPart().
          getEnvelope();
      SOAPHeader sh = envelope.getHeader();
      SOAPBody soapBody = envelope.getBody();
      soapBody.addChildElement(envelope.
          createName(
          "DPC_Return_Message")).
          addTextNode(
          "Dette er en response heppeti1");
      try {
        soapBody.addChildElement(envelope.
            createName(
            "Debug")).addTextNode(
            replyText.toString());
      }
      catch (Exception e) {
```

```
        System.err.println(
            "Dette gikk ikke");
      }
      return msg;
    }
    catch (Exception e) {
      replyText.append(
          "Error in processing or replying to a message"
          + e.toString());
      return null;
    }
  }

  /**
   * This method generate a call to the corrert method for subsequent
   * handling of the request based on the DPCSoapMessage object.
   * @param dpcSoapMessage the DPCSoapMessage object generated in
   *                       the omMessage method
   * @return SOAPMessge the message to be returned to the user
   * @author Sverre Joki
   */
  private SOAPMessage generateCall(
      DPCSoapMessage msg) {
    System.err.print(
        "DPCServlet.generateCall ["
        + cal.getTime().toString() + "]");
    String command = msg.dpcCommand;
    SOAPMessage returnMessage;

    if (command.equalsIgnoreCase(this.
        GET_USER_BY_ID)) {
      System.err.println(
          "get user by userID");
      System.err.println(
          "parameters are in an space separated string");
      StringTokenizer st = new
          StringTokenizer(msg.
                          dpcParameters, " ");
      String parameter = new String();
      System.err.println("if there is more than one parameter in the
message, get the first one");
      System.err.println(
          "and dump the rest");
      parameter = st.nextToken();
      System.err.println(
          "calling getUserDataMessage. the first parameter is:"
          + parameter);
      returnMessage = getUserDataMessage(
          parameter);

    }

    else if (command.equalsIgnoreCase(this.
        GET_BOOKSHELF)) {
      System.err.println("get bookself");
      System.err.println(
```

```
          "parameters are in an space separated string");
      StringTokenizer st = new
          StringTokenizer(msg.
                          dpcParameters, " ");
      String parameter = new String();
      System.err.println("if there is more than one parameter in the
message, get the first one");
      System.err.println(
          "and dump the rest");
      parameter = st.nextToken();
      System.err.println(
          "calling getBookshelf. the first parameter is:"
          + parameter);
      returnMessage = getBookShelf(
          parameter);

    }
    else if (command.equalsIgnoreCase(this.
        GET_METADATA_BY_ID)) {
      System.err.println(
          "get metadata by mdpid");
      returnMessage = getMetadataMessage(
          msg.dpcParameters);

    }
    else if (command.equals(this.SET_USER)) {
      System.err.println("Set data methods");
      returnMessage = addUserData(msg);
    }
    else if (command.equals(this.SET_METADATA)) {
      System.err.println("Set data methods");
      returnMessage = addMetadata(msg);
    }
    else {
      returnMessage = createErrorMessage(
          "Invalid search");
    }
    if (returnMessage != null) {
      return returnMessage;
    }
    else {
      return createErrorMessage(
          "an error occured");
    }
  }

  /**********************************
   * USER METHODS Methods that gets / sets user data
   **********************************/
  /**
   * Generates a soapmessage containing the userdata for the given userid.
   * Usually uses getUserData(String uid) to fetch the userdata.
   * @param uid the userId of the person in question
   * @return SOAPMessage the soapmessage contianing the requested userInfo
   */
  public SOAPMessage getUserDataMessage(String
```

```
      uid) {
    System.err.println(cal.getTime().toString()
                       +
                       ": getUserDataMessage. input uid: "
                       + uid);
    SOAPMessage msg;
    try {
      System.err.println(
          "get user by userID, calling searchInterface");
      String doc = null;
      if (searchInterface != null) {
        doc = searchInterface.getUserData(
            uid);
      }
      else {
        ServletContext ctx =
            getServletContext();
        Properties props = (Properties)
            ctx.getAttribute(
            "conf");
        searchInterface = new
            SearchInterface(props);
        doc = searchInterface.getUserData(
            uid);
      }

      if (doc != null) {
        System.err.println(
            "Dette er documentet før vi legger det til i meldinga:"
            + doc);
        //creating the soap message
        msg = createReturnMessage(doc);
        return msg;
      }
      else {
        return createErrorMessage(
            "no record found");
      }

    }
    catch (Exception e) {
      e.printStackTrace(System.err);
      return createErrorMessage(
          "An error occured:"
          + e.toString());
    }
  }

  /**
   * Det som står nedenfor er tatt ut av prioriteringsgrunner
   *
   */

  /**
   * This method returns a message containg users with the given name
   * @param String the user name you want to be found
```

```
 * @author Sverre Joki
 */
public SOAPMessage getUsersMessage(String
    userName) {
  SOAPMessage msg = null;
  UserData parser = new UserData();
  try {
    ArrayList list = searchInterface.
        getUser(userName);
    if (list != null && list.size() > 0) {
      msg = parser.convertToSOAP(list);
    }
    else {
      //create error SOAP Message
      msg = createErrorMessage(
          "No users found");
    }
  }
  catch (Exception e) {
    msg = createErrorMessage(
        "No users found");
  }
  return msg;
}

/**
 * This method finds all useres which belongs to a give institution.
 * @param String the institution in question
 * @return SOAPMessage containing the matchin users in the given
 *                      institution
 * @author Sverre Joki
 */
public SOAPMessage getUsersInstitutionMessage(
    String
    institution) {
  SOAPMessage msg = null;
  UserData parser = new UserData();
  try {
    ArrayList list = searchInterface.
        getUsersInstitution(
        institution);
    if (list != null && list.size() > 0) {
      msg = parser.convertToSOAP(list);
    }
    else {
      //create error SOAP Message
      msg = createErrorMessage(
          "No users found");
    }
  }
  catch (Exception e) {
    msg = createErrorMessage(
        "No users found");
  }
  return msg;
}
```

```
/**
 * This method finds all usres with a given position within any
 * instituion.
 * @param PositionCategory The position categoy in question
 * @return ArrayList containing the matching users with the given
 *                   positions
 * @author Sverre Joki
 */
public SOAPMessage getUsersMessage(
    PositionCategory
    positionCategory) {
  SOAPMessage msg = null;
  UserData parser = new UserData();
  try {
    ArrayList list = searchInterface.
        getUsersByPosition(
        positionCategory);
    if (list != null && list.size() > 0) {
      msg = parser.convertToSOAP(list);
    }
    else {
      //create error SOAP Message
      msg = createErrorMessage(
          "No users found");
    }
  }
  catch (Exception e) {
    msg = createErrorMessage(
        "No users found");
  }
  return msg;
}

/**
 * This method adds an usre to the database. It then generates a
 * return message telling whoever want to read it, that all went well.
 * If the update doesn't go well, the returnmessage tells you that.
 * @param DPCMessage the dpc message containing
 * @return SOAPMessage
 * @author Sverre Joki
 */
public SOAPMessage addUserData(DPCSoapMessage
    dpcMsg) {
  SOAPMessage msg = null;
  //Get the attached xml stings from the dpcMsg and create UserData
  //object of them.
  Iterator it = dpcMsg.dpcAttchments;
  while (it.hasNext()) {
    String xmlUserRecord = (String) it.
        next();
    //Then add them to the database
    try {
      boolean ok = searchInterface.
          addUser(
          xmlUserRecord);
```

```
        if (ok) {
          msg = createErrorMessage(
              "1: ok");
        }
        else {
          msg = createErrorMessage(
              "0: update failed");
        }
      }
      catch (Exception e) {
        msg = createErrorMessage(
            "0: update failed");
      }
    }
    return msg;
  }

  /*******************************
   * METHODS FOR SETTING/GETTING
   * METADATA
   *******************************/

  /**
   * Creates a SOAPMessage containing the LocalMetadataREcord matching the
   * MetadataPostID input.
   * @param String The id of the metadatarecord in question
   * @return SOAPMessage a message containing the result from the search
   * @author Sverre Joki
   */
  public SOAPMessage getMetadataMessage(String
      mdpid) {
    System.err.println(this.getClass().
                       getName()
                       +
                       "getMetadataMessage():["
                       + cal.getTime().
                       toString()
                       + "]:START");
    SOAPMessage msg = null;
    try {
      System.err.println(this.getClass().
                         getName()
                         +
                         "getMetadataMessage():["
                         + cal.getTime().toString()
                         +
                         "]:trying to get message");
      String doc = null;
      if (searchInterface != null) {
        doc = searchInterface.getMetadata(
            mdpid);
      }
      else {
        ServletContext ctx =
            getServletContext();
        Properties props = (Properties)
```

```
            ctx.getAttribute(
            "conf");
        searchInterface = new
            SearchInterface(props);
        doc = searchInterface.getMetadata(
            mdpid);
      }
      System.err.println(this.getClass().
                         getName()
                         +
                         "getMetadataMessage():["
                         + cal.getTime().toString()
                         + "]:got message");

      msg = createReturnMessage(doc);

      System.err.println(this.getClass().
                         getName()
                         +
                         "getMetadataMessage():["
                         + cal.getTime().toString()
                         + "]:got return-message");

    }
    catch (Exception e) {
      msg = createErrorMessage(
          "No records found");
    }
    System.err.println(this.getClass().
                       getName()
                       +
                       "getMetadataMessage():["
                       + cal.getTime().
                       toString() + "]:END");
    return msg;
  }

  /**
   * Creates a SOAPMessage containing the LocalMetadataREcord matching the
   * UserID input.
   * @param userID the userId you want to search for in the metadatabase.
   * @return SOAPMessage having the metadata record for the given userID
   */
  public SOAPMessage getMetadataByUser(String
      userid) {
    SOAPMessage msg = null;
    Document doc;
    try {
      String s = searchInterface.
          getMetadataUid(userid);
      msg = createReturnMessage(s);
    }
    catch (Exception e) {
      msg = createErrorMessage(
          "No records found");
    }
```

```
  return msg;
}

/**
 * Generates a SOAPMessage containing records whith corresponding
 * subjectwords.
 * @param String the subject words in question
 * @return SOAPMessage a message containing the result from the search
 * @author Sverre Joki
 */
public SOAPMessage getMetadataMessage(
    ArrayList subjectWords) {
  SOAPMessage msg = null;
  try {
    ArrayList list = searchInterface.
        getMetadataBySubject(
        subjectWords);
    LocalMetadataRecord rec = new
        LocalMetadataRecord();
    msg = rec.convertToSOAP(list);
  }
  catch (Exception e) {
    msg = createErrorMessage(
        "No records found");
  }
  return msg;
}

/**
 * This method adds a record to the metadata-database in the DPC system.
 * @param dpcMsg the DPCSoapMessage containing the metadatarecord
 *               you want to add to the database
 * @return SOAPMessage indicating the success of the request
 * @author Sverre Joki
 */
public SOAPMessage addMetadata(DPCSoapMessage
    dpcMsg) {
  SOAPMessage msg = null;
  Iterator it = dpcMsg.dpcAttchments;
  while (it.hasNext()) {
    String xmlRecord = (String) it.next();
    try {
      boolean ok = searchInterface.
          addMetadata(
          xmlRecord);
      if (ok) {
        msg = createErrorMessage(
            "1: ok");
      }
      else {
        msg = createErrorMessage(
            "0: updete failed");
      }
    }
    catch (Exception e) {
      msg = createErrorMessage(
```

```
            "0: updete failed");
      }
    }
    return msg;
  }

  /******************************
   * METHODS FOR SETTING/GETTING
   * BOOKSHELVES
   ******************************/

  /**
   * Gets the bookshelf of the user indicated by the user id.
   * - get the userdata from udb
   * - get the metadata from the
   * @param UserID
   * @return void
   */
  public SOAPMessage getBookShelf(String uid) {

    System.err.println(this.getClass().
                       getName()
                       + "getBookshelf():["
                       + cal.getTime().
                       toString()
                       + "]:START");
    SOAPMessage msg = null;
    try {
      System.err.println(this.getClass().
                         getName()
                         + "getBookshelf():["
                         + cal.getTime().toString()
                         +
                         "]:trying to get message");
      String doc = null;
      if (searchInterface != null) {
        System.err.println(
            "searchinterface!=null");
        doc = searchInterface.
            getBookshelf(uid);
      }
      else {
        System.err.println(
            "searchinterface=null");
        ServletContext ctx =
            getServletContext();
        Properties props = (Properties)
            ctx.getAttribute(
            "conf");
        searchInterface = new
            SearchInterface(props);
        doc = searchInterface.
            getBookshelf(uid);
      }
      System.err.println(this.getClass().
                         getName()
```

```
                          + "getBookshelf():["
                          + cal.getTime().toString()
                          + "]:got message:\n"
                          + doc
                          + "\n end of message");

        msg = createReturnMessage(doc);

        System.err.println(this.getClass().
                          getName()
                          + "getBookshelf():["
                          + cal.getTime().toString()
                          + "]:got return-message");

      }
      catch (Exception e) {
        msg = createErrorMessage(
            "No records found");
      }
      System.err.println(this.getClass().
                        getName()
                        + "getBookshelf():["
                        + cal.getTime().
                        toString() + "]:END");
      return msg;
    }

    /**
     * THIS METHOD sends an bookshelf by to the user that is specified by
     * the userID. Not implemented.
     *
     * @param UserID
     * @return void
     * @exception
     * @roseuid 3D5E69F101EA
     */
    public void mailBookShelf(UserID uid) {}

    //-------------------------------------
    // Utility methods
    //--------------------------------------

    /***
     * This method creates a simple SOAPmessage containing some text.
     * The metod takes as argument a string which represents the message
     * that is to be returned to the SOAP message sender.
     * @param String the message text to be included in the message
     * @return SOAPMessage the soap message that is generated by this method
     * @author Sverre Joki
     */
    private SOAPMessage createReturnMessage(
        String message) {
      try {
        SOAPMessage msg = fac.createMessage();
        SOAPEnvelope envelope = msg.
            getSOAPPart().
```

```
          getEnvelope();
      SOAPHeader sh = envelope.getHeader();
      SOAPBody soapBody = envelope.getBody();
      soapBody.addChildElement(envelope.
          createName(
          "DPC_Return_Message")).
          addChildElement(
          "ReturnMessage").addTextNode(
          message);
      return msg;
    }
    catch (Exception e) {
      replyText.append(
          "Error in processing or replying to a message"
          + e.toString());
      return null;
    }

  }

  /***
   * This method creates a simple SOAPmessage indicating errors. The
   * metod takes as argument a string which represents the errormessage
   * that is to be returned to the SOAP message sender.
   * @param String the message text to be included in the errormessage
   * @return SOAPMessage the soap message that is generated by this method
   * @author Sverre Joki
   */
  private SOAPMessage createErrorMessage(String
      message) {
    try {
      SOAPMessage msg = fac.createMessage();
      SOAPEnvelope envelope = msg.
          getSOAPPart().
          getEnvelope();
      SOAPHeader sh = envelope.getHeader();
      SOAPBody soapBody = envelope.getBody();
      soapBody.addChildElement(envelope.
          createName(
          "DPC_Return_Message")).
          addChildElement(
          "ErrorMessage").addTextNode(
          message);
      return msg;
    }
    catch (Exception e) {
      replyText.append(
          "Error in processing or replying to a message"
          + e.toString());
      return null;
    }
  }
}
```

G.2 DPC.DPCSOAPMessage

```
package DPC;

import DPC.DB.*;
import DPC.Interface.*;
import DPC.xml.*;
import DPC.*;
import javax.xml.soap.*;
import javax.xml.messaging.*;
import javax.xml.transform.*;
import java.util.*;

/**
 * <p>Title: PDC WebService</p>
 * <p>Description: DPC Servlet/WebService</p>
 * <p>
 * This is an class for the representation of an internal DPC Message.
 * This message had information about what command and parameters the
 * client sent, and has a structure to contain the attachements sent
 * in a SOAPMessage.
 * </p>
 * @author Sverre Joki
 * @version 1.0
 */
public class DPCSoapMessage {

  /***
   * The command in ths SOAPMessage
   */
  public String dpcCommand;
  /***
   * The parameters in the SOAP message
   */
  public String dpcParameters;

  /***
   * This attribute is used to logging.
   */
  public StringBuffer initLog = new StringBuffer();

  /***
   * This attribute is used to iterate over all attachments in the
   */

  public Iterator dpcAttchments;

  /***
   * Constructs an empty DPCSoapMessage
   */
  public DPCSoapMessage() {}

  /***
   * Constructs a DPCSoapMessage with the data supplied in the parameters.
   * This means that the object will be initialized with a command, a
```

```
   * parameter and attachments.
   * @param String command the command that the SOAPMessage wanted
   * @param String parameter the parameters to the given command
   * @param ArrayList attachments the list of attachements that came
   *                  along the SOAP message
   */
  public DPCSoapMessage(String command,
                        String parameter,
                        ArrayList attachments) {
    dpcCommand = command;
  }

  /***
   * This contructor uses an incoming SOAPMessage to create a
   * DPCSoapMessage object. The SOAPMessage will be parsed to identify
   * the command, parameters and attachments, and the attributes of the
   * object will be set accordingly.
   * @param SOAPMessage the SOAPMessage you want to create a
DPCSoapMessage from
   */
  public DPCSoapMessage(SOAPMessage msg) {
    if (msg != null)
      init(msg);
    else
      initLog.append("Error in SOAPMessage");
  }

  /***
   * This method uses an incoming SOAPMessage to initialize this
   * DPCSoapMessage object. The SOAPMessage will be parsed to identify
   * the command, parameters and attachments, and the attributes of the
   * object will be set accordingly.
   * @param SOAPMessage the SOAPMessage you want to create a
DPCSoapMessage from
   * @return void
   */
  public void init(SOAPMessage msg) {
    try {
      SOAPPart sp = msg.getSOAPPart();
      SOAPEnvelope env = sp.getEnvelope();
      SOAPBody sb = env.getBody();
      initLog.append(
          "\nElement local name og first element:").
          append(sb.
                 getElementName().getLocalName());
      Iterator it = sb.getChildElements();
      try {
        initLog.append(
            "\nPassed first Iterator creation\n");
        int i = 0;
        while (it.hasNext()) {
          i++;
          SOAPElement bodyElement = (SOAPElement)
              it.next();
          initLog.append(i + ". "
                         + bodyElement.
```

```
                        getElementName().
                        getLocalName());
         try {
           Iterator it2 = bodyElement.
               getChildElements();
           SOAPElement element = (SOAPElement)
               it2.next();
           initLog.append("\ncommand"
                          + element.
                          getElementName().
                          getLocalName());
           dpcCommand = element.getElementName().
               getLocalName();
           initLog.append("\nparameter"
                          + element.getValue());
           dpcParameters = element.getValue();
         }
         catch (Exception e) {
           initLog.append(
               "\niterator gikk ikke2");
         }
       }
       dpcAttchments = msg.getAttachments();
     }
     catch (Exception e) {
       initLog.append(
           "\nIterator 1 .hasNext gikk ikke");
     }
   }
   catch (Exception e) {
     initLog.append(
         "\nDPCSOAPMessage encoutered an error:"
         + e.toString());
   }

 }

 /***
  * This method fetches the command from a SOAPMessage.
  * @param SOAPMessage the SOAPMessage you want to fetch the command from
  * @return String the string value of the command
  */
 public String getCommand(SOAPMessage msg) {
   initLog.append(
       "\n\nDPCSOAPMessage.getCommand() kjører nå...\n");
   try {
     SOAPPart sp = msg.getSOAPPart();
     SOAPEnvelope env = sp.getEnvelope();
     SOAPBody sb = env.getBody();
     initLog.append("element local name:").
         append(sb.getElementName().
                getLocalName());
     Iterator it = sb.getChildElements();
     SOAPBodyElement bodyElement = (
         SOAPBodyElement) it.next();
     initLog.append("\nelement local2").append(
```

```
          bodyElement.
          getElementName().getLocalName());
      try {
        Iterator it2 = bodyElement.
            getChildElements();
        SOAPElement se = (SOAPElement) it2.next();
        initLog.append(
            "\nDette er første subelement:").
            append(se.
                   getElementName().getLocalName());
        return se.getElementName().getLocalName();
      }
      catch (Exception e) {
        initLog.append(
            "\n sub elementer gikk ikke så bra");
        return null;
      }
    }
    catch (Exception e) {
      initLog.append(
          "DPCSOAPMessage encoutered an error:"
          + e.toString());
      return null;
    }

  }

  /**
   * Fetches the parameters from the input. The parametes are returned in
   * a String containing all parameters as they appear in the soap
   * message. That is, as space separated words.
   * @param SOAPMessage the SOAPMessage to fetch the parameters from
   * @param String the command that has the parameters you want
   * @return String the string value of the parameters.
   */
  public String getParameters(SOAPMessage msg,
                              String cmd) {
    initLog.append(
        "\n\nDPCSOAPMessage.getParams() kjører nå...\n");
    try {
      SOAPPart sp = msg.getSOAPPart();
      SOAPEnvelope env = sp.getEnvelope();
      SOAPBody sb = env.getBody();
      initLog.append("\nelement local name:").
          append(sb.getElementName().
                 getLocalName());
      try {
        Iterator it = sb.getChildElements();
        try {
          initLog.append("\njalla1\n");
          int i = 0;
          while (it.hasNext()) {
            i++;
            SOAPElement bodyElement = (
                SOAPElement) it.next();
            initLog.append(i + ". "
```

```
                           + bodyElement.
                           getElementName().
                           getLocalName());
            try {
              Iterator it2 = bodyElement.
                  getChildElements();
              SOAPElement element = (SOAPElement)
                  it2.next();
              initLog.append("command"
                             + element.
                             getElementName().
                             getLocalName());
              dpcCommand = element.getElementName().
                  getLocalName();
              initLog.append("parameter"
                             + element.getValue());
              dpcParameters = element.getValue();
            }
            catch (Exception e) {
              initLog.append(
                  "iteartor gikk ikke2");
              dpcParameters = null;
            }
          }
        }
        catch (Exception e) {
          initLog.append(
              "Iterator 1 .hasNext gikk ikke");
          dpcParameters = null;
        }
      }
      catch (Exception e) {
        initLog.append("Iterator 1 gikk ikke");
        dpcParameters = null;
      }
      return dpcParameters;
    }
    catch (Exception e) {
      return
          "DPCSOAPMessage encoutered an error:"
          + e.toString();
    }

  }

  /***
   * This method gives you an Iterator of all attachments in this object.
   */
  public Iterator getAttachments() {
    return dpcAttchments;
  }
}
```

G.3 DPC.Conf

```
package DPC;

import javax.servlet.*;
import javax.servlet.http.*;
import DPC.Interface.SearchInterface;
import java.util.*;

/**
 * <p>Title: PDC WebService Server</p>
 * <p>Description: DPC Servlet/WebService</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Sverre Joki
 * @version 1.0
 * <p>
 * Denne klassen tar seg av initialisering av DPC servleten.
 * </p>
 * <p>
 * systemet har flere klasser som bruker samme konfig, og denne klassen
 * sørger for å laste denne konfigen inn i servletkonteksten.
 * </p>
 */
public class Conf
    extends HttpServlet {

  /**
   * Dette er objektet som vi lagrer alle init-parametrene fra web.xml
   * fila i. Dette objektet lagres i servlet konteksten og er dermed
   * tilgjengeli i alle servletene som kjører i denne applikasjonen.
   * Dette åpner for at vi kan lagre mange verdier her om vi trenger.
   */
  Properties props;

  /**
   * Tom konstruktør for et konfigurasjonsobjekt. Denne klassen kalles
   * når servlet serveren (tomcat) starter. Dette er satt opp i web.xml
   * fila somligger i WEB-INF katalogen for DPC applikasjonen. Når Conf
   * objektet kallesvil det først kalle init metoden som er en del av
   * Servlet strukturen. Dette skjer automatisk så lenge load-on-startup
   * er satt i web.xml.
   */
  public Conf() {}

  /***
   * Setter opp conigen til servletene. Laster inn parametrene fra
   * web.xmlfila, og setter dem inn i et properies objekt som igjen
   * gjøres tilgjengelig i servletcontexten til denne servlet maskina.
   * For å bruke dette objektet må man kalle
   * <pre>
   * ServletContext ctx = conf.getServletContext();
   * Propserties p = (Properties)ctx.getAttribute("conf");
   * p.getProperty("dpc.serv.url");
   * </pre>
```

```
   * @param conf the ServletConfig
   * @return void
   * @throws ServletException in case of errors
   */
  public void init(ServletConfig conf) throws ServletException {
    System.err.println("DPC.Conf()");
    super.init(conf);
    try {
      System.err.println("Forsøker å hente paramere fra web.xml");
      for (Enumeration e = conf.getInitParameterNames();
           e.hasMoreElements(); ) {
        String name, value;
        name = (String) e.nextElement();
        System.err.println("name=" + name);
        value = conf.getInitParameter(name);
        System.err.println("value=" + value);
        props.setProperty(name, value);
        try {
          System.err.println(
              "Forsøker å lage et searchInterafcea objekt");
          SearchInterface si = new SearchInterface();
          conf.getServletContext().setAttribute("searchinterface", si);
        }
        catch (Exception e1) {
          System.err.println(
              "Searchinterface objekt gikk ikke å lage");
          throw new ServletException("searchinterface init error");
        }
      }
      conf.getServletContext().setAttribute("conf", props);
    }
    catch (Exception e) {
      throw new ServletException(
          "initparameter or searchinterface init error");
    }
  }
}
```

G.4 DPC.DB.BookShelf

```
package DPC.DB;

import DPC.DB.*;
import DPC.Interface.*;
import DPC.xml.*;
import DPC.*;
import java.util.ArrayList;
import java.util.Iterator;
import javax.xml.soap.*;
import javax.xml.messaging.*;

/**
 * <p>Title: PDC WebService</p>
 * <p>Description: DPC Servlet/WebService</p>
 * <p>
 * This class represent a BookShelf(personal collection) for a
 * single user.
 * </p>
 * @author Sverre Joki
 * @version 1.0
 */

public class BookShelf {
  //Bookshelf varaibles/objects

  /***
   * This attribte contains the user data for the given bookshelf
   */
  private UserData userData;

  /***
   * This attribute contains the personal desctiptions for
   * the given bookshelf
   */
  private ArrayList personalDescriptions;

  /***
   * Object used to create a SOAPMessage.
   */
  private MessageFactory mf;

  /***
   * A SOAPMessage that is returned inthe methods
   */
  private SOAPMessage msg;

  /***
   * Part of a SOAPMessage that is used when parsing the UserData
   * object, and creating the SOAPMessage
   */
  private SOAPPart sp;

  /***
```

```
 * Part of a SOAPMessage that is used when parsing the UserData
 * object, and creating the SOAPMessage
 */
private SOAPEnvelope envelope;

/***
 * Part of a SOAPMessage that is used when parsing the UserData
 * object, and creating the SOAPMessage
 */
private SOAPHeader hdr;

/***
 * Part of a SOAPMessage that is used when parsing the UserData
 * object, and creating the SOAPMessage
 */
private SOAPBody bdy;

/***
 * Part of a SOAPMessage that is used when parsing the UserData
 * object, and creating the SOAPMessage
 */
private SOAPBodyElement bodyElement;

/***
 * This contructor creates an BookShelf  object, but does not
 * add any data to the object.
 */
public BookShelf() {
}

/***
 * This method retrieves an ArrayList of personal descriptions in this
 * bookshelf.
 * @return ArrayList an array of personal desctiption
 *                   (PersonaDescription objects)
 */

public ArrayList getPersonalDescriptions() {
  return personalDescriptions;
}

/***
 * This method adds a Local MetadataRecord to the ArrayList
 * containg all PersonalDescription objects.
 * @param desc the PersonalDescription object to add to tha ArrayList
 * @return void
 */
public void addPersonalDescription(PersonalDescription desc) {
  personalDescriptions.add(desc);
}

/***
 * This method sets the objects in the ArrayList, list, into the
 * PersonalDescription ArrayList in this object.
 * @param list the ArrayList containing the objects to add to the
 *              PersonalDescription ArrayList
```

```
 * @return void
 */

public void setPersonaDescriptions(ArrayList list) {
  personalDescriptions = list;
}

/**
 * Returns the userdata in this bookshelf
 * @return UserData a object describing the user
 */
public UserData getUserData() {
  return userData;
}

/**
 * Sets the user in this bookshelf.
 * @param innUserData the user data to use as user in this bookshelf
 * @return void
 */
public void setUserData(UserData innUserData) {
  userData = innUserData;
}


/**
 * <p>
 * This method send this bookshelf by mail to the user already
 * registered in  the bookshelf. Returns a boolean indicating success.
 * </p>
 * <p>
 * NOT IMPLEMENTED YET!</p>
 * @return boolean true if the sending goes ok, false if not.
 */
public boolean sendByMail() {
  boolean ok = false;
  return ok;
}

/***
 * This method converts this bookshelf to an xml document that is
 * returned in a pure text string.
 * @return String the xml document string
 * @throws Exception if something goes wrong
 */
public String convertToXml() throws Exception {
  if (! (userData == null)) {
    StringBuffer xml = new StringBuffer();
    //add start xml tag
    xml.append(getBookShelfXmlStart());
    //add userdata
    xml.append(userData.getUserDataXml());
    //add personalrecordS
    Iterator it = personalDescriptions.iterator();
    while (it.hasNext()) {
      //get each personaldescription
```

```
        PersonalDescription pd = (PersonalDescription)
            it.next();
        xml.append(pd.getPersonalDescriptionAsXml());
      }
      return xml.toString();
    }
    else {
      throw new Exception(
          "no userdata present i bookshelf");
    }
  }

  /***
   * This method creates a x ml document header for a Bookshelf
   * xml document.
   * @return String the xml document header text.
   */
  public String getBookShelfXmlStart() {
    StringBuffer out = new StringBuffer();
    out.append(
        "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
    out.append("<!DOCTYPE User SYSTEM \"http://www.idi.ntnu.no/~sverrema/
dpc/bookshelf.dtd\">\n");
    return out.toString();
  }
}
```

G.5 DPC.DB.CollectionRecord

```
package DPC.DB;

import DPC.DB.*;
import DPC.Interface.*;
import DPC.xml.*;
import DPC.*;
import java.util.ArrayList;
import java.util.Iterator;

/**
 * <p>Title: PDC WebService</p>
 * <p>Description: DPC Servlet/WebService</p>
 * <p>
 * This class is an internal representation of a record in the collection
 * database. This object keeps track of the personal description,
 * user id, and subject word of this record.
 * </p>
 * @author Sverre Joki
 * @version 1.0
 */
public class CollectionRecord {

  /***
   * This attribute holds the value for the reocrd Id of this record
```

```
   * in the collections database
   */
  private String recordID = null;

  /***
   * This attribute has the value for the userID of the user that
   * has written the personal desctiption
   */
  private String userID = null;

  /***
       * This attribute has the subject words that the given user has used
   * to describe the personal desription
   */
  private ArrayList subjectWords = null;

  /***
       * This attribute holds the personal descripton or personal metadata
   * that a user has added to a metadata record.
   */
  private String personalDescription = null;

  /***
   * Constructs an empty CollectionRecord object
   */
  public CollectionRecord() {
  }

  /***
   * This constructor  creates a CollectionRecord object using the
   * given paramters.
   * @param uid the userId to use for the CollectionRecord
   * @param recordID the recordId to use for the CollectionRecord
   * @param subjectWordsIn the subject words to use
   *                       for the CollectionRecord
   * @param description the description to use for the CollectionRecord
   */
  public CollectionRecord(String uid, String recordID,
                          ArrayList subjectWordsIn,
                          String description) {
    setUserID(uid);
    setRecordID(recordID);
    setSubjectWords(subjectWordsIn);
    personalDescription = description;
  }

  /**
   * Tis method resets all attributes within this object to null.
   * @return void
   */
  public void reset() {
    recordID = null;
    userID = null;
    subjectWords = null;
    personalDescription = null;
  }
```

```
/***
 * This method sets the user id of this object to the value specified
 * in the parameter uid.
 *
 * @param uid the user id to use as userID in this object
 * @return void
 */
public void setUserID(String uid) {
  userID = uid;
}

/**
 * This method insterts the recordID string into the recordID attribute
 * of this object.
 * @param recordId the value to add to the recordID of this object
 * @return void
 */
public void setRecordID(String recordID) {
  this.recordID = recordID;
}

/***
 * This method sets the description of this object to the value
 * specified in the parameter description.
 *
 * @param description the description to use as description
 *                    in this object
 * @return void
 */
public void setDescription(String description) {
  personalDescription = description;
}

/***
 * This method adds a subject word to the list of  subject words
 * in this object.
 * The word to be added is supplied in a string parameter.
 *
 * @param word the word to add to the subjectWords in this object
 * @return void
 */
public void addsubjectWord(String word) {
  subjectWords.add(word);
}

/***
 * This method sets the subject words to use in this object.
 * The subject words to be used is supplied in a ArrayList parameter.
 *
 * @param words the ArrayList of words to use as the
 *              subjectWords in this object
 * @return void
 */
public void setSubjectWords(ArrayList words) {
  subjectWords = words;
```

```
  }

  /***
   * This method sets the subject words to use in this object.
   * The subject words to be used is supplied in a string array parameter.
   *
   * @param words the string array of words to use as
   * the subjectWords in this object
   * @return void
   */
  public void setSubjectWords(String[] words) {
    for (int i = 0; i < words.length; i++) {
      subjectWords.add(words[i]);
    }
  }

  /***
   * This method retrievs the userID used in this object
   * @return String the userId of this object
   */
  public String getUserID() {
    return userID;
  }

  /***
   * This method retrievs the personalDescription used in this object
   * @return String the personaDescription of this object
   */
  public String getDescription() {
    return personalDescription;
  }

  /***
   * This method retrievs the subjectWords used in this object
   * @return ArrayList the subjectwords of this object
   */
  public ArrayList getsubjectWords() {
    return subjectWords;
  }

  /***
   * This method retrieves the record ID for this object.
   * @return String the record id for this object
   */
  public String getRecordID() {
    return recordID;
  }

  /**
   * This method converts this object to a xml document and also
   * adds a xml document header.
   * @return String this object converted to a well formed xml document
   */
  public String convertToXml() {
    StringBuffer sb = new StringBuffer();
    sb.append("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
```

```
    sb.append("<!DOCTYPE MetadataRecord SYSTEM
\"H:\\Hovedoppgave\\system\\dpc-database\\LocalMetadataRecord.dtd\">");
    sb.append("<CollectionRecord>");
    sb.append("<UserID>");
    sb.append(this.getUserID());
    sb.append("</UserID>");
    sb.append("<RecordID>");
    sb.append(this.getRecordID());
    sb.append("</RecordID>");
    //Iterate through personal descriptions
    ArrayList list = this.getsubjectWords();
    if (list != null) {
      sb.append("<SubjectWords>");
      Iterator it = list.iterator();
      while (it.hasNext()) {
        PersonalDescription pd = (PersonalDescription) it.next();
        sb.append("<UserID>");
        sb.append(pd.getUserID());
        sb.append("</UserID>");
        ArrayList sList = pd.getsubjectWords();
        if (sList != null) {
          Iterator sit = sList.iterator();
          while (sit.hasNext()) {
            String word = (String) sit.next();
            sb.append("<SubjectWord>");
            sb.append(word);
            sb.append("</SubjectWord>");
          }
        }
        sb.append("<Description>");
        sb.append(pd.getDescription());
        sb.append("</Description>");

      }
      sb.append("</PersonalDescriptions>");
    }
    sb.append("</MetadataRecord>");

    return sb.toString();
  }
}
```

G.6 DPC.DB.LocalMetadataRecord

```
package DPC.DB;

import DPC.DB.*;
import DPC.Interface.*;
import DPC.xml.*;
import DPC.*;
import java.util.ArrayList;
import java.util.Iterator;
import javax.xml.soap.*;
import javax.xml.messaging.*;

/***
 * This class is an internal representation of a local metadata record.
 * This class has attribuites to keep a list of personal desriptions
 * and a list of users. There is also an URI attribute, to link
 * descriptions and users to a metadata record.
 * @author
 */
public class LocalMetadataRecord {

  /***
   * This attribute is the id for the loca metadata record
   */
  private String localRecordID;

  /***
   * This attribute id an ArryList containing PersonalDescripton objects
   */
  private ArrayList descriptions;

  /***
   * This attribute id an ArryList containing userIDs
   */
  private ArrayList users;

  /***
   * This attribute id an ArryList containing the
   * URI to the metadata record
   */
  private String metadataURI;

  //SOAPMessage variables
  private MessageFactory mf;
  private SOAPMessage msg;
  private SOAPPart sp;
  private SOAPEnvelope envelope;
  private SOAPHeader hdr;
  private SOAPBody bdy;
  private SOAPBodyElement bodyElement;

  /***
   * Constructs an empty LocalMetadata object
   */
```

```
public LocalMetadataRecord() {
  descriptions = new ArrayList();
  users = new ArrayList();
  localRecordID = null;
  metadataURI = null;

}

/***
 * This method returns the LocalMetadataRecord object.
 * @return the LocalMetadataRecord object.
 */
public LocalMetadataRecord get() {
  return this;
}

/**
 * This method insterts the recordID string into the
 * localRecordID attribute of this object.
 * @param recordId the value to add to the localRecordID of this object
 * @return void
 */
public void setRecordID(String recordID) {
  localRecordID = recordID;
}

/***
 * This method returns the localRecordID of this object.
 * @return String the localRecordID of this object
 */
public String getRecordID() {
  return localRecordID;
}

/**
 * This method insterts the metadataURI string into the
 * metadataURI attribute of this object.
 * @param metadataURI the value to inset as the
 *                    metadataURI of this object
 * @return void
 */
public void setmetadataURI(String metadataURI) {
  this.metadataURI = metadataURI;
}

/***
 * This method returs the metadataURI of this object
 * @return the metadataURI of this object
 */
public String getmetadataURI() {
  return metadataURI;
}

/**
 * utgår
 */
```

```
//  public void setPersonalDescription(PersonalDescription pd) {
//    descriptions.add(pd);
//  }

  /**
   * utgår
   *
     public ArrayList getPersonalDescription() {
    return descriptions;
     }
   */

  /***
   * This method sets the ArrayList of users of this object.
   * The ArrayList must contain userIDs for eash user.
   * @param users the ArrayList of users to set for this object
   * @return void
   */
  public void setUsers(ArrayList users) {
    this.users = users;
  }

  /***
   * This method returns the ArrayList of users for this object.
   * @return the ArrayList of users for this object.
   */
  public ArrayList getUsers() {
    return users;
  }

  /**
   * This method adds a single userID to the ArrayList of users
   * in this object.
   * @param uid the user ID to add to the ArrayList
   *            of users in this object
   * @return void
   */
  public void addUser(String uid) {
    users.add(uid);
  }

  /**
   * This is a utility method to convert this LocalMetadataRecord intto
   * a soapmessage ready to be sent to a client.
   * @param metaDataArray this is an ArrayList containing
   *                      LocalMetadataRecord objects that is to be
   *                      converted into a SOAPMessage
   * @return SOAPMessage a SOAPMessage of the LocalMetadataRecord
   *                     objects in the parameter
   */
  public SOAPMessage convertToSOAP(ArrayList metaDataArray) throws
      SOAPException {

    try {
      //-------------------------
      //Setting up the soap message
```

```
    //-------------------------

    // Create a message factory.
    mf = MessageFactory.newInstance();

    // Create a message from the message factory.
    msg = mf.createMessage();

    // Message creation takes care of creating the SOAPPart - a
    // required part of the message as per the SOAP 1.1
    // specification.
    sp = msg.getSOAPPart();

    // Retrieve the envelope from the soap part to start building
    // the soap message.
    envelope = sp.getEnvelope();

    // Create a soap header from the envelope.
    hdr = envelope.getHeader();

    // Create a soap body from the envelope.
    bdy = envelope.getBody();

    // Add a soap body element to the soap body
    bodyElement = bdy.addBodyElement(envelope.createName(
        "DPC_Return_Message"));

    //-------------------------
    //Populating soapmessage with userdata objects
    //-------------------------

    Iterator it = metaDataArray.iterator();
    while (it.hasNext()) {
      LocalMetadataRecord lm = (LocalMetadataRecord) it.next();
      //Create a UserDataElement that contains all needed data
      createMetaDataElement(lm);
    }
    return msg;
  }
  catch (SOAPException e) {
    throw new SOAPException("Error converting Userdata to SOAP");
  }

}

/**
 * <p>
 * Genereates a SOAPMessage based on the LocalMetadataRecord
 * object in this object.</p>
 * <p>
 * This method creates one SOAPMessage based on ONE userdata object.
 * </p>
 * @return SOAPMessage a SOAPMessage of this object
 * @throws SOAPException if something goes wrong during the convertion.
 */
public SOAPMessage convertToSOAP() throws SOAPException {
```

```
    try {
      // Create a message factory, overwriting any old ones
      mf = MessageFactory.newInstance();

      // Create a message from the message factory.
      msg = mf.createMessage();

      // Message creation takes care of creating the SOAPPart - a
      // required part of the message as per the SOAP 1.1
      // specification.
      sp = msg.getSOAPPart();

      // Retrieve the envelope from the soap part to start building
      // the soap message.
      envelope = sp.getEnvelope();

      // Create a soap header from the envelope.
      hdr = envelope.getHeader();

      // Create a soap body from the envelope.
      bdy = envelope.getBody();

      // Add a soap body element to the soap body
      bodyElement = bdy.addBodyElement(envelope.createName(
          "DPC_Return_Message"));

      //Create a UserDataElement that contains all needed data
      createMetaDataElement(this);

      return msg;
    }
    catch (SOAPException e) {
      throw new SOAPException("Error converting Userdata to SOAP");
    }

  }

  /***
   * This method converts the given LocalMetadataRecord into a
   * SOAPElement. This canthe be added to a SOAPMessage object.
   * This method has to be used from within this class.
   * @param lm the LocalMetadataRecord to convert to a SOAPElement
   * @return SOAPElement the converted LocalMetadataRecord
   * @throws SOAPException if something goes wrong during convertion
   */
  private SOAPElement createMetaDataElement(LocalMetadataRecord lm)
      throws SOAPException {

    //Add a userData element to the root element
    SOAPElement metadataElement =
        bodyElement.addChildElement(envelope.createName(
        "LocalMetadataRecord"));
    //add elements to the metadata element
    metadataElement.addChildElement(envelope.createName("RecordID"))
        .addTextNode(lm.getRecordID());
```

```
    metadataElement.addChildElement(envelope.createName("MetadataURI"))
        .addTextNode(lm.getmetadataURI());

    //Iterate through Users
    ArrayList list = lm.getUsers();
    Iterator it = list.iterator();
    while (it.hasNext()) {
      String uid = (String) it.next();
      metadataElement.addChildElement(envelope.createName("UserID"))
          .addTextNode(uid);

    }

    return metadataElement;
  }

  /***
   * This method converst this object into a xml document (string).
   * @return String the xml document of this object
   */

  public String convertToXml() {
    StringBuffer sb = new StringBuffer();
    sb.append("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
    sb.append("<!DOCTYPE MetadataRecord SYSTEM
\"H:\\Hovedoppgave\\system\\dpc-database\\LocalMetadataRecord.dtd\">");
    sb.append("<MetadataRecord>");
    sb.append("<RecordID>");
    sb.append(this.getRecordID());
    sb.append("</RecordID>");
    sb.append("<MetadataURI>");
    sb.append(this.getmetadataURI());
    sb.append("</MetadataURI>");
    //Iterate through Users
    ArrayList list = this.getUsers();
    Iterator it = list.iterator();
    while (it.hasNext()) {
      sb.append("<UserID>");
      String uid = (String) it.next();
      sb.append(uid);
      sb.append("</UserID>");
    }
    sb.append("</MetadataRecord>");

    return sb.toString();
  }

}
```

G.7 DPC.DB.PersonalDescription

```
package DPC.DB;

import DPC.DB.*;
import DPC.Interface.*;
import DPC.xml.*;
import DPC.*;
import java.util.ArrayList;
import java.util.Iterator;

/**
 * <p>Title: PDC WebService</p>
 * <p>Description: DPC Servlet/WebService</p>
 * <p>
 * This class is an internal representation of a record in the
 * metadatabase. This class has attributes for a userID,
 * personal description and a string of subject words that describe
 * the personal metadata in the desription attribute.
 * </p>
 * @author Sverre Joki
 * @version 1.0
 */
public class PersonalDescription {

    /***
     * This attribute has the value for the userID of the user that
     * has written the personal desctiption
     */
    private String userID = null;

    /***
     * This attribute has the subject words that the given user has used
     * to describe the personal desription
     */
    private ArrayList subjectWords = null;

    /***
     * This attribute holds the personal descripton or personal metadata
     * that a user has added to a metadata record.
     */
    private String personalDescription = null;

    /***
     * Constructs an empty PersonalDescription object
     */
    public PersonalDescription() {
    }

    /***
     * Constructs a PersonalDescripton object using the given parameters
     * for userID, subject words and personal desctiption.
     * @param uid the userID to use for the PersonalDescription object
     * @param subjectWordsIn a string array containing the subject
     *                       words to add to this object
```

```
 * @param descrption a string containing the personal description
 */
public PersonalDescription(String uid, String[] subjectWordsIn,
                           String description) {
    setUserID(uid);
    setSubjectWords(subjectWordsIn);
    personalDescription = description;
}

/**
 * This methos resets all attributes within this object to null.
 * @return void
 */
public void reset() {
    userID = null;
    subjectWords = null;
    personalDescription = null;
}

/***
 * This method sets the user id of this object to the value specified
 * in the parameter uid.
 *
 * @param uid the user id to use as userID in this object
 * @return void
 */
public void setUserID(String uid) {
    userID = uid;
}

/***
 * This method sets the description of this object to the value
 * specified in the parameter description.
 *
 * @param description the description to use as personalDescription
 *                    in this object
 * @return void
 */
public void setDescription(String description) {
    personalDescription = description;
}

/***
 * This method adds a subject word to the list of  usbject words
 * in this object.
 * The word to be added is supplied in a string parameter.
 *
 * @param word the word to add to the subjectWords in this object
 * @return void
 */
public void addsubjectWord(String word) {
    subjectWords.add(word);
}

/***
 * This method sets the subject words to use in this object.
```

```
 * The subject words to be used is given in a ArrayList parameter.
 *
 * @param words the ArrayList of words to use as the subjectWords
 *              in this object
 * @return void
 */
public void setSubjectWords(ArrayList words) {
    subjectWords = words;
}

/***
 * This method sets the subject words to use in this object.
 * The subject words to be used is supplied in a string array
 *  parameter.
 *
 * @param words the string array of words to use as the
 *               subjectWords in this object
 * @return void
 */
public void setSubjectWords(String[] words) {
    for (int i = 0; i < words.length; i++) {
        subjectWords.add(words[i]);
    }
}

/***
 * This method retrievs the userID used in this object
 * @return String the userId of this object
 */
public String getUserID() {
    return userID;
}

/***
 * This method retrievs the personalDescription used in this object
 * @return String the personaDescription of this object
 */
public String getDescription() {
    return personalDescription;
}

/***
 * This method retrievs the subjectWords used in this object
 * @return ArrayList the subjectwords of this object
 */
public ArrayList getsubjectWords() {
    return subjectWords;
}

/**
 * This method converts this object to a xml document and also
 * adds a xml document header.
 * @return String this object converted to a well formed xml document
 */
public String convertToXmlDocument() {
    StringBuffer sb = new StringBuffer();
```

```
        sb.append(getPersonalDescriptionAsXml());
        sb.append(getPersonalDescriptionAsXml());
        return sb.toString();
    }

    /**
     * This method returns the xml elements of the attributes of this
     * object. This inclueds PersonalDescription, UserID,
     * and SubjectWords.
     * @return String the xml elements of this object.
     */
    public String getPersonalDescriptionAsXml() {
        StringBuffer sb = new StringBuffer();
        sb.append("<PersonalDescription>");

        sb.append("<UserID>");
        sb.append(userID);
        sb.append("</UserID>");

        Iterator it = subjectWords.iterator();
        while (it.hasNext()) {
            sb.append("<SubjectWord>");
            sb.append( (String) it.next());
            sb.append("</SubjectWord>");

        }

        sb.append("<Descrtiption>");
        sb.append(personalDescription);
        sb.append("</Description>");

        sb.append("</PersonalDescription>");

        return sb.toString();
    }

    /**
     * This method returns a string of the document header for this
object.
     * @return String the xm ldocument header to make the xml
     *                document well formed.
     */
    public String getPersonalDescriptionXmlStart() {
        StringBuffer sb = new StringBuffer();
        sb.append("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
        sb.append("<!DOCTYPE User SYSTEM \"jallajalla\">");
        return sb.toString();

    }
}
```

G.8 DPC.DB.UserData

```
package DPC.DB;

import DPC.DB.*;
import DPC.Interface.*;
import DPC.xml.*;
import DPC.*;
import java.util.StringTokenizer;
import java.util.ArrayList;
import java.util.Iterator;
import javax.xml.soap.*;
import javax.xml.messaging.*;
import javax.xml.transform.*;

/**
 * This class is a java implementation of the data stored about a user
 * in the UserDataBase. The data about the user if contained in the
 * attributes of this class.
 * @author Sverre Joki
 */
public class UserData {

  /***
   * This object is used to store the name of this particular user.
   */
  public String name;

  /***
   * This object is used to store the userID of this particular user.
   */
  public String userID;

  /***
   * This object is used to store the adress of this particular user.
   */
  public String adress;

  /***
   * This object is used to store the post code of this particular user.
   */
  public String postCode;

  /***
   * This object is used to store the city of this particular user.
   */
  public String city;

  /***
   * This object is used to store the email-adress for
   * this particular user.
   */
  public String email;

  /***
```

```
 * This object is used to store the position of this particular user.
 */
public String position;

/***
 * This object is used to store the instituion this particular
 * use belongs to
 */
public String institution;

/***
 * This object is used to store the interestes of
 * this particular user.
 */
public ArrayList interests;

/***
 * This object is used to store a decrition of this particular user.
 */
public String description;

//SOAPMessage stuff
private MessageFactory mf;
private SOAPMessage msg;
private SOAPPart sp;
private SOAPEnvelope envelope;
private SOAPHeader hdr;
private SOAPBody bdy;
private SOAPBodyElement bodyElement;

public String xmlRawData;

public UserData() {}

/**
 * Creates a new UserData object with the parameteres given.
 * All parameters are given in strings, but the interests.
 * Interest is a comma separated string containing the insterest
 * for the particular user.
 *
 * @param userName the name of the user you want to create an
 *                   userobject for
 * @param userID the userID  of the user you want to create an
 *                 userobject for
 * @param adress the adress of the user you want to create an
 *                 userobject for
 * @param postCode the post Code of the user you want to create an
 *                 userobject for
 * @param city the city of the user you want to create an
 *               userobject for
 * @param email the email of the user you want to create an
 *                userobject for
 * @param interest the interest of the user you want to create an
 *                   userobject for
 * @param position the position of the user you want to create an
 *                   userobject for
```

```
 * @param instituion the instituion of the user you want to create
 *                    an userobject for
 * @param description the description of the user you want to
 *                     create an userobject for
 * @param xmlString an xml string
 */
public UserData(String userName, String userID, String adress,
                String postCode, String city, String email,
                String interest,
                String position, String instituion,
                String description,
                String xmlString) {

  name = userName;
  this.userID = userID;
  this.adress = adress;
  this.postCode = postCode;
  this.city = city;
  this.email = email;
  this.position = position;
  this.institution = instituion;
  //parse interests
  StringTokenizer st = new StringTokenizer(interest, ", ");
  while (st.hasMoreTokens()) {
    interests.add(st.nextToken());
  }
  this.description = description;
  xmlRawData = xmlString;
}

/***
 * This method ocnverts an ArrayList containing several user objects,
 * to a SOAPMessage
 * @param userDataArray an array containg the userdata object from.
 *                      Each userdata object is then created into
 *                     an xml-structure and inserted into a SOAPMessage.
 * @return SOAPMessage that can be sent to the sender. This message
 *                      contains all users in the userDataArray
 */
public SOAPMessage convertToSOAP(ArrayList userDataArray) throws
    SOAPException {

  try {
    //-------------------------
    //Setting up the soap message
    //-------------------------

    // Create a message factory.
    mf = MessageFactory.newInstance();

    // Create a message from the message factory.
    msg = mf.createMessage();

    // Message creation takes care of creating the SOAPPart - a
    // required part of the message as per the SOAP 1.1
    // specification.
```

```
    sp = msg.getSOAPPart();

    // Retrieve the envelope from the soap part to start building
    // the soap message.
    envelope = sp.getEnvelope();

    // Create a soap header from the envelope.
    hdr = envelope.getHeader();

    // Create a soap body from the envelope.
    bdy = envelope.getBody();

    // Add a soap body element to the soap body
    bodyElement = bdy.addBodyElement(envelope.createName(
        "DPC_Return_Message"));

    //--------------------------
    //Populating soapmessage with userdata objects
    //--------------------------

    Iterator it = userDataArray.iterator();
    while (it.hasNext()) {
      UserData userData = (UserData) it.next();
      //Create a UserDataElement that contains all needed data
      createUserDataElement(userData);
    }

    return msg;
  }
  catch (SOAPException e) {
    throw new SOAPException("Error converting Userdata to SOAP");
  }

}

/**
 * Genereates a SOAPMessage based on the this object.
 * This method creates one SOAPMessage based on ONE userdata object
 * @return SOAPMessage based on the data in this object.
 */
public SOAPMessage convertToSOAP() throws SOAPException {
  return convertToSOAP(this);
}

/**
 * Genereates a SOAPMessga based on the Userdata object given as input.
 * This method creates one SOAPMessage based on ONE userdata object
 * @param userData a UserData object to convert into a SOAPMessage
 * @return a SOAPMessage is returned
 */
public SOAPMessage convertToSOAP(UserData userData) throws
    SOAPException {

  try {
    // Create a message factory.
    mf = MessageFactory.newInstance();
```

```
      // Create a message from the message factory.
      msg = mf.createMessage();

      // Message creation takes care of creating the SOAPPart - a
      // required part of the message as per the SOAP 1.1
      // specification.
      sp = msg.getSOAPPart();

      // Retrieve the envelope from the soap part to start building
      // the soap message.
      envelope = sp.getEnvelope();

      // Create a soap header from the envelope.
      hdr = envelope.getHeader();

      // Create a soap body from the envelope.
      bdy = envelope.getBody();

      // Add a soap body element to the soap body
      bodyElement = bdy.addBodyElement(envelope.createName(
          "DPC_Return_Message"));

      //Create a UserDataElement that contains all needed data
      createUserDataElement(userData);

      return msg;
    }
    catch (SOAPException e) {
      e.printStackTrace(System.err);
      throw new SOAPException("Error converting Userdata to SOAP");
    }

  }

  /***
   * This method converts a UserData object info a SOAPElement.
   * This method can be used if you vwant to convert a lot of
   * UserData  object into a SOAPMessage. Each UserData object can
   * then be added to a SOAPMessage as a SOAPElement
   * @param ud the UserData object you want to convert to a SOAPElement
   * @return SOAPElement the SOAPElement after the convertion of the
   *                     UserData object.
   * @throws SOAPException if something goes wrong
   */
  private SOAPElement createUserDataElement(UserData ud) throws
      SOAPException {

    //Add a userData element to the root element
    SOAPElement userdataElement =
        bodyElement.addChildElement(envelope.createName("UserData"));
    //add elements to the userdata element
    userdataElement.addChildElement(envelope.createName("UserID"))
        .addTextNode(ud.userID);
    userdataElement.addChildElement(envelope.createName("Name"))
        .addTextNode(ud.name);
```

```
    userdataElement.addChildElement(envelope.createName(
        "StreetAdress"))
        .addTextNode(ud.adress);
    userdataElement.addChildElement(envelope.createName("PostCode"))
        .addTextNode(ud.postCode);
    userdataElement.addChildElement(envelope.createName("City"))
        .addTextNode(ud.city);
    userdataElement.addChildElement(envelope.createName("Email"))
        .addTextNode(ud.email);
    userdataElement.addChildElement(envelope.createName("Position"))
        .addTextNode(ud.position);
    userdataElement.addChildElement(envelope.createName("Institution"))
        .addTextNode(ud.institution);
    //Adding interest is a little more comlicated because it is more tha
one
    ArrayList interests = ud.interests;
    Iterator intIt = interests.iterator();
    while (intIt.hasNext()) {
      String interestString = (String) intIt.next();
      userdataElement.addChildElement(envelope.createName("Interest"))
          .addTextNode(interestString);
    }
    userdataElement.addChildElement(envelope.createName("Description"))
        .addTextNode(ud.description);

    return userdataElement;
  }

  /***
   * This method ocnverts this object into a xml-document in a string
   * object. This xml document can then be sent to a database for
   * storage. It can also be added to a SOAPMessage and sent.
   * @return String the xml document of <code>this</code> object.
   */
  public String convertToXML() {
    StringBuffer sb = new StringBuffer();
    sb.append(getXmlElementTag());
    sb.append(getUserDataXml());
    return sb.toString();
  }

  /***
   * This method is used to create a xml string from the data contained in
   * this UserData object.
   * @return String the xml elements of this UserData object
   */
  public String getUserDataXml() {
    StringBuffer sb = new StringBuffer();
    sb.append("<User>");
    sb.append("<UserID>");
    sb.append(this.userID);
    sb.append("</UserID>");
    sb.append("<Name>");
    sb.append(this.name);
    sb.append("</Name>");
    sb.append("<StreetAdress>");
```

```
    sb.append(this.adress);
    sb.append("</StreetAdress>");
    sb.append("<City>");
    sb.append(this.city);
    sb.append("</City>");
    sb.append("<PostCode>");
    sb.append(this.postCode);
    sb.append("</PostCode>");
    sb.append("<Email>");
    sb.append(this.email);
    sb.append("</Email>");
    sb.append("<Position>");
    sb.append(this.position);
    sb.append("</Position>");
    sb.append("<Institution>");
    sb.append(this.institution);
    sb.append("</Institution>");
    Iterator it = this.interests.iterator();
    while (it.hasNext()) {

      sb.append("<Interest>");
      sb.append( (String) it.next());
      sb.append("</Interest>");

    }
    sb.append("<Description>");
    sb.append(this.description);
    sb.append("</Description>");
    sb.append("</User>");

    return sb.toString();
  }

  /***
   * This method is used to return the xml document header for this
   * UserData object as a xml document in a string.
   * @return String the xml document header
   */
  public String getXmlElementTag() {
    StringBuffer sb = new StringBuffer();
    sb.append("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
    sb.append("<!DOCTYPE User SYSTEM \"H:\\Hovedoppgave\\system\\dpc-
database\\LocalMetadataRecord.dtd\">");
    return sb.toString();

  }
}
```

G.9 DPC.Interface.CollectionDBInterface

```
package DPC.Interface;

import DPC.DB.*;
import DPC.Interface.*;
import DPC.xml.*;
import DPC.*;
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import org.xmldb.api.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.lang.reflect.*;

/**
 * <p>
 * This class is the communication class that connects the
 * SearchInterface to the Collection Database containing all
 * personal colletion added by the users.
 *</p>
 * @author Sverre Joki
 */
public class CollectionDBInterface
    extends DataBaseInterface {

  //These constants are used to search in the database

  /***
   * String constant identifiying this parameter in the collection
   * database
   */
  public static String USERID = "userid";

  /***
   * String constant identifiying this parameter in the collection
   * database
   */

  public static String RECORDID = "recordid";

  /***
   * The path to the collection in the database
   * to use for the UserDBInterface
   */
  private String collection = "collections";

  /***
   * The path to the collection to use for the database
   */
  private String collectionPath;

  /***
   * This contructor creates a standard CollectionDBInterface object,
   * and initializes the connetcion to the database using the
```

```
   * attributes specified above as <code>driver</code> and
   * <code>collectionPath</code>.
   *
   * @throws Exception If anything goes wrong during initialization, the
   *                   contructor writes the exception to the error log, and
   *                     throws an Exception indication a failure.
   */
  public CollectionDBInterface() throws Exception {
    System.err.println(cal.getTime().toString() + ":"
                       + this.getClass().getName());
    try {
      System.out.println("trying to create "
                         + this.getClass().getName()
                         + " without parameters");
      System.out.println("init(\"org.exist.xmldb.DatabaseImpl\",
\"xmldb:exist://fenris.idi.ntnu.no:7180/exist/db/dpc\");");
      init(super.driver, collectionPath);
    }
    catch (Exception e) {
      System.err.println("init error occured " + e.toString());
      e.printStackTrace(System.err);
      throw new Exception("Error initialzing collectionDB");
    }
  }

  /***
   * This contructor creates a standard CollectionDBInterface object,
   * and initializes the connetcion to the database using the
   * parameteres specified as <code>dbDriver</code> and
   * <code>collectionPath</code>.
   * @param dbDriver the driver class used as database connection driver
   * @param collectionPath the path to the correct collection to use for
   *                              the MCollection Database
   * @throws Exception If anything goes wrong during initialization, the
   *                     contructor writes the exception to the error log,
   *                     and throws an Exception indicating a failure.
   */
  public CollectionDBInterface(String dbDriver, String collectionPath)
      throws Exception {
    System.err.println(cal.getTime().toString() + ":"
                       + this.getClass().getName());
    super.driver = dbDriver;
    this.collectionPath = collectionPath;
    try {
      System.out.println("trying to create "
                         + this.getClass().getName()
                         + " with parameters");
      init(super.driver, collectionPath);
    }
    catch (Exception e) {
      System.err.println("init error occured " + e.toString());
      e.printStackTrace(System.err);
      throw new Exception("Error initialzing collectionDB");

    }
```

```
}

/***
 * This method initializes the connetcion to the database using the
 * parameteres specified
 * as <code>dbDriver</code> and <code>collectionPath</code>
 * @param dbDriver the driver class used as database connection driver
 * @param collectionPath the path to the correct collection to use for
 *                             the MetadataDatabase
 * @throws XMLDBException If anything goes wrong during initialization,
 *       the contructor writes the exception to the error log,
 *       and throws an XMLDBException indicating a failure.
 */
public void init(String driver, String collectionPath) throws
    XMLDBException, Exception {
  //Setting up the connection to the database
  try {
    System.err.println("trying to init this interface: "
                       + this.getClass().getName());
    System.err.println("\nWe are using this driver:" + driver
                       + "\nand this url:" + collectionPath);
    Class c = Class.forName(driver);
    Database database = (Database) c.newInstance();
    DatabaseManager.registerDatabase(database);
    database.setProperty("create-database", "true");
    col = DatabaseManager.getCollection(collectionPath);
    col.setProperty("pretty", "true");
    col.setProperty("encoding", "ISO-8859-1");
    System.err.println("finished with init of this interface: "
                       + this.getClass().getName());
  }
  catch (XMLDBException e) {
    System.err.println("XML:DB Exception occured " + e.toString());
    e.printStackTrace(System.err);
    throw new Exception("XML:DB Exception occured " + e.toString());
  }
  catch (Exception e) {
    System.err.println("An exception occured " + e.toString());
    e.printStackTrace(System.err);
    throw new Exception("An exception occured " + e.toString());
  }
}

/***
 * This method creates a collectionrecord from the data supplied as
 * parameters and adds it to the database. The method uses the
 * collection as set up in the init method or contructor.
 * @param userID the userid of the user you want to add
 * @param recordID the ID of the record you want to add a
 *                 description about
 * @param description the description that the given user has added
 *                          to the metadata-record
 * @param subjectWords an arraylist of subjectwords that is used
 *                              to describe the description given.
 */
public boolean addCollectionRecord(String userID,
```

```
      String recordID,
      String description,
      ArrayList subjectWords) {

    //create new collectionrecord based on input
    CollectionRecord rec =
        new CollectionRecord(userID, recordID, subjectWords,
                             description);
    return addCollectionRecord(rec);
  }

  /***
   * This method adds a collectionrecord to the data database.
   * The method uses the collection database
   * as set up in the init method or contructor.
   * @param CollectionRecord rec the CollectionRecord object that is to
   *                                    be added to the database.
   * @return boolean indicating if the database update went ok.
   */
  public boolean addCollectionRecord(CollectionRecord rec) {

    //create xmlstring (document) based on record
    String xmlRecord = rec.convertToXml();
    try {
      //get a db connection
      //create a userDataobject
      XMLResource document = (XMLResource) col.createResource(null,
          "XMLResource");
      //create a xml document from the userdata object
      document.setContent(xmlRecord);
      col.storeResource(document);
      return true;
    }
    catch (XMLDBException e) {
      e.printStackTrace(System.err);
      return false;
    }
  }

  /***
   * This method get the collection record from the database.
   * The parameter to specify is the userid of the user you want
   * to get the record for. The return object is a array of xmlresources
   * @param userid the user id to look for
   * @return XMLResource the collection record
   */
  public XMLResource getCollectionUser(String userid) throws
      CollectionNotFoundException {
    System.err.println(this.getClass().getName()
                       + ".getCollectionUser(): [" +
                       cal.getTime().toString() + "]:START");
    try {
      //henter resource
      ResourceSet results = getCollection(this.USERID, userid);
      ResourceIterator it = results.getIterator();
      XMLResource res = (XMLResource) it.nextResource();
```

```
    System.err.println(this.getClass().getName() +
                       ".getCollectionUser(): [" +
                       cal.getTime().toString() + "]:END"
                       + res.getContent().toString());

    return res;
  }
  catch (Exception e) {
    //skriver ut feilmelding til catalina.out
    System.err.println(this.getClass().getName()
                       + ".getCollectionUser(): [" +
                       cal.getTime().toString() +
                       "]:no such user found exception:"
                       + e.toString());
    e.printStackTrace(System.err);
    //kaster en Exception om at brukeren ikke er funnet
    throw new CollectionNotFoundException(e.toString());
  }
}

/***
 * This method get the collection record from the database. The
 * parameter to specify is the userid of the user you want to get
 * the record for.
 * @deprecated
 */
public XMLResource getCollectionRecord(String recid) throws
    CollectionNotFoundException {
  try {
    ResourceSet results = getCollection(this.RECORDID, recid);
    ResourceIterator it = results.getIterator();
    XMLResource res = (XMLResource) it.nextResource();
    return res;
  }
  catch (Exception e) {
    throw new CollectionNotFoundException(e.toString());
  }
}

/***
 * This method is used to fetch data from the database based on a
 * parameter and a value. This method connects to the database using
 * the driver and path specified in the attribuets in this class.
 * @param param the database-field to search for data in
 * @param value the value to search for in the specified parameter
 * @return ResourceSet the ResourceSet object containing the
 *               found records
 * @throws XMLDBException if anything goes wrong during the connection
 *               to the database
 */
private ResourceSet getCollection(String param, String value) throws
    XMLDBException {
  System.err.println(this.getClass().getName()
                     + ".getCollection(): [" +
                     cal.getTime().toString() + "]:START");
```

```
    //create XPATH query from userName
    String xpath = "//*[" + param + "='" + value + "']";
    //send query
    System.err.println("Dette er xpath:" + xpath);
    XPathQueryService service =
        (XPathQueryService) col.getService("XPathQueryService", "1.0");
    //get response
    ResourceSet resultSet = service.query(xpath);
    System.err.println(this.getClass().getName()
                       + ".getCollection(): [" +
                       cal.getTime().toString() + "]:END");
    return resultSet;

  }

  /***
   * This method retrieves a number of userIDs from the
   * collection database for the record identified by the given recordID.
   * @param recordID the record ID, for which you will get all user IDs
   * @return ArrayLlist If records for the given recordID are found,
   *         the userIDs are added to an ArrayLlist, and returned. If
   *         the ArrayList is empty, there was no users found for the
   *         given recordID
   * @throws Exception if something goes wrong when conncting to the
   *         database, the Exception is written to the errorlog,
   *         and an exception is thrown, indicating failure.
   */
  protected ArrayList getUserID(String recordID) throws Exception {
    ArrayList list = new ArrayList();
    ResourceSet resultset = getCollection(RECORDID, recordID);
    ResourceIterator resIt = resultset.getIterator();
    while (resIt.hasMoreResources()) {
      Resource res = resIt.nextResource();
      CollectionRecord rec = new CollectionRecord();
      CollectionXmlParser xp =
          new CollectionXmlParser( (String) res.getContent());
      rec = xp.parse( (String) res.getContent());
      list.add(rec.getRecordID());
    }
    return list;
  }
}
```

G.10 DPC.Interface.CollectionNotFoundException

```
package DPC.Interface;

/**
 * <p>Title: PDC WebService</p>
 * <p>Description: DPC Servlet/WebService</p>
 * <p>This Excpetion indicates that the collection was not
 * found in the database</p>
 * @author Sverre Joki
 * @version 1.0
 */

public class CollectionNotFoundException
        extends Exception {

    /***
     * Creates an CollectionNotFoundException that has no message
     */
    public CollectionNotFoundException() {
        super();
    }

    /***
     * Creates an CollectionNotFoundException that has a message.
     * The mesasge is given in the string parameter
     * @param s the message string
     */
    public CollectionNotFoundException(String s) {
        super(s);
    }

}
```

G.11 DPC.Interface.DatabaseInterface

```
package DPC.Interface;

import DPC.DB.*;
import DPC.Interface.*;
import DPC.xml.*;
import DPC.*;
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import org.xmldb.api.*;
import java.util.GregorianCalendar;

/**
 * This class isthe superclass for all the database interfaces. This
 * class has data about where det databases are located, and what
 * driver the databases needs. This informastion is available in the
 * driver and dburi attributes.
```

```
 *
 * @author Sverre Joki
 */
public class DataBaseInterface {

  /***
   * This is the attribute specifying what database driver that is
   * needed to connect to the databases. This spiecifies if it
   * is an Exist or XIndice database
   */
  protected String driver = "org.exist.xmldb.DatabaseImpl";

  /***
   * This attribute specifies the default url to the databases
   */
  protected String dburi =
      "xmldb:exist://fenris.idi.ntnu.no:7180/exist/db/dpc";

  public GregorianCalendar cal = new GregorianCalendar();

  //The database colletion
  Collection col = null;

  public DataBaseInterface() {}

  /***
   * This method initializes the connetcion to the database using the
   * parameteres specified
   * as <code>driver</code> and <code>collectionPath</code>
   * @param dbDriver the driver class used as database connection
   *                 driver
   * @param collectionPath the path to the correct collection to
   *                       use for the MetadataDatabase
   * @throws Exception If anything goes wrong during initialization,
   *                   the contructor writes the exception to the
   *                   error log, and throws an Exception indicating
   *                   failure.
   */

  public void init(String driver, String collectionPath) throws
      Exception {
    //Setting up the connection to the database
    try {
      System.err.println("trying to init this interface: " +
                         this.getClass().getName());
      System.err.println("We are using this driver:" + driver +
                         "\nand this url:" + collectionPath);
      Class c = Class.forName(driver);
      Database database = (Database) c.newInstance();
      DatabaseManager.registerDatabase(database);
      database.setProperty("create-database", "true");
      col = DatabaseManager.getCollection(collectionPath);
      col.setProperty("pretty", "true");
      col.setProperty("encoding", "ISO-8859-1");

      System.err.println("finished with init of this interface: " +
```

```
                        this.getClass().getName());

    }
    catch (XMLDBException e) {
      System.err.println("XML:DB Exception occured " + e.toString());
      e.printStackTrace(System.err);
      throw new Exception("XML:DB Exception occured " + e.toString());
    }
    catch (Exception e) {
      System.err.println("An exception occured " + e.toString());
      e.printStackTrace(System.err);
      throw new Exception("An exception occured " + e.toString());
    }
  }

  /***
   * This method is used to send a quesry to a collection.
   * @param xpath the xpath expression to send
   * @return ResourceSet the resourceSet returned form the database
   */
  protected ResourceSet query(String xpath) throws XMLDBException {
    System.err.println(this.getClass().getName() + ".query():"
                       + cal.getTime().toString() + ":START");
    System.err.println(
        "Dette er xpathen som blir sendt som query til exist:"
        + xpath);
    //send query
    XPathQueryService service =
        (XPathQueryService) col.getService("XPathQueryService", "1.0");
    //get response
    ResourceSet resultSet = service.query(xpath);
    System.err.println(this.getClass().getName() + ".query():"
                       + cal.getTime().toString() + ":END");
    return resultSet;
  }
}
```

G.12 DPC.Interface.MetadataDBInterface

```
package DPC.Interface;

import DPC.DB.*;
import DPC.Interface.*;
import DPC.xml.*;
import DPC.*;
import javax.xml.transform.*;
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import org.xmldb.api.*;
import java.util.ArrayList;

/**
 * <p>
 * This class is the communication class that connects the
 * SearchInterface to the Metadatabase containing all personal
 * metadata added by the users.
 * </p>
 * @author Sverre Joki
 */
public class MetadataDBInterface
    extends DataBaseInterface {

  /***
   * The path to the collection in the database
   * to use for the UserDBInterface
   */
  private String collection = "/localmetadata";

  /***
   * The path to the collection to use for the database
   */
  private String collectionPath;

  /***
   * This contructor creates a standard MetadataDBInterface object, and
   * initializes the connetcion to the database using the attributes
   * specified in the super-class.
   * @throws Exception If anything goes wrong during initialization, the
   *                    contructor writes the exception to the error
   *                   log, and throws an Exception indication a failure.
   */
  public MetadataDBInterface() throws Exception {
    System.err.println(cal.getTime().toString() + ":"
                       + this.getClass().getName());
    try {
      collectionPath = super.dburi + collection;
      init(driver, collectionPath);
    }
    catch (Exception e) {
      System.err.println("init error occured " + e.toString());
      e.printStackTrace(System.err);
      throw new Exception("Error initialzing metadata DB");
```

```
    }
  }

  /***
   * This contructor creates a standard MetadataDBInterface object, and
   * initializes the connetcion to the database using the parameteres
   * specified as <code>dbDriver</code> and <code>collectionPath</code>
   * @param dbDriver the driver class used as database connection driver
   * @param collectionPath the path to the correct collection to use for
   *                               the MCollection Database
   * @throws Exception If anything goes wrong during initialization, the
   *                    contructor writes the exception to the error
   *                   log, and throws an Exception indicating a failure.
   */
  public MetadataDBInterface(String dbDriver, String collectionPath)
      throws Exception {
    System.err.println(cal.getTime().toString() + ":"
                       + this.getClass().getName());
    driver = dbDriver;
    this.collectionPath = collectionPath;
    try {
      init(driver, collectionPath);
    }
    catch (Exception e) {
      System.err.println("init error occured " + e.toString());
      e.printStackTrace(System.err);
      throw new Exception("Error initialzing metadata DB");
    }

  }

  /***
   * This method initializes the connetcion to the database using the
   * parameteres specified
   * as <code>dbDriver</code> and <code>collectionPath</code>.
   * @param dbDriver the driver class used as database connection driver
   * @param collectionPath the path to the correct collection to use for
   *                               the MetadataDatabase
   * @throws XMLDBException If anything goes wrong during initialization,
   *                    the contructor writes the exception to the error
   *                    log, and throws an XMLDBException indicating a
   *                    failure.
   * @deprecated use super.init instead
   */
  public void init(String driver, String collectionPath) throws
      XMLDBException {
    //Setting up the connection to the database
    try {
      System.err.println("trying to init this interface: "
                         + this.getClass().getName());
      Class c = Class.forName(driver);
      Database database = (Database) c.newInstance();
      DatabaseManager.registerDatabase(database);
      col = DatabaseManager.getCollection(collectionPath);
    }
    catch (XMLDBException e) {
```

```
      System.err.println("XML:DB Exception occured " + e.toString());
      e.printStackTrace(System.err);
      throw new XMLDBException();
    }
    catch (Exception e) {
      System.err.println("An exception occured" + e.toString());
      e.printStackTrace(System.err);
      throw new XMLDBException();
    }
  }


  /**
   * This metod returns an object containg a metadatarecord matching the
   * input MetadataPostID(mdpid). If more than one object is found,
   * this is an error, and the first will be returned.
   * @param mdpid a unique identifier identifying a record in this
   *              database
   */
  public LocalMetadataRecord getMetadataByMDPID(String mdpid)
      throws Exception {
    LocalMetadataRecord lm = new LocalMetadataRecord();
    ResourceSet resultSet = getMetadata("RecordID", mdpid);
    //Generate LocalMetadataRecord object from the resultSet
    ResourceIterator resIt = resultSet.getIterator();
    if (resIt.hasMoreResources()) {
      Resource res = resIt.nextResource();
      XmlToLocalMetadataRecordParser xp =
          new XmlToLocalMetadataRecordParser();
      lm = xp.parse((String)res.getContent());
    }
    //return it!
    return lm;
  }

  /**
   * This metod returns an object containg a metadatarecord matching the
   * input MetadataPostID(mdpid). If more than one object is found,
   * this is an error, and the first will be returned.
   * @param mdpid a unique identifier identifying a record in this
   *             database
   * @return LocalMetadataRecord a metadatarecord
   */
  public ArrayList getMetadataByMDPID(String mdpid) throws
      MetadataNotFoundException {
    System.err.println(
        this.getClass().getName() + ".getMetadataByMDPID(), " +
        cal.getTime().toString() +
        "Start"
        );
    //the return object
    ArrayList v = new ArrayList();
    try {

      System.err.println(
          "Dette er parametrene som blir brukt til å lage xPath query:");
```

```
    System.err.println("Value:" + mdpid);
    String param = mdpid.substring(mdpid.lastIndexOf("objectid="),
        mdpid.length());

    //create XPATH query from parameter
    String xpath = "//rdf:Description[@rdf:about='" + mdpid + "']";
    ResourceSet resultSet = query(xpath);
    //Genreate a metadataobject from the resultSet object
    ResourceIterator ri = resultSet.getIterator();
    //iterate through the reslutset
    while (ri.hasMoreResources()) {
      //generate a parser to parse the xml string
      XMLResource res = (XMLResource) ri.nextResource();
      //add metadataobject to vector
      v.add(res);
    }
    System.err.println(
        this.getClass().getName() + ".getMetadataByMDPID(), " +
        cal.getTime().toString() +
        "Done"
        );

    return v;
  }
  catch (XMLDBException e) {
    throw new MetadataNotFoundException(e.toString());
  }
}

/**
 * This Method returns an ArrayListcontaing MetadataRecords
 * mathing the input subject words. The subjectwords are to be
 * supplied in an String-array.
 * @param subjects the list of subject to search for in the
 *                          database this object is connectied to
 * @return ArrayList a list of MetadataRecords matching the
 *                   given subjectlist
 */
public ArrayList getMetadataBySubject(String[] subjects) throws
    MetadataNotFoundException {
  try {
    //the return object
    ArrayList v = new ArrayList();
    //iterate trough resourcesets
    int i = 0;
    while (i < subjects.length) {
      //get a resourceset from the db
      ResourceSet resultSet = query(
          "//rdf:Description[dc:subject&='" +
          subjects[i] + "']");
      //Genreate a metadataobject from the resultSet object
      ResourceIterator ri = resultSet.getIterator();
      //iterate through the reslutset
      while (ri.hasMoreResources()) {
        XMLResource res = (XMLResource) ri.nextResource();
        //add metadataobject to vector
```

```
          v.add(res);
        }
        i++;
      }
      return v;
    }
    catch (XMLDBException e) {
      throw new MetadataNotFoundException(e.toString());
    }
  }

  /**
   * This Methos returns an ArrayList containg MetadataRecords
   * mathing the input UserId.
   * @param uid the userID to fetch metadatarecords for
   * @deprecated Use the method in searchInterface instead!!
   */
  public ResourceSet getMetadataXmlByUid(String uid) throws Exception {
    //get a resourceset from the db and return it.
    return getMetadata("UserID", uid);
  }

  /**
   * This Method returns an ArrayList containg MetadataRecords
   * matching the input uid.
   * @deprecated Use the method in searchInterface instead!!
   */
  public ArrayList getMetadataByUid(String uid) throws Exception {
    System.err.println(
        this.getClass().getName() + ".getMetadataByUid(), " +
        cal.getTime().toString() +
        ": Start"
        );
    try {
      //the return object
      ArrayList v = new ArrayList();

      System.err.println(
          "Dette er parametrene som blir brukt til å lage xPath query:");
      System.err.println("Value:" + uid);

      //create XPATH query from userName
      String xpath = "//rdf:Description[dc:relation='" + uid + "']";
      ResourceSet resultSet = query(xpath);
      //Genreate a metadataobject from the resultSet object
      ResourceIterator ri = resultSet.getIterator();
      //iterate through the reslutset
      while (ri.hasMoreResources()) {
        //generate a parser to parse the xml string
        XMLResource res = (XMLResource) ri.nextResource();
        //add metadataobject to vector
        v.add(res);
      }
      System.err.println(
          this.getClass().getName() + ".getMetadataByUid(), " +
          cal.getTime().toString() +
```

```
        "Done"
        );

    return v;
  }
  catch (XMLDBException e) {
    throw new MetadataNotFoundException(e.toString());
  }
}

/***
 * This method is used to fetch data from the database based on a
 * parameter and a value. This method connects to the database using
 * the driver and path specified in the attribuets in this class.
 * @param param the database-field to search for data in
 * @param value the value to search for in the specified parameter
 * @return ResourceSet the ResourceSet object containing the found
 *                     records
 * @throws XMLDBException if anything goes wrong during the connection
 *                        to the database
 */
private ResourceSet getMetadata(String param, String value) throws
    XMLDBException {
  System.err.println(cal.getTime().toString() + ":"
                     + this.getClass().getName() +
                     ".getUser");
  System.err.println(
      "Dette er parametrene som blir brukt til å lage xPath query:");
  System.err.println("Param:" + param + ", Value:" + value);

  //create XPATH query from userName
  String xpath = "//record[" + param + "='" + value + "']";
  //send query
  XPathQueryService service =
      (XPathQueryService) col.getService("XPathQueryService", "1.0");
  //get response
  ResourceSet resultSet = service.query(xpath);
  return resultSet;
}

/***
 * This method is used to add a xml-record to the database. This method
 * accepts a String object that contains the xml-document. This document
 * will be added to the database.
 * @param xmlrecord the xml-record that will be added to the database
 * @return boolean indicating sucess
 * @throws Exception if anything unexpected happend, an Exception
 *                   will be thrown.
 */
public boolean addMetadata(String xmlRecord) throws Exception {
  try {
    //get a db connection
    //create a userDataobject
    XMLResource document = (XMLResource) col.createResource(null,
        "XMLResource");
    //create a xml document from the userdata object
```

```
      document.setContent(xmlRecord);
      col.storeResource(document);
    }
    catch (XMLDBException e) {
      e.printStackTrace(System.err);
      throw new Exception(
          "Adding metadata didn't go so well.");
    }

    return false;
  }

}
```

G.13 DPC.Interface.MetadataNotFoundException

```
package DPC.Interface;

/**
 * <p>Title: PDC WebService</p>
 * <p>Description: DPC Servlet/WebService</p>
 * <p>This Excpetion indicates that the metadatarecord was
 * not found in the database</p>
 * @author Sverre Joki
 * @version 1.0
 */

public class MetadataNotFoundException extends Exception{

    /***
     * Creates an UserNotFoundException that has no message
     */
    public MetadataNotFoundException() {
        super();
    }

    /***
     * Creates an CollectionNotFoundException that has a message.
     * The mesasge is given in the string parameter
     * @param s the message string
     */
    public MetadataNotFoundException(String s) {
        super(s);
    }

}
```

G.14 DPC.Interface.SearchInterface

```
package DPC.Interface;

import DPC.*;
import DPC.Interface.UserDBInterface;
import DPC.DB.*;
import DPC.xml.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import javax.xml.parsers.*;
import org.xmldb.api.modules.*;
import org.dom4j.Document;
import org.dom4j.DocumentException;
import org.dom4j.io.SAXReader;
import org.dom4j.DocumentHelper;
import org.dom4j.Element;
import java.util.*;
import java.io.*;

/**
 * <p>Title: PDC WebService</p>
 * <p>Description: DPC Servlet/WebService</p>
 * <p>This class is used to perform searches in the DPC system.
 * This class har connections to the 3 different databases in the
 * DPC system, and builds
 * records that the methods specify.</p>
 * @author Sverre Joki
 * @version 1.0
 */
public class SearchInterface
    extends DataBaseInterface {

  /***
   * This is the attribute that holds this class' name. Used for
   * debugging purposes.
   */
  private static String cName = "SearchInterface";

  /***
   * This is the UserDbInterfae connection class
   */
  private UserDBInterface theUserDBInterface;

  /***
   * This is the object that conntect the SearchInterface to the
   * metadata database interface
   */
  private MetadataDBInterface theMetadataDBInterface;

  /***
   * This is the object that conntect the SearchInterface to the
   * collections database interface
   */
  private CollectionDBInterface theCollectionDBInterface;
```

```
/***
 * This contructor creates a SearchInterface object using de
 * properties specified in the parameters. The properties
 * needed are:
 * <ul><li>
 * collection.db.driver
 * </li><li>
 * collection.db.url
 * </li><li>
 * localmetadata.db.driver
 * </li><li>
 * localmetadata.db.url
 * </li><li>
 * users.db.driver
 * </li><li>
 * users.db.url
 * </li></ul>
 *
 * @param p proterties that has the correct values
 * @throws Exception if anything goes wrong
 */
public SearchInterface(Properties p) throws Exception {
  try {
    System.err.println("SearchInterface init");

    String driverClass = p.getProperty("collection.db.driver",
        super.driver).
        trim();
    String dbUrl = p.getProperty("collection.db.url", super.dburi).
        trim();
    System.err.println("CollectionDB init; d:" + driverClass
                       + ", u:"
                       + dbUrl);
    theCollectionDBInterface = new CollectionDBInterface(
        driverClass,
        dbUrl);
    System.err.println("CollectionDB init ok\n");

    driverClass = p.getProperty("localmetadata.db.driver",
        super.driver);
    dbUrl = p.getProperty("localmetadata.db.url", super.dburi);
    System.err.println("MetadataDB init; d:" + driverClass + ", u:"
                       + dbUrl);
    theMetadataDBInterface = new MetadataDBInterface(driverClass,
        dbUrl);
    System.err.println("metadataDB init ok\n");

    driverClass = p.getProperty("users.db.driver", super.driver);
    dbUrl = p.getProperty("users.db.url", super.dburi);
    System.err.println("UserDB init; d:" + driverClass + ", u:"
                       + dbUrl);
    theUserDBInterface = new UserDBInterface(driverClass, dbUrl);
    System.err.println("userDB init ok\n");

  }
```

```
  catch (Exception e) {
    try {
      //Init av interface med parametre fra web.xml gikk ikke,
      //Prøver å initiere med default verdier som om det ble kalt
      //en tom contructor
      System.err.println(
          "SearchInterface init without web.xml parameters");
      theCollectionDBInterface = new CollectionDBInterface();
      System.err.println("CollectionDB init ok");
      theMetadataDBInterface = new MetadataDBInterface();
      System.err.println("metadataDB init ok");
      theUserDBInterface = new UserDBInterface();
      System.err.println("userDB init ok");
    }
    catch (Exception e1) {
      e1.printStackTrace(System.err);
      throw new Exception(
          "Error when initializing database interfaces: " +
          e1.toString());
    }
  }
}

/***
 * This contructor creates a SearchInterface object using the
 * default contructors of the separate interfaces
 * @throws Exception if anything goes wrong
 */
public SearchInterface() throws Exception {
  try {
    System.err.println("SearchInterface init");
    theCollectionDBInterface = new CollectionDBInterface();
    System.err.println("CollectionDB init ok");
    theMetadataDBInterface = new MetadataDBInterface();
    System.err.println("metadataDB init ok");
    theUserDBInterface = new UserDBInterface();
    System.err.println("userDB init ok");
  }
  catch (Exception e) {
    e.printStackTrace(System.err);
    throw new Exception(
        "Error when initializing database interfaces: " +
        e.toString());
  }

}

///////////////////
// METADATA METHODS
///////////////////

/**
 * Creates a text document object matching the MetadataPostID input.
 * @param mdpid the metadata record id
 * @return Document the xml document containing the record matching
 *                  the given input
```

```
 * @exception
 */
public String getMetadata(String mdpid) throws Exception {
  System.err.println(
      this.getClass().getName() +
      ".getMatadata() ["
      + cal.getTime().toString() +
      "]:START");
  try {
    ArrayList list = theMetadataDBInterface.getMetadataByMDPID(
        mdpid);
    Iterator it = list.iterator();
    StringBuffer sbuf = new StringBuffer();
    try {
      //iterate through all document object returned
      while (it.hasNext()) {
        XMLResource xr = (XMLResource) it.next();
        String resourceString = (String) xr.getContent();
        System.err.println(cal.getTime().toString() + ": " + cName
                           + ":" +
                           resourceString);
        sbuf.append(resourceString);
      } //while
      System.err.println(this.getClass().getName()
                         + ".getMEtadata(),[" +
                         cal.getTime().toString() + "]:END");
      return sbuf.toString();
    }
    catch (Exception e) {
      System.err.println(this.getClass().getName()
                         + ".getMtadata(),[" +
                         cal.getTime().toString() +
                         "]:Error parsing xml from database");
      throw new Exception("Error parsing xml from database");
    }
  }
  catch (Exception e) {
    System.err.println(this.getClass().getName()
                       + ".getMtadata(),[" +
                       cal.getTime().toString() +
                       "]:Error getting xml from database");
    throw new Exception("Error getting xml from database");
  }
}

/**
 * Creates a Document object containing all records matching the
 * userid input.
 * @param uid UserId to the user in question
 * @return Document containing all records this user has registered
 *                   in the metadata db
 * @exception
 */
public String getMetadataUid(String uid) throws Exception {
  System.err.println(
      this.getClass().getName() + ".getMetadataUid(), " +
```

```
        cal.getTime().toString() +
        "Start"
        );

    //get collectionrecords based on the uid
    ArrayList list = theMetadataDBInterface.getMetadataByUid(uid);
    Iterator it = list.iterator();
    StringBuffer sbuf = new StringBuffer();
    try {
      while (it.hasNext()) {
        XMLResource res = (XMLResource) it.next();
        String resourceString = (String) res.getContent();
        System.err.println(cal.getTime().toString() + ": " + cName
                           + ":" +
                           resourceString);
        sbuf.append(resourceString);
      }
      System.err.println(
          this.getClass().getName() + ".getMetadataUid(), " +
          cal.getTime().toString() +
          "Done with this result:" + sbuf.toString()
          );

      return sbuf.toString();
    }
    catch (Exception e) {
      throw new Exception("");
    }
  }

  /**
   * This method returns a Document object based on a
   * position category
   * @param positionCategory
   * @return Document containing the matching user with the given
   *                   positions
   * @exception
   */
  public Document getUsersByPosition(String positionCategory) throws
      Exception {
    ArrayList list = theUserDBInterface.getUsersBy(theUserDBInterface.
        POSITION,
        positionCategory);
    return convertToDocument(list);
  }

  /**
   * This method returns a Document object based on a
   * institution
   * @param String institution
   * @return Document containing the matchin users in the given
   *         institution
   * @exception
   */
  public Document getUsersInstitution(String institution) throws
      Exception {
```

```
    ArrayList list = null;
    list = theUserDBInterface.getUsersBy(theUserDBInterface.
        INSTITUTION,
        institution);
    return convertToDocument(list);
  }

  /**
   * This method returns an ArrayList containing all matching users
   * in the UserDB. The list contains UserData objects.
   * @param String
   * @return ArrayList of users matching the input userName
   * @exception
   * @roseuid 3D5E692B0179
   */
  public Document getUsersByName(String userName) throws Exception {
    ArrayList list = theUserDBInterface
        .getUsersBy(theUserDBInterface.NAME, userName);

    return convertToDocument(list);
  }


  /**
   * This method gets the userData for a given user
   * @param String uid the userId of the person in question
   * @return UserData the userData object contianing the requested
   *                  userInfo
   * @exception
   */
  public String getUserData(String uid) throws Exception {
    System.err.println(
        this.getClass().getName() + ".getUserData, " +
        cal.getTime().toString() +
        "Start"
        );
    try {
      //generate UserData object from theUserDBInterface
      System.err.println(cal.getTime().toString() +
                         ": calling userDB interface");
      XMLResource xr = theUserDBInterface.getUserDataByUid(uid);
      System.err.println(cal.getTime().toString() + ": " + cName
                         + ":" +
                         (String) xr.getContent());

      String s = (String) xr.getContent();
      return s;
    }
    catch (Exception e) {
      System.err.println(cal.getTime().toString() + ": "
                         + cName + " throwing exception:"
                         + e.toString());
      e.printStackTrace(System.err);
      throw new Exception("No such user found");
    }
  }
```

```
/**
 * this method retrieves the bookshelf to the user with the given
 * userId. The returnValue is a ArrayList object that can be
 * converted later. The arrayList contains  a userata document from
 * the database and all metadataobjects that
 * @param userID the wanter user id
 * @return String xml record
 * @throws Exception in case of errors
 */
public String getBookshelf(String userID) throws Exception {
  System.err.println(cal.getTime().toString() + ": "
                   + this.getClass().getName()
                   + ".getBookShelf, START");

  //the return buffer
  StringBuffer retList = new StringBuffer();

  try {
    //for the collection handling
    Document document = null;
    //get the user
    XMLResource user = theUserDBInterface.getUserDataByUid(userID);
    System.err.println(cal.getTime().toString() + "cName:\n" +
                       (String) user.getContent()
                       + "Record end\n\n");
    String s1 = (String) user.getContent();
    System.err.println("get user content as String:" + s1);
    retList.append(s1);
    System.err.println("appended to SB");

    //Get collection for this user
    XMLResource xRes = theCollectionDBInterface.getCollectionUser(
        userID);
    System.err.println("Got collection");
    String collection = (String) xRes.getContent();
    System.err.println("got col. as string:" + collection);
    Document doc = DocumentHelper.parseText(collection);
    System.err.println("Created Document from col");
    Element root = doc.getRootElement();
    System.err.println("got root element");
    Iterator it = root.elementIterator();
    //først henter vi ut alle elementer fra "samlingen"
    while (it.hasNext()) {
      System.err.println("taverserer document");
      Element element = (Element) it.next();
      System.err.println("Henter elementet ut fra Document'et:" +
                         element.getName());
      if (element.getName().equals("recordid")) {
        System.err.println("fant recordid element");
        //alle elem. i samling som peker på en metadatapost henter vi ut
        String recordid = element.getTextTrim();
        System.err.println("Fant det som en string:" + recordid +
                           "\n henter metadata...");
        try {
          //get the metdatarecords for this record
```

```
          System.err.println(
              "begynner å hente ut metadata for posten");
          ArrayList records = theMetadataDBInterface.
              getMetadataByMDPID(
              recordid);
          it = records.iterator();
          System.err.println("Traverserer poster av metadata:" +
              element.getName());
          while (it.hasNext()) {
            System.err.println("Travererer metadataposter");
            XMLResource rec = (XMLResource) it.next();
            String s2 = (String) rec.getContent();
            System.err.println(
                "Dette er metadataposten jeg fant:" + s2);
            retList.append(s2);
          } //while
        }
        catch (MetadataNotFoundException e) {
          System.err.println(cal.getTime().toString() + ": "
              + cName + " throwing exception:"
              + e.toString());
          e.printStackTrace(System.err);
        }
      } //if element name equals recordid
    } //while getting metadata records
    System.err.println("Dette er det som sendes tilbake:"
                       + retList.toString());
    System.err.println(cal.getTime().toString() + ": " +
                       this.getClass().getName()
                       + ".getBookShelf, END");
  }
  catch (CollectionNotFoundException e) {
    System.err.println(this.getClass().getName()
                       + ".getBookShelf() ["
                       + cal.getTime().toString() +
                       "]: No collection found for this user");
    retList.append("Collection Not Found");

  }
  catch (UserNotFoundException e) {
    System.err.println(this.getClass().getName()
                       + ".getBookShelf() ["
                       + cal.getTime().toString() +
                       "]: Error parsing xml from database");
    retList.append("User Not Found");
  }
  finally {
    System.err.println(cal.getTime().toString() + ": " +
                       this.getClass().getName()
                       + ".getBookShelf, finally clause");

    System.err.println("Dette er det som sendes tilbake:"
                       + retList.toString());

    System.err.println(cal.getTime().toString() + ": " +
                       this.getClass().getName()
```

```
                      + ".getBookShelf, END");

    return retList.toString();
  }
}

/***
 * This method adds a metadata xml record to the metadata database
 * @param xmlRecord the metadata record
 * @return boolean intidaitng success or failure
 */
public boolean addMetadata(String xmlRecord) {
  //first find out if this record already exist
  //then add it to the database (if needed!)
  try {
    return this.theMetadataDBInterface.addMetadata(xmlRecord);
  }
  catch (Exception e) {
    e.printStackTrace(System.err);
    return false;
  }
}

/***
 * This method add a user wrapped in an xmlrecord to the database.
 * @param xmlRecord the userdata record
 * @return boolean intidaitng success or failure
 */
public boolean addUser(String xmlRecord) {
  try {
    return this.theUserDBInterface.addUser(xmlRecord);
  }
  catch (Exception e) {
    e.printStackTrace(System.err);
    return false;
  }
}

/***
 * Add personal metadata to the collections database.
 * @param String the xml record.
 * @return boolean indication success
 */
public boolean addPersonalMetadata(String xmlRecord) {
  CollectionRecord cr = new CollectionRecord();
  CollectionXmlParser parser = new CollectionXmlParser(xmlRecord);
  cr = parser.getcollectionRecord();
  return theCollectionDBInterface.addCollectionRecord(cr);
}

/***
 * This method converts a ArrayList containng record into a Document
 * object.
 * @param list the ArrayList
 * @return Docuemnt the document object
 * @throws Exception in case of errors
```

```
   */
  private Document convertToDocument(ArrayList list) throws Exception {
    Iterator it = list.iterator();
    try {
      DocumentBuilderFactory fac = DocumentBuilderFactory.newInstance();
      DocumentBuilder docbuild = fac.newDocumentBuilder();
      Document doc = null;
      while (it.hasNext()) {
        XMLResource res = (XMLResource) it.next();
        String text = (String) res.getContent();

        System.err.println(cal.getTime().toString() + ": " + cName
                           + ":" +
                           text);

        Document document = DocumentHelper.parseText(text);
      }
      return doc;
    }
    catch (Exception e) {
      throw new Exception("Could not convert arrayList to Document");
    }
  }
}
```

G.15 DPC.Interface.UserDBInterface

```
package DPC.Interface;

import DPC.DB.*;
import DPC.Interface.*;
import DPC.xml.*;
import DPC.*;
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import org.xmldb.api.*;
import java.util.ArrayList;

/**
 * <p>This class/object is the gateway to the userdatabase. This
 * class is created to allow for easy switching of databases in the
 * dpc servlet
 * <p>
 * This interface "contains" a normal databaseconnection to the
 * database server but this can be switched eith another collection.
 * <p>
 * UserDBInterface also has methos that the DPCServlet needs to
 * contanct the DataBase in question.
 *
 * @author Sverre Joki
 */
public class UserDBInterface
    extends DataBaseInterface {
```

```
//These variables is used to search in a database

/***
 * String constant identifiying this parameter in the SOAP request
 */
public static String USER_ID = "UserID";

/***
 * String constant identifiying this parameter in the SOAP request
 */
public static String INSTITUTION = "Institution";

/***
 * String constant identifiying this parameter in the SOAP request
 */
public static String POSITION = "Position";

/***
 * String constant identifiying this parameter in the SOAP request
 */
public static String NAME = "Name";

/***
 * The database type to use. xindice or exist
 */
String databaseType = "exist";

Collection udbCol = null;

/***
 * The path to the collection in the database
 * to use for the UserDBInterface
 */
private String collection = "/users";

/***
 * The path to the collection to use for the database
 */
private String collectionPath;

/***
 * This constructor creates a new UserDBInterface instance, and
 * initializes it with the attributes specified in the super class.
 * @throws Exception if initializatoin goes wrong, or something else
 *                   happens.
 */
public UserDBInterface() throws Exception {
  System.err.println(cal.getTime().toString() + ":"
                    + this.getClass().getName());
  try {
    init(super.driver, super.dburi + collection);
  }
  catch (Exception e) {
    System.err.println("init error occured " + e.toString());
    e.printStackTrace(System.err);
```

```
      throw new Exception("Error initialzing metadata DB");
    }
  }

  /***
   * This constructor creates a new UserDBInterface instance, and
   * initializes it with the data specified in the parameters.
   * @param driver the driver class to use when initalizing the connection
   *               to the database
   * @param collectionPath the path to the collection in the database.
   * @throws Exception if initializatoin goes wrong, or something else
   *                   happens.
   */
  public UserDBInterface(String driver, String collectionPath) throws
      Exception {
    System.err.println(cal.getTime().toString() + ":"
                       + this.getClass().getName());
    try {
      init(driver, collectionPath);
    }
    catch (Exception e) {
      System.err.println("init error occured " + e.toString());
      e.printStackTrace(System.err);
      throw new Exception("Error initialzing metadata DB");
    }
  }
  /***
   * This method initializes the connetcion to the database using the
   * parameteres specified
   * as <code>dbDriver</code> and <code>collectionPath</code>.
   * @param dbDriver the driver class used as database connection driver
   * @param collectionPath the path to the correct collection to use for
   *                                 the User DB
   * @throws XMLDBException If anything goes wrong during initialization,
   *                   the contructor writes the exception to the error
   *                   log, and throws an XMLDBException indicating a
   *                   failure.
   * @deprecated use super.init instead
   */
  public void init(String driver, String collectionPath) throws
      Exception {
    //Setting up the connection to the database
    try {
      System.err.println("trying to init this interface: "
                         + this.getClass().getName());
      Class c = Class.forName(driver);
      Database database = (Database) c.newInstance();
      DatabaseManager.registerDatabase(database);
      udbCol = DatabaseManager.getCollection(collectionPath);
    }
    catch (XMLDBException e) {
      System.err.println("XML:DB Exception occured " + e.toString());
      e.printStackTrace(System.err);
      throw new XMLDBException();
    }
    catch (Exception e) {
```

```
      System.err.println("An exception occured");
      e.printStackTrace(System.err);
      throw new Exception();
    }
  }

  /**
   * Searches the userdatabase for users with a corresponding value. If
   * the user is found it returns a ArrayList containing all XMLResource
   * objects matching the criteria. If the searchparameter is not
   * found for the given searchType, it throws an exception. This is
   * only to tell the calling object that an user was  not found.
   * It will throw an exception in any error situations.
   * @param searchType the type of search to perform.
   * @return ArrayList containing the user data from the search
   * @throws Exception if something goes wrong during the search
   */
  public ArrayList getUsersBy(String searchType,
                              String searchParameter) throws
      Exception {
    ArrayList list = null;
    //get resultset
    if (searchType.equals(this.USER_ID) ||
        searchType.equals(this.INSTITUTION) ||
        searchType.equals(this.NAME) ||
        searchType.equals(this.POSITION)) {

      ResourceSet resultSet = getUser(searchType, searchParameter);
      //generate list of userDataobjects
      ResourceIterator it = resultSet.getIterator();
      while (it.hasMoreResources()) {
        XMLResource res = (XMLResource) it.nextResource();
        list.add(res);
      }
      return list;
    }
    else {
      throw new Exception("User Not found");
    }

  }

  /**
   * Searches the userdatabase for users with a corresponding uid.
   * If the user is found it returns a UserData object. If the uid is
   * not found, it throws an exception. This is only to tell the
   * calling object that an user was not found.It will throw an
   * exception in any error situations.
   * @param uid the user id to use for the search
   * @return XMLResource the XMLResource object as returned by the
   *                     database
   * @throws UserNotFoundException if the user specified does not exist in
   *                               the database.
   */
  public XMLResource getUserDataByUid(String uid) throws
      UserNotFoundException {
```

```
    System.err.println(cal.getTime().toString()
                       + "UserDBInterface.getUserByUid(" + uid + ");");
    try {
      //get resultset
      ResourceSet resultSet = getUser(this.USER_ID, uid);
      //convert response to userDataobject
      ResourceIterator ri = resultSet.getIterator();
      System.err.println(
          "UserDBInterface.getUserdataByUid: iterating through
resultset");
      XMLResource res = (XMLResource) ri.nextResource();
      return res;
    }
    catch (Exception e) {
      throw new UserNotFoundException("User not found:" + e.toString());
    }
  }

  /**
   * Searches the userdatabase for users with a corresponding name. If
   * the user is found it returns a ArrayList containing all UserData
   * objects matching the criteria. If the name is not found, it throws
   * an exception. This is only to tell the calling object that an user
   * was not found.It will throw an exception in any error situations.
   * @param userName the user name to search for
   * @return ArrayList containing the userData rfam the database
   * @throws UserNotFoundException if ther are no users foung that matches
   *                               the given user name, this exception
   *                               will be thrown
   */
  public ArrayList getUsersByName(String userName) throws
      UserNotFoundException {
    try {
      ArrayList list = new ArrayList();
      //get resultset
      ResourceSet resultSet = getUser("Name", userName);
      //generate list of userDataobjects
      ResourceIterator it = resultSet.getIterator();
      while (it.hasMoreResources()) {
        XMLResource res = (XMLResource) it.nextResource();
        list.add(res);
      }
      return list;
    }
    catch (Exception e) { throw new UserNotFoundException(); }
  }


  /**
   * This method updates or adds the UserData object given as parameter
   * to the database
   * @param xmlRecord the userdata as an xml strig, to save in the
   *                  database.
   * @return boolean indicating a success or failure
   * @throws Exception if something goes wrong
   */
```

```
  public boolean addUser(String xmlRecord) throws Exception {
    try {
      //get a db connection
      //create a userDataobject
      XMLResource document = (XMLResource) udbCol.createResource(null,
          "XMLResource");
      //create a xml document from the userdata object
      document.setContent(xmlRecord);
      udbCol.storeResource(document);
    }
    catch (XMLDBException e) {
      e.printStackTrace(System.err);
      throw new Exception(
          "Adding userdata didn't go so well. Why....?");
    }
    return false;
  }

  /**
   * Defalut get method to get Userinfo from the UserDB. The parameter
   * specifies what type of data to search(userId, name etc.), and the
   * value specifies the value to match
   * @param param what type of data to search(userId, name etc.)
   * @param value the value to search for in the give databasefield
   *              specified by the param
   * @return ResourceSet a ResourceSet with the records from the database
   * @throws XMLDBException if something goes wrong during the retrieval
of
   *                        data from the database.
   */
  private ResourceSet getUser(String param, String value) throws
      XMLDBException {

    System.err.println(cal.getTime().toString() + ":"
                       + this.getClass().getName() + ".getUser");
    System.err.println(
        "Dette er parametrene som blir brukt til å lage xPath query:");
    System.err.println("Param:" + param + ", Value:" + value);
    //create XPATH query from userName
    String xpath = "//*[" + param + "='" + value + "']";
    System.err.println("Dette er queryet som går til databasen:"
                       + xpath);
    //send query
    XPathQueryService service = (XPathQueryService) col.getService(
        "XPathQueryService", "1.0");
    System.err.println("Satt opp XPathQuesryService");
    service.setProperty("pretty", "true");
    System.err.println("Satt property pretty");
    service.setProperty("encoding", "ISO-8859-1");
    //get response
    System.err.println("Satt encoding");
    ResourceSet resultSet = service.query(xpath);
    System.err.println("Dette er etter at quesry er sendt");
    if (resultSet != null)
      System.err.println("resultSet != null" + resultSet.toString());
    else
```

```
      System.err.println("resultSet == null");

    return resultSet;
  }
}
```

G.16 DPC.Interface.UserNotFoundException

```
package DPC.Interface;

/**
 * <p>Title: PDC WebService</p>
 * <p>Description: DPC Servlet/WebService</p>
 * <p>This Excpetion indicates that the user was not found in the
 * database</p>
 * @author Sverre Joki
 * @version 1.0
 */
public class UserNotFoundException
    extends Exception {

  /***
   * Creates an UserNotFoundException that has no message
   */
  public UserNotFoundException() {
    super();
  }

  /***
   * Creates an UserNotFoundException that has a message. The
   * mesasge is given in the string parameter
   * @param s the message string
   */
  public UserNotFoundException(String s) {
    super(s);
  }
}
```

G.17 DPC.xml.UserDataToXmlParser

```
package DPC.xml;

import DPC.DB.*;
import DPC.Interface.*;
import DPC.xml.*;
import DPC.*;
import javax.xml.soap.*;
import javax.xml.messaging.*;
import java.util.*;

/**
 * <p>Title: PDC WebService</p>
 * <p>Description: DPC Servlet/WebService</p>
 * This class is an utility class to parse user data. The class has
 * methods to convert a UserData object into a SOAPMessage object,
 * and convert and ArrayList of UserDataobject into a SOAPMessage.
 * @author Sverre Joki
 * @version 1.0
 */
public class UserDataToXmlParser {

  /***
   * Object used to create a SOAPMessage.
   */
  private MessageFactory mf;

  /***
   * A SOAPMessage that is returned inthe methods
   */
  private SOAPMessage msg;

  /***
   * Part of a SOAPMessage that is used when parsing the UserData
   * object, and creating the SOAPMessage
   */
  private SOAPPart sp;

  /***
   * Part of a SOAPMessage that is used when parsing the UserData
   * object, and creating the SOAPMessage
   */
  private SOAPEnvelope envelope;

  /***
   * Part of a SOAPMessage that is used when parsing the UserData
   * object, and creating the SOAPMessage
   */
  private SOAPHeader hdr;

  /***
   * Part of a SOAPMessage that is used when parsing the UserData
   * object, and creating the SOAPMessage
   */
```

```
  private SOAPBody bdy;

  /***
   * Part of a SOAPMessage that is used when parsing the UserData
   * object, and creating the SOAPMessage
   */
  private SOAPBodyElement bodyElement;

  /***
   * This constructor creates an empty UserDdataToXmlParser object.
   */
  public UserDataToXmlParser() {
  }

  /***
   * This method converts an group of UserData objects
   * into a SOAPMessage. The UserData object must be given
   * in a ArrayList object.
   * The SOAPMessage can be sendt to
   * a client useing the DPC system.
   * @param ArrayList the ArrayList that contains the UserData
   *         object that will be converted into a single SOAPMessage
   * @return SOAPMessage containing the SOAP formatted UserData
   * @throws SOAPException if some part of the convertion goes wrong,
   *                       a SOAPExceptoin with the message
   *                       "Error converting Userdata to SOAP"
   */
  public SOAPMessage convertToSOAP(ArrayList userDataArray) throws
      SOAPException {

    try {
      //--------------------------
      //Setting up the soap message
      //--------------------------

      // Create a message factory.
      mf = MessageFactory.newInstance();

      // Create a message from the message factory.
      msg = mf.createMessage();

      // Message creation takes care of creating the SOAPPart - a
      // required part of the message as per the SOAP 1.1
      // specification.
      sp = msg.getSOAPPart();

      // Retrieve the envelope from the soap part to start building
      // the soap message.
      envelope = sp.getEnvelope();

      // Create a soap header from the envelope.
      hdr = envelope.getHeader();

      // Create a soap body from the envelope.
      bdy = envelope.getBody();
```

```
    // Add a soap body element to the soap body
    bodyElement = bdy.addBodyElement(envelope.createName(
        "DPC_Return_Message"));

    //-------------------------
    //Populating soapmessage with userdata objects
    //-------------------------

    Iterator it = userDataArray.iterator();
    while (it.hasNext()) {
      UserData userData = (UserData) it.next();
      //Create a UserDataElement that contains all needed data
      createUserDataElement(userData);
    }
    return msg;
  }
  catch (SOAPException e) {
    throw new SOAPException("Error converting Userdata to SOAP");
  }
}

/**
 * This method genereates a SOAPMessage based on the Userdata
 * object given as input. The SOAPMessage contains only this user.
 * <p>
 * This method creates one SOAPMessage based on ONE userdata object
 * </p>
 * @param userData the UserData object that will be converted
 *                 into a single SOAPMessage
 * @return SOAPMessage containing the SOAP formatted UserData
 * @throws SOAPException if some part of the convertion goes wrong,
 *                       a SOAPExceptoin with the message
 *                       "Error converting Userdata to SOAP"
 */
public SOAPMessage convertToSOAP(UserData userData) throws
    SOAPException {

  try {
    // Create a message factory.
    mf = MessageFactory.newInstance();

    // Create a message from the message factory.
    msg = mf.createMessage();

    // Message creation takes care of creating the SOAPPart - a
    // required part of the message as per the SOAP 1.1
    // specification.
    sp = msg.getSOAPPart();

    // Retrieve the envelope from the soap part to start building
    // the soap message.
    envelope = sp.getEnvelope();

    // Create a soap header from the envelope.
    hdr = envelope.getHeader();
```

```
      // Create a soap body from the envelope.
      bdy = envelope.getBody();

      // Add a soap body element to the soap body
      bodyElement = bdy.addBodyElement(envelope.createName(
          "DPC_Return_Message"));

      //Create a UserDataElement that contains all needed data
      createUserDataElement(userData);
      return msg;
    }
    catch (SOAPException e) {
      throw new SOAPException("Error converting Userdata to SOAP");
    }

  }

  /***
   * This method is used to create a SOAPElement of a single
   * UserData object. This methos is used by the other methods of
   * this class to create user data elements to add to a SOAPMessage.
   * <p>
   * The method creates child elements like this structure:
   * <pre>
   * UserData
   *   - UserID
   *   - Name
   *   - StreetAdress
   *   - PostCode
   *   - City
   *   - Email
   *   - Position
   *   - Institution
   *   - Interest +
   *   - Descpription
   * </pre>
   * @param ud the UserData object to convert into a SOAPElement
   * @return SOAPElement the SOAPElement that is created of the
   *                     UserData object
   * @throws SOAPException if some of the operations on the
   *                       SOAPElement goes wrong
   */

  private SOAPElement createUserDataElement(UserData ud) throws
      SOAPException {

    //Add a userData element to the root element
    SOAPElement userdataElement =
        bodyElement.addChildElement(envelope.createName("UserData"));
    //add elements to the userdata element
    userdataElement.addChildElement(envelope.createName("UserID"))
        .addTextNode(ud.userID);
    userdataElement.addChildElement(envelope.createName("Name"))
        .addTextNode(ud.name);
    userdataElement.addChildElement(envelope.createName(
        "StreetAdress"))
```

```
        .addTextNode(ud.adress);
    userdataElement.addChildElement(envelope.createName("PostCode"))
        .addTextNode(ud.postCode);
    userdataElement.addChildElement(envelope.createName("City"))
        .addTextNode(ud.city);
    userdataElement.addChildElement(envelope.createName("Email"))
        .addTextNode(ud.email);
    userdataElement.addChildElement(envelope.createName("Position"))
        .addTextNode(ud.position);
    userdataElement.addChildElement(envelope.createName("Institution"))
        .addTextNode(ud.institution);
    ArrayList interests = ud.interests;
    Iterator intIt = interests.iterator();
    while (intIt.hasNext()) {
      String interestString = (String) intIt.next();
      userdataElement.addChildElement(envelope.createName("Interest"))
          .addTextNode(interestString);
    }
    userdataElement.addChildElement(envelope.createName("Description"))
        .addTextNode(ud.description);
    return userdataElement;
  }
}
```

G.18 DPC.xml.XmlToCollectionRecordParser

```
package DPC.xml;

import DPC.DB.*;
import DPC.Interface.*;
import DPC.xml.*;
import DPC.*;
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import java.io.StringReader;
import java.util.ArrayList;
import java.util.Vector;

/**
 * <p>Title: XmlToCollectionRecordParser</p>
 * <p>Description: XmlToCollectionRecordParser</p>
 * This class is a utility class to convert a xml document into
 * a CollectionRcord document. This CollectionRcord objcet is used
 * to handle the collection record internally in the DPC system.
 * @author Sverre Joki
 * @version 1.0
 */
public class XmlToCollectionRecordParser
    extends DefaultHandler {
```

```
/***
 * This attribute is used during the convertion of the xml structure
 * to store the text of the parsed xml document elements.
 */
private StringBuffer textBuffer;

/***
 * This attribute is used during the convertion of the xml structure
 * to keep the CollectionRecord object.
 */
private CollectionRecord cr;

/***
 * This attribute is used during the convertion of the xml structure.
 * recordID keeps the id for the record.
 */
private String recordID = null;

/***
 * This attribute is used during the convertion of the xml structure.
 * userId keeps the id for the user.
 */
private String userID = null;

/***
 * This attribute is used during the convertion of the xml
 * structure. subjectWords keeps the list of subjectword in
 * this collection record.
 */
private ArrayList subjectWords = null;

/***
 * This attribute is used during the convertion of the xml structure.
 * description keeps the description in the collection record
 */
private String description = null;

/***
 * This attribute is used during the convertion of the xml structure.
 * True of the investigated element contains a record id
 */
boolean isRecordID;

/***
 * This attribute is used during the convertion of the xml structure
 * True of the investigated element contains a subject word
 */
boolean isSubjectWord;

/***
 * This attribute is used during the convertion of the xml structure
 * True of the investigated elemen contains a description
 */
boolean isDescription;

/***
```

```
 * This attribute is used during the convertion of the xml structure
 * True of the investigated elemen contains a user id
 */
boolean isUserID;

/***
 * This constructor creates a empty XmlToCollectionRecordParser
 * object. This object is used to parse a collection record from
 * xml structure into an internal datastructure used by the
 * DPC system.
 */
public XmlToCollectionRecordParser() {
}

/***
 * This method is used to parse a collection record from xml structure
 * into an internal ldatastructure used by the DPC system. The xml
 * structure must be deliverd as a string object containig the xml
 * document.
 * @param xmlInput the String containing the xml document
 * @return CollectionRecord the new object created from the xml document
 * @throws SAXException if something goes wrong during parsing
 */
public CollectionRecord parse(String xmlInput) throws SAXException {

  // Use an instance of ourselves as the SAX event handler
  DefaultHandler handler = new XmlToCollectionRecordParser();

  // Use the default (non-validating) parser
  SAXParserFactory factory = SAXParserFactory.newInstance();
  try {

    // Parse the input
    SAXParser saxParser = factory.newSAXParser();
    //creates an InputSource from a StringReader created from xmlInput
    saxParser.parse(new InputSource(new StringReader(xmlInput)),
                    handler);
  }
  catch (Throwable t) {
    //Error occured in parsing xml. throw exception
    throw new SAXException("Error in parsing xml");
  }
  return cr;
}

//============================================================
// SAX DocumentHandler methods
//============================================================

/**
 * This method is performed when the SaxParser finds the start of the
 * xml document. This is only done once for a collection xml document.
 * The method instasiates a CollectionRecord object.
 * @return void
 * @throws SAXExcpetion if something unexpected happens
 */
```

```
  public void startDocument() throws SAXException {
    cr = new CollectionRecord();
    reset();
  }

  /***
   * This method is called when the SaxParser finds the end of the
   * xml document. This method adds the values found by the parser
   * to the CollectionRecord object.
   * @return void
   * @throws SAXExcpetion if something unexpected happens
   */
  public void endDocument() throws SAXException {
    //adding personalDescriptions to metadatarecord
    cr.setUserID(userID);
    cr.setRecordID(recordID);
    cr.setDescription(description);
    cr.setSubjectWords(subjectWords);
    reset();
  }

  /***
   * <p>
   * This method is performed when the SaxParser finds any element in
   * the xml document. Because this method is called for very different
   *  elements, this method checks the element name, and set the values
   * of the code>isRecord</code>, <code>isDescription</code>,
   * <code>isUserId</code> and <code>isSubjectWord</code> accordingly.
   * </p>
   * <p>
   * This method  is only trigged at the start of the element. The
   * endElement method is used to add the data in the element, when
   * the end of the element is found. The text in the element are
   * collected in the method.
   * </p>
   * @param namespaceURI the namespaceURI
   * @param sName the simple name of the element
   * @param qName the qualified name of the element
   * @param attrs the attributes to the element
   * @return void
   * @throws SAXExcpetion if something unexpected happens
   */
  public void startElement(String namespaceURI, String sName,
                         String qName, Attributes attrs) throws
      SAXException {

    String eName = sName; // element name
    if ("".equals(eName)) {
      eName = qName; // not namespaceAware
    }
    if (eName.equals("RecordID")) {
      isRecordID = true;
    }
    else if (eName.equals("Description")) {
      isDescription = true;
    }
```

```
    else if (eName.equals("UserID")) {
      isUserID = true;
    }
    else if (eName.equals("SubjectWord")) {
      isSubjectWord = true;
    }
  }

  /***
   * <p>
   * This method is called whe the end of an element is found. Based
   * on the values in the <code>isRecord</code>,
<code>isDescription<code>,
   * <code>isUserId</code> and <code>isSubjectWord</code>. The method
   * collects the characters in the element, and adds theese to
   * the correct attribute in the CollectionRecord object.
   * </p><p>
   * The textBuffer is reset at the end, to not add new data to old.
   * </p>
   * @param namespaceURI the namespaceURI
   * @param sName the simple name of the element
   * @param qName the qualified name of the element
   * @param attrs the attributes to the element
   * @return void
   * @throws SAXExcpetion if something unexpected happens
   */
  public void endElement(String namespaceURI, String sName,
                         String qName) throws SAXException {

    //finding the elementname
    String eName = sName; // element name
    if ("".equals(eName)) {
      eName = qName; // not namespaceAware
    }
    if (isRecordID) {
      recordID = textBuffer.toString();
    }
    else if (isDescription) {
      //insert an Desctiption element, and wait for it to fill.
      description = textBuffer.toString();
    }
    else if (isUserID) {
      userID = textBuffer.toString();
    }
    else if (isSubjectWord) {
      subjectWords.add(textBuffer.toString());
    }
    reset();
  }

  /***
   * Thie method is called each time the SaxParser find a charachter in
the
   * xml document. This method appends the found characters to
   * the <code>textBuffer</code> attribute in this class. This data is
   * available to the endElement method.
```

```
   * @param buf[] a char array
   * @param offset Where to start
   * @param len How long to collect
   * @return void
   * @throws SAXExcpetion if something unexpected happens
   */
  public void characters(char buf[], int offset, int len) throws
      SAXException {
    String s = new String(buf, offset, len);
    if (textBuffer == null) {
      textBuffer = new StringBuffer(s);
    }
    else {
      textBuffer.append(s);
    }
  }

  /***
   * This method is used to reset the attributes of this class between
   * elements.
   */
  private void reset() {
    isRecordID = false;
    isDescription = false;
    isUserID = false;
    isSubjectWord = false;
  } //reset
}
```

G.19 DPC.xml.XmlToLocalMetadataRecordParser

```
package DPC.xml;

import DPC.DB.*;
import DPC.Interface.*;
import DPC.xml.*;
import DPC.*;

import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

import java.io.StringReader;
import java.util.ArrayList;
import java.util.Vector;

/**
 * <p>Title: PDC WebService</p>
 * <p>Description: DPC Servlet/WebService</p>
 * This class is a utility class to convert a xml document into
```

```
 * a LocalMetadataRecord object. This LocalMetadataRecord objcet is used
 * to handle the metadta record internally in the DPC system.
 * @author Sverre Joki
 * @version 1.0
 */
public class XmlToLocalMetadataRecordParser
    extends DefaultHandler {

  /***
   * This attribute is used during the convertion of the xml structure
   * to store the text of the parsed xml document elements.
   */
  private StringBuffer textBuffer;

  /***
   * This attribute is used during the convertion of the xml structure
   * to keep the CollectionRecord object.
   */
  private LocalMetadataRecord lm;

  /***
   * This attribute is used during the convertion of the xml structure.
   * recordID keeps the id for the record.
   */
  private String recordID = null;

  /***
   * This attribute is used during the convertion of the xml structure.
   * metadataURI keeps the uri for the metadata record.
   */
  private String metadataURI = null;

  /***
   * This attribute is used during the convertion of the xml structure.
   * True of the investigated element contains a metadata record
   */
  boolean isMetadataRecord;

  /***
   * This attribute is used during the convertion of the xml structure.
   * True of the investigated element contains a record id
   */
  boolean isRecordID;

  /***
   * This attribute is used during the convertion of the xml structure.
   * True of the investigated element contains a metadata uri
   */
  boolean isMetadataURI;

  /***
   * This constructor creates a empty XmlToLocalMetadataRecordParser
object.
   * This object is used to parse a local metadata record from
   * a xml structure into an internal ldatastructure used by the DPC
system.
```

```
 */
public XmlToLocalMetadataRecordParser() {
}

/***
 * This method is used to parse a local metadata record from a xml
 * structure into  an internal ldatastructure used by the DPC system.
 * The xml structure must be deliverd as a string object containig
 * the xml document.
 * @param xmlInput the String containing the xml document
 * @return LocalMetadataRecord the new object created from the xml
 *                             document
 * @throws SAXException if something goes wrong during parsing
 */
public LocalMetadataRecord parse(String xmlInput) throws
    SAXException {

  // Use an instance of ourselves as the SAX event handler
  DefaultHandler handler = new XmlToLocalMetadataRecordParser();

  // Use the default (non-validating) parser
  SAXParserFactory factory = SAXParserFactory.newInstance();
  try {
    // Parse the input
    SAXParser saxParser = factory.newSAXParser();
    //creates an InputSource from a StringReader created from xmlInput
    saxParser.parse(new InputSource(new StringReader(xmlInput)),
                    handler);

  }
  catch (Throwable t) {
    //Error occured in parsing xml. throw exception
    t.printStackTrace(System.err);
    throw new SAXException("Error in parsing xml");
  }
  return lm;
}

//=============================================================
// SAX DocumentHandler methods
//=============================================================

/**
 * This method is performed when the SaxParser finds the start of the
 * xml document. This is only done once for a collection xml document.
 * The method instasiates a LocalMetadataRecord object.
 * @return void
 * @throws SAXExcpetion if something unexpected happens
 */
public void startDocument() throws SAXException {
  lm = new LocalMetadataRecord();
  reset();
}

/***
 * This method is called when the SaxParser finds the end of the
```

```
   * xml document. This method adds the values found by the parser
   * to the LocalMetadataRedcord object.
   * @return void
   * @throws SAXExcpetion if something unexpected happens
   */
  public void endDocument() throws SAXException {

    //adding personalDescriptions to metadatarecord
//    lm.setUsers(users);
    lm.setRecordID(recordID);
    lm.setmetadataURI(metadataURI);
    reset();
  }

  /***
   * <p>
   * This method is performed when the SaxParser finds any element in the
   * xml document. Because this method is called for very different
   * elements, this method checks the element name, and set the values
   * of the code>isRecord</code> and <code>isMetadataURI</code>
   * accordingly. <p><p>
   * This method  is only trigged at the start of the element. The
   * endElement method is used to add the data in the element,
   * when the end of the element is found. The text in the element
   * are collected in the characters method.
   * </p>
   * @param namespaceURI the namespaceURI
   * @param sName the simple name of the element
   * @param qName the qualified name of the element
   * @param attrs the attributes to the element
   * @return void
   * @throws SAXExcpetion if something unexpected happens
   */
  public void startElement(String namespaceURI,
                           String sName,
                           String qName,
                           Attributes attrs) throws SAXException {

    String eName = sName; // element name
    if ("".equals(eName))
      eName = qName; // not namespaceAware

    if (eName.equals("RecordID"))
      isRecordID = true;
    else if (eName.equals("MetadataURI)"))
      isMetadataURI = true;
  }

  /***
   * <p>
   * This method is called whe the end of an element is found. Based
   * on the values in  <code>isRecord</code> and
   * <code>isMetadataURI</code>.
   * The method collects the characters in the element, and adds
   * theese to the correct attribute in the CollectionRecord object.
   * </p><p>
```

```
 * The textBuffer is reset at the end, to add new new data to old.
 * </p>
 * @param namespaceURI the namespaceURI
 * @param sName the simple name of the element
 * @param qName the qualified name of the element
 * @param attrs the attributes to the element
 * @return void
 * @throws SAXExcpetion if something unexpected happens
 */
public void endElement(String namespaceURI,
                       String sName,
                       String qName
                       ) throws SAXException {

  //finding the elementname
  String eName = sName; // element name
  if ("".equals(eName))
    eName = qName; // not namespaceAware
  if (isRecordID) {
    recordID = textBuffer.toString();
  }
  else if (isMetadataURI) {
    metadataURI = textBuffer.toString();
  }
  reset();
}

/***
 * Thie method is called each time the SaxParser find a character in the
 * xml document. This method appends the found characters to
 * the <code>textBuffer</code> attribute in this class. This data is
 * available to the endElement method.
 * @param buf[] a char array
 * @param offset Where to start
 * @param len How long to collect
 * @return void
 * @throws SAXExcpetion if something unexpected happens
 */
public void characters(char buf[], int offset, int len) throws
    SAXException {
  String s = new String(buf, offset, len);
  if (textBuffer == null) {
    textBuffer = new StringBuffer(s);
  }
  else {
    textBuffer.append(s);
  }
}

/***
 * This method is used to reset the attributes of this class between
 * elements.
 */
private void reset() {
  isMetadataRecord = false;
  isRecordID = false;
```

```
    isMetadataURI = false;
  } //reset
}
```

G.20 DPC.xml.XmlToUserdataParser

```
package DPC.xml;

import DPC.DB.*;
import DPC.Interface.*;
import DPC.xml.*;
import DPC.*;
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import java.io.StringReader;
import java.util.ArrayList;
import java.util.Vector;
import java.util.GregorianCalendar;

/**
 * <p>Title: XmlToUserdataParser</p>
 * <p>Description: XmlToUserdataParser</p>
 * This class is a utility class to convert a xml document into
 * a UserData object This UserData objcet is used
 * to handle the user data internally in the DPC system.
 * @author Sverre Joki
 * @version 1.0
 */
public class XmlToUserdataParser
    extends DefaultHandler {

  private GregorianCalendar cal = new GregorianCalendar();

  /***
   * This attribute is used during the convertion of the xml structure
   * to store the text of the parsed xml document elements.
   */
  private StringBuffer textBuffer;

  /***
   * This attribute is used during the convertion of the xml structure
   * to keep the UserData object.
   */
  private UserData userdata;

  /***
   * This attribute is used during the convertion of the xml structure.
   * userID keeps the id for the user.
   */
  private String userID = null;
```

```
/***
 * This attribute is used during the convertion of the xml structure.
 * name keeps the user name in the UserData object
 */
private String name = null;

/***
 * This attribute is used during the convertion of the xml structure.
 * streetAdress keeps the street adress in the UserData object
 */
private String streetAdress = null;

/***
 * This attribute is used during the convertion of the xml structure.
 * city keeps the user's city in the UserData object
 */
private String city = null;

/***
 * This attribute is used during the convertion of the xml structure.
 * postCode keeps the post code in the UserData object
 */
private String postCode = null;

/***
 * This attribute is used during the convertion of the xml structure.
 * email keeps the user's email adress in the UserData object
 */
private String email = null;

/***
 * This attribute is used during the convertion of the xml structure.
 * position keeps the user's work-position in the UserData object
 */
private String position = null;

/***
 * This attribute is used during the convertion of the xml structure.
 * institution keeps the user's institution in the UserData object
 */
private String institution = null;

/***
 * This attribute is used during the convertion of the xml structure.
 * interests keeps the user's interests in the UserData object
 */
private ArrayList interests = null;

/***
 * This attribute is used during the convertion of the xml structure.
 * description keeps the description in the collection record
 */
private String description = null;

/***
```

```
 * This attribute is used during the convertion of the xml structure.
 * True of the investigated element contains a user id
 */
boolean isUserID;

/***
 * This attribute is used during the convertion of the xml structure.
 * True of the investigated element contains a name
 */
boolean isName;

/***
 * This attribute is used during the convertion of the xml structure.
 * True of the investigated element contains a Street adress
 */
boolean isStreetAdress;

/***
 * This attribute is used during the convertion of the xml structure.
 * True of the investigated element contains a city
 */
boolean isCity;

/***
 * This attribute is used during the convertion of the xml structure.
 * True of the investigated element contains a post code
 */
boolean isPostCode;

/***
 * This attribute is used during the convertion of the xml structure.
 * True of the investigated element contains a email adress
 */
boolean isEmail;

/***
 * This attribute is used during the convertion of the xml structure.
 * True of the investigated element contains a postition
 */
boolean isPosition;

/***
 * This attribute is used during the convertion of the xml structure.
 * True of the investigated element contains a institution
 */
boolean isInstitution;

/***
 * This attribute is used during the convertion of the xml structure.
 * True of the investigated element contains a interst
 */
boolean isInterest;

/***
 * This attribute is used during the convertion of the xml structure.
 * True of the investigated element contains a description
```

```
   */
  boolean isDescription;

  private String cName = "XmlToUserdataParser";

  /***
   * This constructor creates a empty XmlToUserdataParser object.
   * This object is used to parse a user record from xml structure into
   * an internal datastructure used by the DPC system.
   */
  public XmlToUserdataParser() {}

  /***
   * This method is used to parse a user data record from a xml
   * structure into  an internal datastructure used by the DPC system.
   * The xml structure must be deliverd as a string object containig
   * the xml document.
   * @param xmlInput the String containing the xml document
   * @return UserData the new object created from the xml document
   * @throws SAXException if something goes wrong during parsing
   */
  public UserData parse(String xmlInput) throws SAXException {
    String mName = "parse";
    System.err.println(cal.getTime().toString() + cName + "." + mName
                       + " This is the xml input: " + xmlInput);

    // Use an instance of ourselves as the SAX event handler
    DefaultHandler handler = new XmlToLocalMetadataRecordParser();

    // Use the default (non-validating) parser
    SAXParserFactory factory = SAXParserFactory.newInstance();
    System.err.println(cal.getTime().toString() + cName + "." + mName
                       + ": Before trying to parse");
    try {
      userdata = new UserData();
      // Parse the input
      SAXParser saxParser = factory.newSAXParser();
      //creates an InputSource from a StringReader created from xmlInput
      saxParser.parse(new InputSource(new StringReader(xmlInput)),
                      handler);
      System.err.println(cal.getTime().toString() + cName + "."
                         + mName + ": after trying to parse");
      System.err.println(userdata.toString());
    }
    catch (Throwable t) {
      //Error occured in parsing xml. throw exception
      System.err.print("Unntak i parser");
      t.printStackTrace(System.err);
      throw new SAXException("Error in parsing xml");
    }
    return userdata;
  }

//============================================================
// SAX DocumentHandler methods
//============================================================
```

```
/**
 * This method is performed when the SaxParser finds the start of the
 * xml document. This is only done once for a collection xml document.
 * @return void
 * @throws SAXExcpetion if something unexpected happens
 */
public void startDocument() throws SAXException {
  System.err.println(cName + ".starDocument");
  reset();
}

/***
 * This method is called when the SaxParser finds the end of the
 * xml document. This method adds the values found by the parser
 * to the UserData object.
 * @return void
 * @throws SAXExcpetion if something unexpected happens
 */
public void endDocument() throws SAXException {
  System.err.println(cName + ".endDocument");
  userdata.userID = userID;
  userdata.name = name;
  userdata.adress = streetAdress;
  userdata.city = city;
  userdata.postCode = postCode;
  userdata.email = email;
  userdata.position = position;
  userdata.institution = institution;
  userdata.interests = interests;
  userdata.description = description;

  //resetting element indicators
  reset();
}

/***
 * <p>
 * This method is performed when the SaxParser finds any element in the
 * xml document. Because this method is called for very different
 * elements, this method checks the element name, and set the values
 * of the attributes indicating which element is found accordingly.</p>
 * <p>
 * This method  is only trigged at the start of the element. The
 * endElement method is used to add the data in the element, when
 * the end of the element is found. The text in the element are
 * collected in the characters method.
 *</p>
 * @param namespaceURI the namespaceURI
 * @param sName the simple name of the element
 * @param qName the qualified name of the element
 * @param attrs the attributes to the element
 * @return void
 * @throws SAXExcpetion if something unexpected happens
 */
public void startElement(String namespaceURI,
```

```
                           String sName,
                           String qName,
                           Attributes attrs) throws SAXException {
    System.err.println(cName + ".startElement");
    try {
      String eName = sName; // element name
      if ("".equals(eName))
        eName = qName; // not namespaceAware

      if (eName.equals("UserID"))
        isUserID = true;

      else if (eName.equals("Name)"))
        isName = true;

      else if (eName.equals("StreetAdress"))
        isStreetAdress = true;

      else if (eName.equals("City"))
        isCity = true;

      else if (eName.equals("PostCode"))
        isPostCode = true;

      else if (eName.equals("Email"))
        isEmail = true;

      else if (eName.equals("Position"))
        isPosition = true;

      else if (eName.equals("Institution"))
        isInstitution = true;

      else if (eName.equals("Interest"))
        isInterest = true;

      else if (eName.equals("Description"))
        isDescription = true;
    }
    catch (Exception e) {
      e.printStackTrace(System.err);
      throw new SAXException("An error occured during parsing");
    }
  }

  /***
   * <p>
   * This method is called whe the end of an element is found. Based on the
   * values in  <attributers specifying which element that is found by the
parser.
   * The method collects the characters in the element, and adds theese to
   * the correct attribute in the CollectionRecord object.
   * </p><p>
   * The textBuffer is reset at the end, to avoid appending new data to
old.
   * </p>
```

```
   * @param namespaceURI the namespaceURI
   * @param sName the simple name of the element
   * @param qName the qualified name of the element
   * @param attrs the attributes to the element
   * @return void
   * @throws SAXExcpetion if something unexpected happens
   */
  public void endElement(String namespaceURI,
                         String sName,
                         String qName
                         ) throws SAXException {

    System.err.println(cName + ".endElement");

    if (isUserID)
      userID = textBuffer.toString();

    else if (isName)
      name = textBuffer.toString();

    else if (isStreetAdress)
      streetAdress = textBuffer.toString();

    else if (isCity)
      city = textBuffer.toString();

    else if (isPostCode)
      postCode = textBuffer.toString();

    else if (isEmail)
      email = textBuffer.toString();

    else if (isPosition)
      position = textBuffer.toString();

    else if (isInstitution)
      institution = textBuffer.toString();

    else if (isInterest)
      interests.add(textBuffer.toString());

    else if (isDescription)
      description = textBuffer.toString();

  } //endElement

  /***
   * Thie method is called each time the SaxParser find a charachter in
the
   * xml document. This method appends the found characters to
   * the <code>textBuffer</code> attribute in this class. This data is
   * available to the endElement method.
   * @param buf[] a char array
   * @param offset Where to start
   * @param len How long to collect
   * @return void
```

```
   * @throws SAXExcpetion if something unexpected happens
   */
  public void characters(char buf[], int offset, int len) throws
      SAXException {
    try {
      String s = new String(buf, offset, len);
      if (textBuffer == null) {
        textBuffer = new StringBuffer(s);
        System.err.println("textbuffer i charachters == null");
      }
      else {
        System.err.println("textbuffer i charachters != null: append");
        textBuffer.append(s);
      }
    }
    catch (Exception e) {
      e.printStackTrace(System.err);
      throw new SAXException("error getting characters");
    }
  } //characters

  /***
   * This method is used to reset the attributes of this class between
   * elements.
   */
  private void reset() {
    isUserID = false;
    isName = false;
    isStreetAdress = false;
    isCity = false;
    isPostCode = false;
    isEmail = false;
    isPosition = false;
    isInstitution = false;
    isInterest = false;
    isDescription = false;
  } //reset

} //end XmlToUserdataParser
```