UNIVERSITETET I OSLO
Institutt for informatikk

Transformering av tabulære data til RDF

Masteroppgave

Fabian Alknes

2. mai 2011

Sammendrag

Det offentlige genererer og samler inn store mengder data og er av stor interesse for flere sektorer i samfunnet. Medier, bedrifter og privatpersoner kan skape nye tjenester, ny innsikt og økonomiske verdier basert på disse dataene.

Effektiv viderebruk av data forutsetter applikasjonsuavhengige maskinleselige formater. Tim Bernes-Lee's prinsipper om lenkede data viser hvordan man kan benytte seg av RDF spesifikasjonen fra World Wide Web Consortium for å strukturere og tilgjengeliggjøre data som utnytter verdensvevens generelle struktur. På denne måten blir data tilgjengelige for allmennheten og kan enkelt viderebrukes. Per i dag er det derimot slik at mesteparten av offentlige data er lagret i proprietære formater som Microsoft Excel eller i relasjonelle databasesystemer.

Data tilgjengeliggjort er ofte representert som tabulære data. RDF er godt egnet for å representere slik data, ettersom spesifikasjonen kan sees som en abstraksjon av dette. Det finnes allerede applikasjoner som transformerer eller eksponerer tabulære data til RDF. RDFizer beskrevet i denne oppgaven er et slikt program. Dette tar for seg transformering av data lagret i Microsoft Excel regneark. Denne transformasjonen er styrt ut ifra et templat som angir regler for hvordan transformeringen av regnearket skal gjennomføres. Reglene muliggjør enkel styring av transformasjonsprosessen, og hvordan man konstruerer subjekter som verdier i regnearket knyttes til som enten typede literaler eller navngitte ressurser.

Forord

Arbeidet med denne masteroppgaven er utført ved Institutt for Informatikk ved Universitetet i Oslo i perioden 2010 til 2011. Masteroppgaven representerer arbeidet jeg har gjort med transformering av Microsoft Excel regneark til RDF.

Jeg vil benytte muligheten til å sende en takk til Audun Stolpe, som var min veileder ved Institutt for Informatikk, for hjelp og veiledning under arbeidet med oppgaven. Vil også takke venner og medstudenter som har gjort perioden ved Universitetet minneverdig.

Fabian Alknes
1.6.2011

Innhold

Figurer

Tabeller

Kapittel 1

Introduksjon

1.1 Semicolon

Denne oppgaven er skrevet i tilknytning til forskningsprosjektet Semicolon. Semicolon er et Brukerstyrt Innovasjonsprosjekt med deltagelse fra flere store offentlige etater og forskningsinstitusjoner, bland annet Institutt for Informatikk (IFI) ved Universitetet i Oslo (UiO). Semicolon har fokus på økt elektronisk samhandling i offentlig sektor. Samhandling mellom etater, næringsliv og privatpersoner er en nødvendig forutsetning for å effektivisere offentlig tjenesteproduksjonen samt for å heve kvaliteten i tjenestene.

Til tross for tunge investeringer i nye IT-systemer i offentlig sektor, er det fortsatt slik at informasjonsutveksling, i et stort antall tilfeller, er papirbasert. Den er derfor ikke tilgjengelig for datamaskiner og følgelig heller ikke direkte gjenbrukbar. Videre er det nærmest ingen eksempler på at tjenester er lenket sammen på tvers av organisasjoner, for eksempel fra fastlege til trygd eller fra NAV til Lånekassen. Slike tjenester fordrer strømlinjeformede tverrsgående informasjonskanaler; eksisterende dataregistre må integreres og informasjonen som følger prosessene må beskrives på en slik måte at det blir mulig for maskiner å utnytte ulike informasjonsfragmenter for ulike formål.

Semicolon har fokus på åpne data hvor deltagerene i prosjektet forvalter flere interessante data i kategorien åpne data hvor utfordringen er å legge til rette for at disse dataene kan bli tatt i bruk og gjerne lede til innovasjon. Med åpne offentlige data menes informasjon som er samlet inn og bearbeidet av offentlige instanser for diverse forvaltningsmål som ikke er underlagt juridiske restriksjoner som kan hindre publisering av dataene over internett, heretter kalt veven. Fokusen ved Semicolon er todelt hvor den første omhandler datautveksling etater imellom der dataene typisk er unndratt offentligheten, og det andre dreier seg istedet om publiseringe av data der man forutsetter at

informasjonen er juridiske åpne og i prinsippet tilgjengelige for allmenheten.

1.2 Oris/Synapse: Semantisk portal til åpne offentlige data.

Det er veldig mye og veldig forskjellig type informasjon som ligger offentlig tilgjengelig i dag, ikke minst fordi en rekke etater er pålagt å publisere åpne data. Imidlertid er det i mange tilfeller vanskelig å finne slik informasjon, rett og slett fordi det ikke eksisterer en felles infrastruktur for den. Det finnes ikke engang en systematisk oversikt over hvem som har hvilke data, langt mindre en oversikt som lenker sammen relatert informasjon fra ulike etater.

Oris/Synapse er en portal som tar for seg det andre fokuspunktet i Semicolon og er ment som et publiseringsverktøy for åpne offentlige data. Oris/Synapse genererer og sammenstiller semantiske data hvor den overordnede strukturen er todelt:

- Å hjelpe allmenheten med å lokalisere og analyserer offentlige datakilder og å gjenbruke data til egne mål.
- Å hjelpe etater og dataeiere med å publiserer RDF data, og eksplisere og gjøre konkret bruk av sine begrepsdefinisjoner.

1.2.1 Synapse

Synapse tilbyr konverteringshjelp hvor man gjennom portalen kan laste opp et Excel regneark og et templat som angir regler for transformasjonsprosessen. Det resulterende datasettet returneres i form av RDF og lagres eksternt hos brukeren. Metadata som beskriver datasettet blir da tilgjengelig gjennom Oris. I tillegg tilbyr Synapse et registreringsgrensesnitt hvor man kan registerer allerede eksisterende datasett slik at de blir tilgjengelige gjennom Oris.

1.2.2 Oris

Oris tilbyr søk i datasett i form av fritekstsøk og kategorier. I tillegg inneholder Oris en handlevogn hvor brukere velger ut datasett registrert i systemet. Oris vil da flette disse datasettene sammen og presenteres dem i et grensesnitt som tilbyr analyseringsfunksjonalitet. Til slutt kan man generere nye RDF datasett fra resultatet for gjenbruk

1.3 Problemstilling

Arbeidet med denne oppgaven har gått ut på transformering av Excel regneark til RDF og inngår som en del av Synapse. Oppgaven er delt inn i to deler.

1. Spesifiserer et templatspråk som angir regler for hvordan et gitt Excel regneark transformerers til RDF
2. Utvikle en applikasjon som uttnytter templatspråket og gjennomfører transformasjonen av Excel regneark.

Synapse er lokalisert på følgende vevadresse: `http://sws.ifi.uio.no/synapse/`

Excel regneark og tilhørende templater som kan benyttes for å teste transformasjonsapplikasjonen i Oris/Synapse er lokasliert på følgende vevadresser:

- Byantikvarens gule liste
  - `http://folk.uio.no/fabianal/rdfizer/gulliste.xls`
  - `http://folk.uio.no/fabianal/rdfizer/gulliste.ttl`
- Liste over utrykninger av Oslo Branvesen
  - `http://folk.uio.no/fabianal/rdfizer/brann.xls`
  - `http://folk.uio.no/fabianal/rdfizer/brann.ttl`
- Statistikk fra Statistisk Sentralbyrå
  - `http://folk.uio.no/fabianal/rdfizer/ssb.xls`
  - `http://folk.uio.no/fabianal/rdfizer/ssb.ttl`

1.4 Oppgavens oppbyggning

Resten av oppgaven er strukturert som følger: I kapittel 2 gjennomgås tilgjengeliggjøring av juridisk åpne data, og hvilke representasjonsformer som er godt egnet med hensyn til gjenbruk. Kapittel 3 tar for seg de essensielle semantiske vevteknologiene, mens kapittel 4 inneholder en gjennomgang av trabulære data og RDF samt eksisterende programmer for transformering av data til RDF. Kapittel 5 tar for seg RDFizer som er laget under arbeidet med denne masteroppgaven for å transformere Microsoft Excel regneark til RDF. Kapittel 6 viser til mulige utvidelser av uttrykkskraften til RDFizer samt eksempler på hvordan dette eventuelt kan representeres i et templat.

Kapittel 2

Tilgjengeliggjøring av offentlige data for gjenbruk

Det offentlige produserer store mengder data som ofte er utarbeidet til et bestemt formål. Det blir brukt mye ressurser i form av tid og penger for å bygge opp og samle inn dataene. Det er vist stor økonomisk interesse for åpning av disse dataene for gjenbruk, hvor data kan bli benyttet i formål de originalt ikke var ment. Kombinering av forskjellige aktøres data kan skape nye og interessante data. Offentlige etater kan benytte andre aktørers og etaters data istedet for å konstruere og samle inn egne data som allerede er produsert, noe som vil kunne medføre et kutt i både tid og ressurser fordi man ungår dobbeltarbeid. Tilgjengeliggjøring av data vil også medføre en større åpenhet og forståelse mellom de forskjellig offentlige aktørene og deres arbeid, samtidig vil det kunne gi en økonomisk gevinst i form av skatteinntekter for nye tjenester.

Nye tjenester som baserer seg på offentlig frigjort data vil kunne føre til verdiskapning i samfunnet. Åpne offentlige data kan for eksempel benyttes for å lage en tjeneste som gir deg informasjon om verneverdige bygninger ut ifra geografiske koordinater eller navnet på et sted, eller en tjeneste som viser hvor den nærmeste politistasjonen er lokalisert. I tillegg til nye tjenester vil kombinering av data fra ulike aktører kunne føre til nye informasjonssammenhenger. Eksempelvis kunne man undersøkt om det er noen sammenheng mellom forskjellige typer branner og været den dagen brannene forekom.

I tillegg til verdiskapning vil tilgjengeliggjøring av offentlige data kunne føre til større transparens i samfunnet hvor for eksempel personer, organisasjoner og medier kan kontrollere den utøvende makten, noe som kan bidra til å styrke demokratiet. Eksempler på transparens kan være:

- statens faktiske bruk av skattepenger.
- hvordan forskjellige kommuner benytter penger i forhold til hverandre.
- fordeling av statlig støtte til forskjellige organisasjoner.

Transparens, gjenbruk og tilgjengeliggjøring av data vil ikke kun ha innvirkning for kontroll av den utøvende makten, men også mediene kan kontrolleres. Når det blir større transparens i grunnlaget for hvordan de har kommet frem til sine nyhetssaker vil andre kunne kontrollere riktigheten av konklusjonen som eventuelt er trukket ut ifra kildegrunnlaget.

Den politiske viljen for tilgjengeliggjøring av offentlige data, og hvilke innvirkning dette vil ha både økonomisk og samfunnsmessig har blitt tydeligere for flere de siste årene. Initiativer i for eksempel Storbritannia, Danmark og USA viser til dette.

2.1 Åpning av data i utlandet

EUs direktiv om gjenbruk av offentlig sektors informasjon, direktiv 2003/98/EF, som har fått tilnavnet "viderebruksdirekivet" ble vedtatt 17. november 2003[2]. Målet med direktivet er å bidra til å fremme økt gjenbruk av informasjon produsert av den offentlige sektor. Et av hovedmålene med viderebruksdirektivet er uttrykket i innledningen, nummerert (5).

> *"Et af hovedformålene med oprettelsen af det indre marked er at skabe betingelser, der fremmer udviklingen af tjenesteydelser, som dækker hele Fællesskabet. Den offentlige sektors informationer er et vigtigt kildemateriale for produkter og tjenester med digitalt indhold, og de vil få stadig større betydning som indholdsressource til de mobile indholdstjenester, der er under udvikling. En bred geografisk dækning på tværs af grænserne er også vigtig i denne forbindelse. Mere vidtgående muligheder for at videreanvende den offentlige sektors informationer bør bl.a. give de europæiske virksomheder mulighed for at udnytte disses potentiale og bidrage til økonomisk vækst og jobskabelse."*

Direktivets artikkel 5 viser til hvordan data fra offentlige aktører burde tilgjengeliggjøres i alle eksisterende formater og språkversjoner, samt i elektronisk form så langt dette er mulig og hensiktsmessig. Denne tilgjengeliggjøringen skal ikke forplikte offentlige aktører til å tilpasse sine data etter anmodning eller stille utdrag fra dokumenter til rådighet hvis dette medfører ekstra arbeid. Målsetningen med direktivet er å samføre praksis i reglene for gjenbruk av offentlig informasjon i EU-landene[2].

Storbritannia

Et av landene som har tatt et stort sprang med innføring av viderebruksdirektivet er Storbritannia. Handlingsplanen "Putting the Frontier First: Smarter Government" som ble presentert av Gordon Brown 7. desember 2007 viser den britiske regjeringens plan for reform av offentlig sektor. I handlingsplanen definerer den britiske regjeringen et sett av prinsipper for offentlig håndtering av data som inneholder blant annet at data skal publiseres i maskinlesbare formater tilrettelagt for viderebruk og benytte seg av standarder gitt av World Wide Web Consortium (W3C). Utformingen av handlingsplanen ligger i kjølevannet av den politiske uroen i landet hvor flere "skandaler" om politikeres bruk av skattepenger til egne goder ble avdekket. Brown utnevnte direktøren for W3C, Tim Berners-Lee, til frontfigur for satsningen som lover en radikal åpning av offentlige sektors datasett og større satsning på transparens[3]. Prinsippene tilsier også at offentlig datasett skal tilgjengeliggjøres gjennom en felles portal, `data.gov.uk`. Målet med denne portalen:

> *"Making this data easily available means it will be easier for people to make decisions and suggestions about government policies based on detailed information"*[4].

Vektleggingen av åpning av datasett i Storbritannia er oppsiktsvekkende og viser en indikasjon på politisk vilje for temaet og har vakt oppmerksomhet internasjonalt[3].

Danmark

I Danmark har IT- og Telestyrelsen[1] satt i gang flere initiativer for å sette søkelys på potensialet i viderebruk av offentlig data gjennom portalen `digitaliser.dk` som inkluderer en katalog over datakilder. Når portalen ble lansert var det samlet inn informasjon om 900 datakilder og blir fortløpende oppdatert [3]. Målet med portalen er.

> *"Digitalisér.dk er den nye fælles indgang til digitalisering for alle myndigheder, leverandører og andre, der ønsker at deltage i udviklingen af det digitale Danmark"*[5]

[1] IT- og Telestyrelsen, Ministriet for Vitenskab, Teknologi og Utvikling – arbeider for å infri potenisalet som ligger i bruket og utbredelsen av Informasjonteknologi i samfunnet

USA

Det er ikke bare i Europa det har blitt igangsatt initiativer for å åpne offentlige data for viderebruk. Myndighetene i USA har gjort mer åpenhet i offentlig forvaltning til en prioritert satsning gjennom sitt "Open Government Initiative", hvor åpning av datasett fra den offentlige sektoren ansees som et av de viktigste virkemidlene. Portalen `data.gov` ble publisert i mai 2009 for å gjøre flere datasett tilgjengelige gjennom et felles tilknytningspunkt. I tillegg til rådata i ulike formater finnes det en katalog over rapporter og analyser av de forskjellige datasettene[3]. Portalen har som mål:

> *"A primary goal of `Data.gov` is to improve access to Federal data and expand creative use of those data beyond the walls of government by encouraging innovative ideas (e.g., web applications). `Data.gov` strives to make government more transparent and is committed to creating an unprecedented level of openness in Government. The openness derived from Data.gov will strengthen our Nation's democracy and promote efficiency and effectiveness in Government."*[6]

I desember 2009 innførte regjeringen i USA et direktiv der offentlige virksomheter ble bedt om å innføre en rekke krav til transparens. I tillegg skulle alle virksomhetene publiserer minst tre tidligere utilgjengelige datasett av "høy kvalitet", i et åpent format og publisere dem gjennom portalen `data.gov` innen 45 dager. Hva som regnes som datasett av "høy kvalitet" er definert i direktivet fra 2009[3].

2.2 Tilgjengeliggjøring av data i Norge

En interdepartemental arbeidsgruppe ble satt ned den 19. juni 2003 av regjeringen. Arbeidsgruppen skulle utrede prinsipper for prising og tilgang til offentlig informasjon for kommersiell utnyttelse og hvordan viderebruksdirektivet kunne innlemmes i norsk lov[7]. Arbeidsgruppens arbeid resulterte i rapporten "Fra bruk til gjenbruk" og ble publisert 30. august 2004 og gir anbefalinger til endringen av direktivet for innpass i norsk lov. Anbefalingene fra rapporten førte til innlemmelse av viderebruksdirekivet i "LOV 2006-05-19 nr 16: Lov om rett til innsyn i dokument i offentleg verksemd (offentleglova)" hvor paragraf 1 sier

> *"Formålet med lova er å leggje til rette for at offentleg verksemd er open og gjennomsiktig, for slik å styrkje informasjons- og*

> *ytringsfridommen, den demokratiske deltakinga, rettstryggleiken for den enkelte, tilliten til det offentlege og kontrollen frå ålmenta. Lova skal òg leggje til rette for vidarebruk av offentleg informasjon."[8].*

Loven gir regler for innsyn i data i form av for eksempel å kunne kreve en kopi av et dokument i alle eksisterende formater og kreve digital kopi. Samtidig gis det regler for hva som er unntatt fra innsyn.

Moderniseringsdepartementet leverte rapporten "eNorge 2009"[9] som definerer målsetninger for hvordan man kan realisere mulighetene informasjonsteknologi gir i Norge. Rapporten statuerer i innledningen at:

> *"Regjeringen ønsker et kunnskapssamfunn hvor alle kan delta og hvor potensialet i informasjonsteknologien utnyttes. Norges avanserte teknologibruk skal gi innbyggere og næringsliv en enklere hverdag og bidra til å fremme verdiskaping og dermed trygge velferden for nye generasjoner. Informasjonsteknologi skal understøtte utvikling av en offentlig sektor som leverer best mulig tjenester for ressursene den disponerer. Innbyggernes og næringslivets behov skal stå i sentrum for utviklingen av det digitale Norge."*

Rapporten "eNorge 2009" oppgir flere mål hvor noen eksempler er:

- "I løpet av 2008 skal det være iverksatt en helhetlig politikk som sikrer effektiv viderebruk av offentlige data for økt verdiskapning og utvikling av nye tjenester, med utgangspunkt i gratisprinsippet"[9].
- "I løpet av 2006 skal det være etablert et sett av forvaltningsstandarder for data- og dokumentutveksling"[9].
- "I løpet av 2008 skal data og dokumentutveksling i offentlig sektor tilfredsstille forvaltningsstandardene"[9].

Dette er uttrykk for en økt vilje når det gjelder åpning og gjenbruk av informasjon produsert av offentlige etater i Norge. Rapporter utarbeidet av og for det offentlige, og innføring av viderebruksdirektivet medfører et viktig steg i retningen av større transparens og gjenbruk. Fornyingsdepartementet opprettet i april 2010 portalen `data.norge.no` som originalt var ment som en blogg med målsetning om å være "en møteplass ikke bare mellom databrukere og Fornyingsdepartementet, men mellom databrukere og dataeiere"[10]. I første innlegg etterlyste statsråden for Fornyings-, administrasjons- og

kirkedepartementet, Rigmor Aasrud, hjelp til hvordan utviklingen av nettstedet skulle gjennomføres og hvilke datasett som skulle tilgjengeliggjøres gjennom nettstedet hvor hun samtidig beskriver sin visjon av data.norge.no som en "møteplass for alle som arbeider med viderebruk av offentlige data. At offentlige data kan brukes av andre enn de etatene og kommunene som har samlet dem inn har nemlig et stort verdiskapingspotensial."[10] og skal være en portal i likhet med Storbritannias `data.gov.uk` og USAs `data.gov`.

Et eksempel på effekten av tilgjengeliggjøring av data er Avinors som sommeren 2009 frigjorde deler av sine data om trafikkbevegelse for fly. Dataene ble frigjort i form av Extensible Markup Language (XML) som kan gjenbrukes av alle såfremt det ikke er i strid med norsk lov og rett. Avinor er et firma som koordinerer trafikkinformasjon fra alle flyselskaper og deler disse dataene, flytider, flyplassnavn og flyselskaper, i form av XML[11]. Anders Christensen i konsulentfirmaet Bekk oppsummerte erfaringene og effekten av frigjøringen og viste til en økt datakvalitet og tjenesteorientering hvor Avinor fikk tilbakemelding på hvilke data som eventuelt manglet. Frigjøringen førte også til økt kollektiv innovasjon gjennom applikasjoner og tjenester som ble opprettet ut ifra de tilgjengelige dataene. Ved å være tidlig ute med åpning av data har Avinor fått en positiv omdømmeeffekt. Dette viser at frigjøring av data kan være med på å lage nye tjenester, hvor også bedrifene som publiserer data vil kunne få en positiv effekt[3].

2.3 Holdninger til åpne data

Offentlighetsloven sier i paragraf 9 at "alle kan kreve innsyn i en samstilling av opplysninger som er elektronisk lagret i databasene til organet dersom samstillingen kan gjøres med enkle fremgansmåter". Allmenheten har dermed rett til tilgang til offentlige data og dokumenter såfremt disse ikke er underlagt unntak fra offentlighetsloven. Et eksempel på unntak vil da være data som er underlagt taushetsplikt.

Et kartleggingsprosjektet har blitt utført av Institutt for informasjon– og medievitenskap ved Universitetet i Bergen, hvis mål var å undersøke hvilke datakilder offentlige virksomheter forvalter, og hva som hindrer tilgjengeliggjøring av mere av disse dataene for viderebruk[3]. Kartleggingen ble utført i perioden august til desember 2009 og resulterte i rapporten "fakta først – Viderebruk av datakilder i offentlig sektor: potenisale og hindringer".

Rapporten viser til at effektiv viderebruk av data forutsetter at virksomhetene i offentlig sektor informerer om datakildene de forvalter og gjør data tilgjengelige i relevante formater. En stor del av virksomhetene som var med i kartleggingsprosjektet tilbyr mangelfull eller ingen informasjon

om datakilder på sine hjemmesider. Dermed svikter en grunnleggende forutsettning for viderebruk av data hos virksomhetene dette gjelder[3]. I tillegg viser kartleggingsprosjektet at kostnaden for å tilgjengeliggjøre data, samt at andre aktører kan missforstå dataene og spre villedende informasjon, er de største hindringene for større frigivelse av data[3].

2.4 Hvordan tilgjengeliggjøre data

Hvordan kan man få best utnyttelse og gjenbruksverdi av dataene når de tilgjengeliggjøres? Hvor skal dataene publiseres slik at flest mulig får tilgang? I hvilke formater skal dataene tilgjengeliggjøres? Hvilke data skal publiseres? Dette er alle spørsmål som er viktig i sammenheng med hvordan man best kan tilgjengeliggjøre data for gjenbruk.

For å forsikre om at flest mulig får tilgang til data burde publiseringen av slike datasett skje gjennom et rom som er tilgengelig og kjent for alle. Veven har forandret måten vi deler kunnskap og informasjon ved å gi tilgang til dokumenter gjennom et felles informasjonsrom. På veven kan man utnytte søkemotorer gitt av for eksempel Google og Microsoft, hvor dokumenter på veven er indeksert og gjort søkbare. Ved da å benytte seg av søkeord vil søkemotoren prøve å gi deg de mest relevante dokumentet ut ifra dette. Dokumenter på veven kobles sammen ved å benytte seg av hypertekstlenker. Ettersom veven er et kjent informasjonsrom burde data gjøres tilgjengelig der slik at de kan nå flest mulig.

Det finnes store menger data produsert av det offentlige og andre som er presentert i forskjellige dokumentformater. Dette kan for eksemel være data lagret i en database som er eksponert på veven i form av Hypertext Markup Language (HTML) tabeller, XML strukturert informasjon eller dokumenter som inneholder tegnseparerte verdier. Hvilket format man tilgjengeliggjør data i er viktig i forhold til gjenbruksgraden av innholdet og eventuelt hvor mye ekstra arbeid som må gjennomføres for å utnytte tilgjengelig data for et bestemt formål. Eksempler på formater som ikke vil være egnet til gjenbruk vil kunne være bilder som gir en visuell representasjon av for eksempel en tabell. Dataene gitt innenfor et slikt bilde vil ikke enkelt kunne gjenbrukes uten mye ekstra arbeid. I tillegg finnes det mange applikasjoner som benytter seg av patentbeskyttede eller lukkede formater som for eksempel et firma har kontroll over og som kan koste penger. Slike properitære formater burde unngås ettersom de ikke nødvendigvis vil kunne være tilgjengelige for alle, og dermed ikke være åpne.

Formatene benyttet for å tilgjengeliggjøre data for gjenbruk burde være maskinprosesserbare i den betydningen at de kan åpnes, leses og behandles

av maskiner samtidig som de burde være i et åpent format. Formatet burde også kunne gjengi strukturen på dataene slik at de bevarer meningen. XML er et eksempel på et slikt format hvor data representeres i form av trær hvor alt som er innenfor en gren er tilhørende det foregående nivået. I tillegg kan tabulær fremstilling i form av tegnseparerte verdier også benyttes ettersom de kan inneholde en rad som gir de forskjellie bestandelene betydning ut ifra en overskriftskolonne.

Informasjonsbitene innenfor et dokument eller datasett er ofte en direkte del av teksten den forekommer i, hvor en maskin ikke vil ha muligheten til å bestemme hvordan delene av dataene er relatert til hverandre. Dette kan for eksempel være en tabell på veven som inneholder en oversikt over hvem som har vært stortingsrepresentanter fordelt på år. Maskinprosesserbare formater er strukturet på en maskintolkbar måte. Dette vil si at det finnes applikasjoner eller verktøy som kan hente ut informasjon av dokumentet på en måte slik at meningen av dataene blir bevart. Eksempler på dette vil være XML som formaterer og markerer bestandelene av et datasett ut ifra gitte beskrivelser. I tillegg vil dette kunne være tabulær fremstilling av data hvor man kan benytte seg av rad, kolonne og celle referanser for å uttrykke hva meningen med dataene er.

Ettersom det ikke alltid er benyttet en og samme standard for å publisere data blir sammstilling av datasette et problem. Samstilling vil kunne medføre at man må konvertere et datasett til et annet format før de faktisk kan slås sammen. Samtidig vil navngivning og identifisering innen forskjellige publiserte dokumenter variere, noe som kan medføre vanskelighet for å samstille data hvor man eventuelt må gjøre noe med identifikatoren i det ene datasettet for at det skal kunne benyttes sammen med det andre datasettet.

Formatene datasettene er publisert i er ikke det eneste man burde tenkte på når man skal tilgjengeliggjøre egne data for gjenbruk. Man burde for eksempel ikke ta forhåndsbestemte valg på hvordan man ønsker at dataene skal utnyttes, data burde ikke aggregeres og heller publiseres i råform.

Brann og redningsetaten i Oslo er et godt eksempel på en etat som tar forhåndsantagelser på hvordan data skal benyttes. På etatens vevsider finnes en oversikt over hvilke typer utrykning brannvesenet har hatt på årlig basis. Denne oversikten er tilgjengeliggjort gjennom et Microsoft Excel regneark som viser antall utrykninger per måned per type utrykning. Dataene er dermed aggregert og gir ingen tilleggsinformasjon om hver enkelt utrykning og begrenser hvordan dataene kan bli gjenbrukt av andre. Ved å aggregere data på forhånd vil de som benytter seg av dataene ikke nødvendigvis få full utnyttelse av innholdet i datasettet til en etat, ettersom mye informasjon kan bli borte, avrundet eller ikke tatt med i det hele tatt. Derfor burde data publiserers i råform så langt dette er mulig.

2.5 Lenkede data

De senere årene har veven utviklet seg fra et globalt informasjonsrom for dokumenter til et informasjonsrom for både dokumenter og dataelementer innenfor dokumenter. Den har dermed utviklet seg fra å være en vev av dokumenter til en vev av data. Lenkede data beskriver en metode for hvordan man kan publiserer strukturert data slik at de kan bli lenket sammen og kan leses av maskiner. Ideen bak "lenkede data" er benytte seg av den generelle arkitekturen for veven for å dele strukturert informasjon i et globalt tilgjengelig rom. Teknologiene benyttet på veven som HTTP muliggjør traversering av dokumenter lokalisert forskjellige steder og er dermed viktig for å lenke sammen data lokalisert innenfor forskjellige datasett.[12]

Tim Berners-Lee publiserte i 2006 et designnotat[13] som viser fire prinsipper for hvordan data kan representeres på veven. Disse fire prinsippene er kjent som "lenkede data prinsipper" og refereres ofte til som den beste metoden for å publisere data på veven.

- Bruk URI'er som navn på ting.
- Bruk HTTP–URI'er slik at personer kan gjøre oppslag på tingen navnene identifiserer.
- Når noen utforsker en URI, gi informasjon om tingen i form av standardene (RDF, SPARQL).
- Inkluder lenker til andre ting gjennom URI'er, slik at andre kan finne og oppdage nye ting.

Prinsippene

Det første prinsipper sier at man skal benytte seg av Uniform Resource Identifiser (URI'er) for å identifiserer og navngi ting. URI'er kan benyttes til å navngi alt som har en tydelig definert betydning som for eksempel filmer eller bøker, til mer abstrakte konsepter som for eksempel relasjonen mellom to dataelementer.[12].

HTTP er en godt kjent protokoll som gir tilgang til ressurser lokalisert på veven. Det andre lenkede data prinsippet sier at man skal benytte seg av HTTP-URI'er for å identifiserer og navngi ting noe som muliggjør oppslag på den navngitte tingen. I figur-2.1 er "The Lord of the Rings" navngitt ved å benytte seg av HTTP-URI.

For å muliggjøre at forskjellige applikasjoner kan utnytte de samme ressursene på veven er det viktig med et standardisert format for å

```
http://dbpedia.org/page/The_Lord_of_the_Rings
```

Figur 2.1: Eksempel på HTTP-URI som navngir tingen "The Lord of the Rings"

representere data. Det tredje prinsipper tilsier at man skal benytte seg av Resource Description Framework (RDF) som er laget spesielt med tanke på global informasjonsdeling[14]. Dette er en datamodell for å strukturere data på veven og beskriver hvordan dataelementer er relatert til hverandre i form av rettede grafer. RDF-grafene uttrykkes i form av tripler (subjekt, predikat og objekt) hvor alle bestanddelene kan navngis av URI'er. Subjektet er den bestanddelen som beskrives, mens predikatet angir hvordan objektet i triplet er relatert til subjektet. Figur-2.2 viser hvordan subjektet, "The Lord of the Rings" er relater til objektet "Fantasy books by series" igjennom predikatet "subjekt".

```
Subjekt: http://dbpedia.org/page/The_Lord_of_the_Rings
Predikat: http://purl.org/dc/terms/subject
Objekt: http://dbpedia.org/resource/Category:
  Fantasy_books_by_series
```

Figur 2.2: Eksempel på et RDF-tripel

Det siste prinsippet til Berners-Lee sier man skal gi lenker til andre URI'er. På denne måten kan man koble data sammen fra forskjellige kilder hvor man benytter seg av datalementer som er navngitt og beskrevet eksternt. Denne sammenkoblingen av dataelementer skaper en vev av data som kan traverserers for å finne nye ting, og plasserer dataelementene i en kontekst.

Hvorfor tilgjengeliggjøre data som lenkede data

Lenkede data egner seg godt til å tilgjengeliggjøre åpne offentlige data ettersom prinsippene er basert på ikke lukkede formater og man kan enkelt kombinere data fra forskjellige kilder for å danne nye datasett. Samtidig er det enkelt å legge med mer lenkede data en det som allerede er i et datasett[15]. Prinsippene for lenking av data egner seg til tilgjengeliggjøring av åpne data av blandt annet følgende grunner.

- Utnytter den allerede eksisterende strukturen for veven
- Baserere seg på åpne standarder

- En datamodell for hvordan data skal struktureres, som beskriver forholdet mellom data
- En metode for å navngi ting

Teknologiene og prinsippene for å publisere "lenkede data" er i stor grad benyttet under arbeidet med denne masteroppgaven. Dette i stor grad for å navngi dataelementer ved å benytte seg av URI'er, strukturere dataelementer i form av RDF og hente ut informasjon av RDF dokumenter ved å benytte seg av SPARQL. Disse teknologiene refereres ofte til som semantisk vevteknologier og er gjennomgått i større detalj i kapittel 3.

The Linking Open Data Project

"The Linking Open Data project"[2] er et av de mest synlige eksemplene hvor Berners-Lee's prinsipper om "lenkede data" benyttes. Prosjektet har som målsetning å indentifiserer og konvertere åpne datasett til RDF ut ifra prinsippene til "lenkede data" og publiserer disse på veven[16].

Prosjektet besto i starten hovedsakelig av utviklerer og forskerer, men har i den senere tid vokst til å inkludere flere større organisasjoner. Denne veksten er i stor grad knyttet til den åpne naturen til prosjektet hvor alle kan delta ved å publisere data som følger Berners-Lee's prinsipper for lenkede data og lenke disse sammen med eksisterende datasett. Figur-2.3 gir et innblikk av veven av data, hvor hver node representerer et entydig datasett publisert som lenkede data[16].

Innholdet varierer fra geografiske data, personer, bøker, filmer, musikk og tv, osv. Som vist i figur-2.3 fungerer noen datasett som knutepunkter for andre datasett. Dette er for eksempel DBpedia[3] som inneholder RDF tripler hentet fra informasjonsboksene på Wikipedia[16]

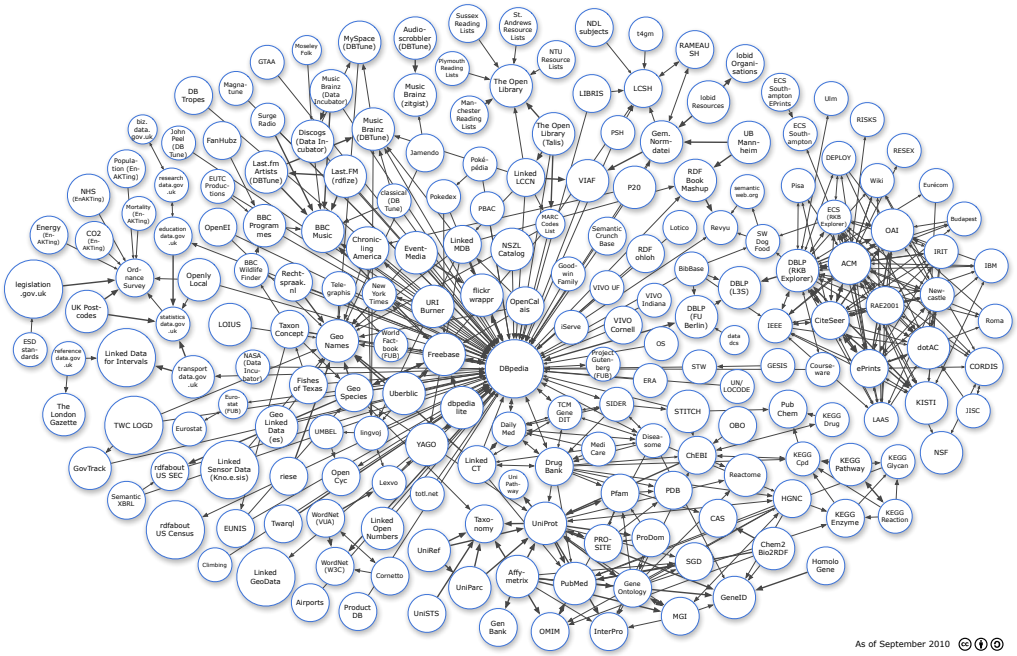
2.6 Femstjernes publisering av data

Tim Berners-Lee oppdaterte i 2010 sitt originale designnotat om lenkede åpne data. Denne oppdateringen inneholdt et stjernebasert graderingssystem for gjenbrukbarheten til publisert data.

1. Tilgjengeliggjort på veven (uansett format), med åpen lisens.

[2] `http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`

[3] `http://dbpedia.org/`



Figur 2.3: The Linking Open Data cloud diagram.

2. Som 1, tilgjengeliggjort i et maskinprosesserbart format(for eksempel excel-format istedet for et bilde av tabell).

3. Som 2, samtidig i et ikke proprietært format (for eksempel dokument med kommaseparerte verdier, CSV, istedet for excel).

4. Som 3, men benytter seg av åpne standarder fra W3C (RDF og SPARQL) for å identifiserer ting slik at ressurser på veven blir lenkbare.

5. Som 4, men også lenk egen data sammen med andres data for å skape kontekst.

Denne oppdateringen ble gitt for å oppmuntre personer og organisasjoner som publiserer data til å tilgjengeligjøre dataene som lenkede data. Data burde åpnes i samsvar med høyest mulig gjenbrukbarhetsgrad, men publisering med den laveste graden er bedre en ingen publisering. Publisering av data med tre stjerner burde være overkommelig for de fleste organsiasjoner. Data lagret i properitære formater som for eksempel Microsoft Excel har mulighet til å eksportere dataene innenfor et dokuemnt som tegnseparerte verdier. Slike dokumenter kan deretter transformeres til RDF ved å benytte seg av verktøy og applikasjoner tilgjengelig på nett (se kapittel 4 for applikasjoner som gjør dette), for å øke gjenbrukbarhetsgraden til fire eller

fem stjerner. For å publisere femstjernes data på veven forutsettes det kunnskap om allerede eksisterende data på den semantiske veven.

Kapittel 3

Semantiske vevteknologier

Den semantiske veven er en samling av data og ressurser koblet sammen gjennom definerte relasjoner i form av lenkede data. Gjennom disse relasjonene skapes en kontekst som fastholder sematikk og språkbegreper [17]. Betydningen av "semantikk" viser til meningen av data. Den semantiske veven skiller seg ikke fra den vanlige veven, men ansees heller som en utvidelse hvor innhold på veven gis veldefinert mening og kan prosesseres og benyttes av maskiner[1]. Ved å uttrykke relasjoner mellom data gis dataene en betydning som ikke varierer avhengig av sammenhengen dataene opptrer i. Den semantiske veven handler i hovedsak om to ting. Hvordan bruke standardiserte formater for å representere data og hvordan tilgjengelig data på veven kan gjenbrukes[1].

W3C har i de siste årene utarbeidet og publisert en rekke anbefalinger som støtter utveksling av semantikkrik informasjon på veven [1]. W3C er den international hovedorganiasjonen for utarbeidelse av spesifikasjoner for veven[18], opprettet og ledet av Tim Berners-Lee som ansees som hovedpersonen bak tanken om den semantiske veven. Spesifikasjoner som er utarbeidet av organisasjonen som RDF[14] og SPARQL[19, 20] utgjør viktige teknologier for den semantiske veven. Spesifikasjoner utarbeidet av W3C, såkalte anbefalinger, ansees som standarder for vevteknologier.

3.1 RDF

For å representere data på den semantiske veven kan man benytte se av RDF[14] som beskriver en datamodell for å representere informasjon på veven i form av rettede grafer. Den semantikse veven antar en maskinprosesserbar måte å representere kunnskap på. Et av hovedmålene med RDF er å representere data og informasjon på en måte som tillater applikasjoner

og teknologier å gjenbruke og samhandle andres data i egne dokumenter og datasett hvor dataenes originale betydning bevares. Grafer uttrykket i RDF trenger ikke å være fullt lokalisert innenfor et dokument eller datasett ettersom URI'er benyttes for å entydig identifisere (navngi) ressurser, begreper og uttrykke relasjoner mellom forskjellige ressurser. På denne måten kan en ressurs benyttet i et datasett eller dokument være beskrevet i et annet lokalisert ut ifra URI'en som identifiserer ressursen. Denne entydige identifiseringen av ressurser og begreper gir muligheten for utveksling av informasjon på tvers av systemer og applikasjoner, og RDF egner seg derfor til å kommuniserer og samordne data fra forskjellige kilder som for eksempel relasjonsdatabaser. Ved å bevare betydningen av innhold forsikrer man seg om at publisert data blir gjenbrukt på en form hvor de ikke blir tatt ut av konteksten de originalt var ment.

Originalt var RDF tiltenkt som en spesifikasjon for hvordan uttrykke metadata for å beskrive data, i form av dokumenter, publisert på veven. Metadata kan defineres som "data om data", men i sammengeng med den semantiske veven vil metadata være "data som beskriver en ressurs på veven" (se kapittel 3.3). RDF spesifikasjonen fra 1999 introduserte en datamodell for hvordan representere metadata og en syntaks for å enkode og transportere data. Målet var å spesifiserer en mekanisme for å beskrive ressurser som ikke var begrenset av spesifikke applikasjonsdomener men fortsatt var egnet for å beskrive informasjon om hvilket som helst domene. RDF spesifikasjonen fra 1999 ble utvidet i 2004 ved å endre konseptet vevressurs til å representere alle data og dataelementer som kan uttrykkes på veven[1].

3.1.1 RDF-modellen

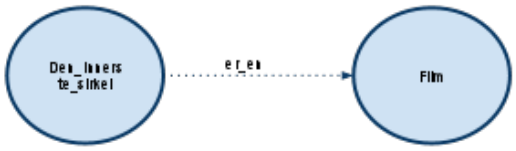
RDF er en standrad for å beskrive relasjoner mellom data/ressurser. I RDF datamodellen beskrives disse relasjonene i form av rettede grafer, et sett noder som er lenket sammen med rettede kanter [1]. I motsetning til vanlig rette grafer kan kantene i en RDF-graf være noder innenfor grafen.

RDF-graf benytter seg av tripler for å representere relasjonene mellom ressurser i form av subjekt, predikat og objekt. Subjektene og objektene i en slik graf utgjør nodene, mens predikatene sier hvordan et subjekt er relatert til et objekt. Noder som inngår i et trippel kan inngå i andre tripler hvor subjektene kan være objekter og objektene kan være subjekter.

I RDF er det subjektene i tripler som blir beskrevet og er enten representet som navngitte ressurser ved å benytte seg av URI'er eller som blanke noder. Predikatene uttrykker relasjone mellom et subjekt og et objekt og er alltid representer i form av en URI, mens objektet i et trippel enten kan forekomme som en navngitt ressurs i form av en URI, som en blank node, eller i form

av literaler. Ressursen som beskrives kan svare til alle ting som kan tydelig identifiseres i en gitt kontekst. Dette kan for eksempel være filmer, bøker, tog eller personer.

Ta setningen "Den Innerste Sirkel er en film". For å uttrykke en slik setning i form av RDF må man først identifisere de forskjellige bestanddelene. Det som beskrives innenfor setninger er "Den innerste Sirkel" og utgjør subjektet. Objektet i setningen er hva subjektet er og vi ser at objektet er en "film". Predikatet som kobler subjektet og objektet sammen er da "er en". Dette kan uttrykkes i form av en RDF-graf som vist i figur-3.1.



Figur 3.1: Grafrepresentasjon av setningen "Den Innerste Sirkel er en film"

URI'er

For å entydig identifisere ressurser på den semantiske veven benyttes URI'er, som er en generell form for å identifisere og navngi ressurser eller data. I motsetning til Uniform Resource Locators (URL) er ikke URI'er begrenset til å identifisere ting som har en posisjon innenfor et nettverk[14]. Ressurser lokalisert på veven kan derfor benytte seg av URL'er, mens URI'er kan benyttes til dette samtidig for å identifiserer andre typer ressurser i form av for eksempel bøker og e-postadresse. I RDF-grafer har man behov for å entydig identifisere ressurser slik at de kan bli referert av andre ressurser innenfor en graf, eller lenkes mot eksterne ressurser [21].

```
scheme:[//authority]path[?query][#fragment]
```

Figur 3.2: Konstruksjonsskjema for URI'er[1]. Delene innenfor brakketer regner som valgfrie.

- *scheme* – skjema klassifiserer typen URI. Dette kan for eksempel være ftp, http, urn, file, mailto[1].
- *authority* – viser som regel til et domenenavn[1]. I figur-3.8 benytter vi oss av domenet `www.example.com`.

- *path* – en hirarkisk oppbygning av en bane separert med /. Dette er hovedbestanddelen i en URI men kan i noen tilfeller være blanke (for eksempel når man benytter seg av e-postaddresser)[1].

- *?query* – benyttes som regel til å angi parametere til for eksempel vevtjenere[1].

- *fragment* – referer som regel til en del av et dokument eller vevressurs[1].

Et generellt problem ved identifisering av ressurser er at disse nødvendigvis ikke er uniformt identifisert innenfor forskjellige dokumenter eller datasett. Et eksempel på dette kan være to forskjellige datasett som begge inneholder filmer. Hvis man i det ene datasettet identifiserer skuespilleren "Morgan Freeman" som "morgan", og i det andre identifiserer "Morgan Spurlock" som "morgan" vil en sammenslåing av disse datasettene føre til en ressurs som beskriver begge skuespillerene.

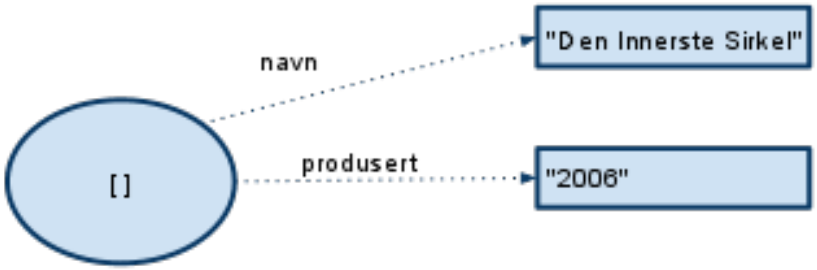
Domenekontoll er en viktig del av utviklingen av den semantiske veven, og bruk av URI'er som navngivning av ressurser. Ved å benytte seg av egne domener for å beskrive egne ressurser har man en større kontroll over ressursene man identifiserer. Dette forutsetter da at ingen benytter seg av domener de selv ikke har kontroll over. URI'er muliggjør enkel utvikling og publisering for bruk på den semantiske veven. En person kan innenfor eget domene utvikle og publiserer ressurser som henviser til konsepter referert og navngitt ut ifra egene URI'er.

Egenskaper ved URI gjør dem nyttige for å representere og navngi ressurser på den semantiske veven. Dette kan for eksempel være at oppslag på en URI burde føre til et dokument som inneholder en beskrivelse av ressursen navngitt av URI'en så lenge denne ressursen er lokalisert på veven. Retningslinjene til Tim Berners-Lee om lenkede data tilsier at vevbaserte URI'er benyttes for å navngi ressurser på den semantiske veven. Oppslag som fører til beskrivelse av navngitte ressurser etablerer en felles forståelse for hva en disse ressursene beskriver.

Blanke noder

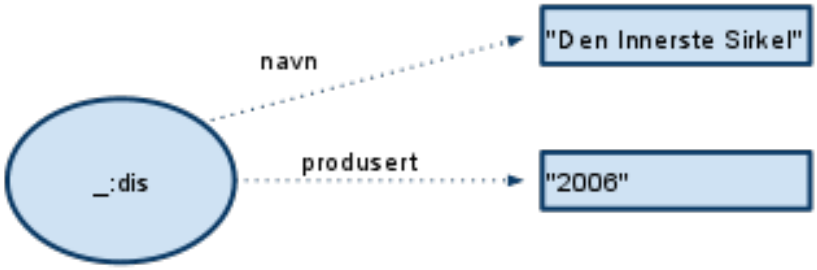
I noen tilfeller vil det være behov for å representere noder i en RDF-graf som ressurser som ikke referere til noen spesifikk URI. Dette kan for eksempel være være som vist i figur-3.3 hvor en film blir representert ved en blank node. Blanke noder blir som regel benyttet i de tilfeller hvor ressurser ikke er ment å bli beskrevet eksplisitt eller hvor man ikke har noen kjent URI [1]. Blanke noder kan også være nyttige i de tilfellene man ønsker å gruppere

andre noder i en graf. Blanke noder kan i en RDF-graf kun opptre som noder og ikke kanter.



Figur 3.3: Ikke navngitte blanke noder

Ofte vil det være behov for å kunne referere til blanke noder innenfor det datasettet de er definert. Dette gjøres ved at man gir de blanke noden et navn, en id, som man vil kunne referere til andre steder i et datasett. Slike noder er unike innenfor datasettet de er definert, men kan ikke benyttes som sterke identifikatorer utenfor dette. I figur-3.4 representeres en film som i figur-3.3, men ved hjelp av en navngitt blank node. Navngitte blanke noder defineres **_:** etterfulgt av navnet man ønsker å benytte for den blanke noden.

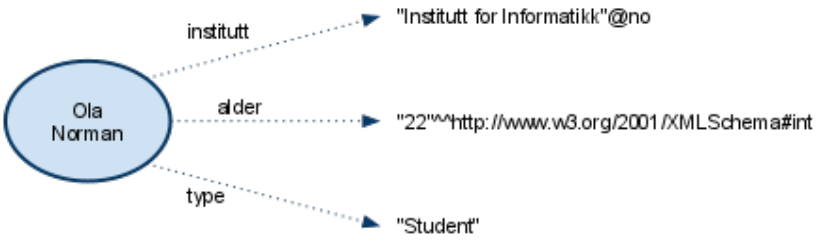


Figur 3.4: Navngitt blank node

Blanke noder kan bli representert uten bruk av navn og benyttes i de tilfellene man ønsker å gruppere uttrykk eller ikke skal referere til noden andre steder innenfor en RDF-graf. For å definere ikke navngitte blanke noder benytter man seg av tegnene [og]. Alle uttrykk som er innenfor dette vil da bli knyttet til den blanke noden i RDF-grafen [17]. I figur-3.3 benytter vi oss av en blank node som ikke er navngitt. Hvordan navngitte blank noder og ikke navngitte blanke noder er representert her følger Turtle Serialisering (se kapittel 3.1.2)

Literaler

RDF benytter seg av literaler for å uttrykke verdier om et subjekt som for eksempel navn og datoer. Literaler kan ta tre former i en RDF-graf. De kan være typede, noe som vil si at man legger ved en datatype til literalene for å definerer hva slags data de inneholder. I figur-3.5 benytter vi oss av datatypen "int" for å si at alderen til "Ola Norman"er et tall. Hvis literalet ikke har noen tilknyttet datatype kan vi sette en språkkode som definerer hvilket språk innholdet av literalet er på. I figur-3.5 definerer vi at instituttet "Ola Norman" tilhører er skrevet på norsk. Kodene benyttet for å angi språket på innholdet i et literal hentes fra ISO 639[1]. Literaler som ikke har språkkode eller datatype regnes som utypede literaler hvor innholdet regnes som tekststrenger. I figur-3.5 vises hvordan "Ola Norman" er tilknyttet instituttet i form av en utypet literal[21]. I motsettning til blanke og navngitte noder er literaler terminale. Dette vil si at man ikke kan knytte ekstra informasjon til literalene utenom språk eller datatype, og at literaler aldri kan forekomme som subjekt i et trippel.

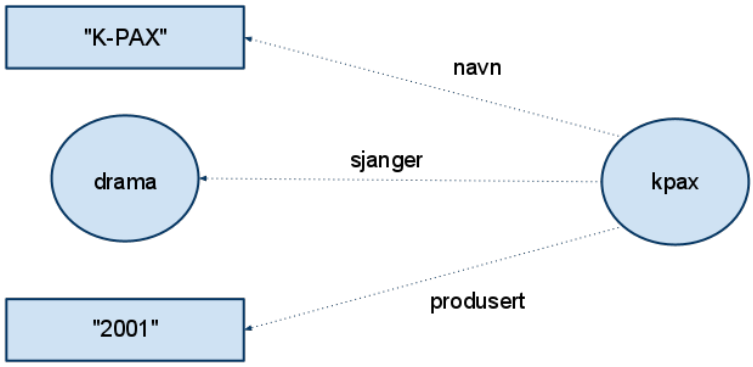


Figur 3.5: Literaler representert i en graf

3.1.2 Serialisering

Visuell representasjon for RDF-grafer som vist i figur-3.6, er enkle for mennesker å forstå, men er uegnet for utveksling av data mellom systemer og applikasjoner. Serialisering av RDF-grafer til dokumenter bestående av tripler i maskinprosesserbare formater er derfor nødvendig.

[1]Kodet representasjon for navn på språk – `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22109`



Figur 3.6: Grafrepresentasjon av informasjon om filmen "K-PAX"

RDF/XML

Det finnes flere former for RDF-serialisering. RDF/XML er et eksempel på dette som benytter seg av den godt definerte XML syntaksen for å representere RDF-grafer. Det finnes gode applikasjoner og verktøy for behandling av informasjon lagret i XML og formatet egner seg derfor godt til behandling av strukturert data. RDF/XML har av denne grunnen blitt valg som den offisielle RDF serialiseringsmetoden. På denne måten forsikrer man seg om at et serialiseringsformat alltid vil være støttet av RDF verktøy.

Det er flere ulempler ved å benytte seg av RDF/XML-serialisering av RDF. Den er syntaktisk kompleks og kan ha flere forskjellige representasjonsformer for en og samme ting, noe som medfører at grafer representert i RDF/XML kan være vanskelige å validere. I tillegg representers XML strukturert data som trær noe som ikke gjør den til en velegnet representasjonsform for RDF datasett, hvor data representeres i grafer.RDF/XML benytter seg av mange attributter og "tags" som for eksempel "rdf:about" for navngivning av ressurser, hvor disse ikke er en del av RDF i seg selv[17]. Dette medfører at RDF/XML-serialisering ikke er godt egnet for å skrive RDF-noder for hånd. Figur-3.7 viset et eksempel for RDF/XML-serialisering.

Terse RDF Triple Language

Terse RDF Triple Language (Turtle) er et serialiseringsformat for RDF. I motsetning til RDF/XML er Turtle en serialiseringform spesiallaget for å representere RDF og er mer velegnet for menneskelig lesning. Ettersom Turtle ikke trenger å benytte seg av den allerede eksisterende XML syntaksen trenger man ikke å representere grafer som trær [17]. Figur-3.8 viser den samme grafen som representert i figur-3.7 uttrykket med turtle-serialisering. Turtles lettleselige og forståelige syntaks ligger også til grunn for syntaksen

```
<?xml version="1.0"?>
 <rdf:RDF xmlns:ex="http://www.example.com/Film/" xmlns:rdf
   ="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <ex:Film>
   <ex:navn>Den Innerste Sirkel</ex:navn>
   <ex:produsert>2006</ex:produsert>
   <ex:sjanger rdf:resource="http://www.example.com/Film/
     thriller" />
   <ex:orginalnavn>The Good Shepherd</ex:orginalnavn>
  </ex:Film>
  <ex:Film>
   <ex:navn>Haisommer</ex:navn>
   <ex:produsert>1975</ex:produsert>
   <ex:sjanger rdf:resource="http://www.example.com/Film/
     thriller" />
   <ex:orginalnavn>Jaws</ex:orginalnavn>
  </ex:Film>
  <ex:Film>
   <ex:navn>K-PAX</ex:navn>
   <ex:produsert>2001</ex:produsert>
   <ex:sjanger>
    <ex:Sjanger rdf:about="http://www.example.com/Film/
      drama">
     <ex:navn>Drama</ex:navn>
    </ex:Sjanger>
  </ex:sjanger>
 </ex:Film>
 <ex:Sjanger rdf:about="http://www.example.com/Film/triller
   ">
  <ex:navn>Thriller</ex:navn>
 </ex:Sjanger>
</rdf:RDF>
```

Figur 3.7: RDF/XML Serialisering

benyttet av SPARQL (se kapittel 3.2). Turtle er en forenkling av Notation3[2] som er laget for å være lettleselig for mennesker og alle RDF-grafer skrevet i Turtle-syntaks er også gyldige ifølge Notation3. Serialiseringsformatet er derimot ikke en standard, men er en av de vanligste formatene for å representere RDF-grafer og regnes som en de facto standard.

[2]http://www.w3.org/DesignIssues/Notation3.html

Turtle benytter seg av et enkelt format for hvert trippel i en RDF graf, hvor et trippel refererer til en nodes relasjon til et objekt eller literal. Subjektet, predikatene og objektene skrives på en linje separert med mellomrom og termineres med punktum. Hvis man skal uttrykke flere tripler som innebærer det samme subjektet behøver man ikke å skrive subjektene flere ganger enn en. Istedet benytter man seg av et semikolon. Linjen etter semikolon vil da kun inneholde predikat og objekt som skal knyttes mot subjektet. Det siste predikatet og objektet som knyttes opp mot et subjekt avsluttes med et punktum [17]. Se figur-3.8 for hvordan bygge opp subjekter i turtle-serialisering.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
  .
@prefix ex: <http://www.example.com/Film/> .

[] rdf:type ex:Film ;
   ex:navn "Den Innerste Sirkel" ;
   ex:produsert "2006" ;
   ex:sjanger ex:thriller ;
   ex:orginalnavn "The Good Shepherd" .

[] rdf:type ex:Film ;
   ex:navn "Haisommer" ;
   ex:produsert "1975" ;
   ex:sjanger ex:thriller ;
   ex:orginalnavn "Jaws" .

[] rdf:type ex:Film ;
   ex:navn "K-PAX" ;
   ex:produsert "2001" ;
   ex:sjanger ex:drama .

ex:triller rdf:type ex:Sjanger ;
  ex:navn "Thriller" .

ex:drama rdf:type ex:Sjanger ;
  ex:navn "Drama" .
```

Figur 3.8: Turtle serialisering

3.1.3 Vokabularer

Vokabularer i RDF er URI'er som er samlet under et felles domene og benyttes for å uttrykke bestemte formål.

I RDF henviserer vokabularer til en samling av navn med tydelig definert betydning[1]. Det er vanlig å benytte seg av slike vokabularer for navngivning og identifisering av ressurser i RDF. På denne måten unngåes det å konstruere navn på ressurser som allerede er tydelig identifisert og beskrevet andre steder på den semantiske veven.

Det eksisterer en rekke vokabularer som dekker flere forskjellige interesseområder og beskriver ofte anvendte ressurser. Vokabularet `http://www.w3.org/1999/01/222-rdf-syntax-ns#` er et eksempel på et godt definert vokabular som for eksempel inneholder ressursen `http://www.w3.org/1999/01/222-rdf-syntax-ns#type`. Denne ressursern benyttes som predikat for å definere hvilken klasse et subjekt tilhører, hvor objektet i triplet definerer klassen. Andre velkjente vokabularer er for eksempel DublinCore (se kapittel 3.3.1) for metadata og Friend of a Friend[3] som benyttes til å beskrive mennesker og relasjoner mellom dem igjennom veldefinerte predikater.

3.1.4 Navnerom

Generelt sett er et navnerom en URI som representerer den delen av URI'er alle elementerne i et vokabular har til felles (Se kapittel 3.1.3), Navnerom opptrer følgelig som en markør for et vokabular. I serialisering av RDF er det vanlig å definere prefikser som representerer navnerom for distinkte deler av URI'er. Et eksempel på et navnerom kan være `http://rdfs.org/ns/void#` som inneholder vokabularer for Vocabulary of Interlinked Datasets (VoID, se kapittel 3.3.2). Navnene som er innhold i dette vokabularet er da URI'er hvor dette navnerommet forekommer som prefiks. Ved å benytte seg av navnerom kan man i RDF serialisering ungå å skrive lange URI'er. I Turtle serialisering gjøres dette ved å slå benytte seg av nøkkelorder "@prefiksen" ut ifra skjemaet vist i figur-3.9. [kortnavn] byttes ut med navnet man ønsker å benytte seg av og [navnerom] angir URI'en for navnerommet.

```
@prefix [kortnavn]: <[navnerom]> .
```

Figur 3.9: Definering av prefiks i Turtle syntaks

Navnerom har ingen innvirkning på gyldigheten av et dokument hvor det forekommer, men utgjør stor forskjell i de tilfellene RDF-grafer blir

[3] `http://www.foaf-project.org/`

konstruert av personer og ikke maskiner. I figur-3.8 benyttes for eksempel navnerommet "rdf" som hensviser til vokabularet gitt av URI'en `http://www.w3.org/1999/01/222-rdf-syntax-ns#`. For å benytte seg av navn innenfor denne URI'en benyttes en konkatinering av navnet til navnerommet, ":" og navnet innenfor vokabularet man skal benytte. Dette kan for eksempel være "rdf:type".

3.2 SPARQL

SPARQL Protocol[19] and RDF Query Language[20] (SPARQL) er en relativt ung standard. Spørrespråket SPARQL brukes for å stille spørringer mot RDF data, mens SPARQL som RDF protkoll er en måte å overføre SPARQL spørringer fra en spørreklient til en spørreprosessor eller et endepunkt. SPARQL brukes for å evaulere spørringer mot RDF-grafer, og benytter seg av enkle RDF-grafer for å bygge opp spørregrafene. Disse spørregrafene er i stor grad representert ved hjelp av Turtle-syntax. Til forskjell fra Turtle kan man benytte seg av variabler og filtre som er en del av serialiseringsformatet. SPARQL består også av et XML basert presentasjonsformat for spørringsresultater [1].

I tillegg til SPARQL finnes det flere andre spørrespråk for å evaluere spørringer mot RDF. Dette er for eksempel RDF Data Query Language (RDQL)[4] og Sesame RDF Query Language (SRQL)[5], men ettersom SPARQL er en W3C standard beskrives kun dette videre i kapittelet[17]. Videre i kapitlet vil de sentrale bestanddelene av SPARQL spørrespråket bli gjennomgått.

3.2.1 SELECT spørringer

SELECT spørringer representerer den generelle formen for SPARQL-spørringer. Uttrykkene som følger etter nøkkelorder SELECT er navn på variabler som skal bli hentet ut fra RDF-grafen man benytter spørringen på. I figur-3.10 er vi ute etter verdiene for variablene "?navn" og "?produsert". Selve spørringen defineres i WHERE delen og danner et grafmønster innenfor krøllparantesene {} [1]. Grafmønsteret består av tripler hvor hver del av bestanddel av triplet kan byttes ut med variabeldefinisjoner. Variabeldefinisjoner i SPARQL markeres ved et spørsmålstegn "?" etterfulgt av navnet man ønsker variabelen skal ha. Hvis grafmønsteret definert for

[4] `http://www.w3.org/Submission/RDQL/`

[5] `http://www.openrdf.org/doc/sesame2/users/ch09.html`

```
PREFIX ex: <http://www.example.com/Film/> .
PREFIX rdf: <url> .
SELECT ?navn ?produsert
WHERE {
  ?x rdf:type ex:Film .
  ?x ex:navn ?navn .
  ?x ex:produsert ?produsert .
}
```

Figur 3.10: SELECT spørring

?navn	?produsert
"K-PAX"	"2001"
"Den Innerste Sirkel"	"2006"
"Haisommer"	"1975"

Tabell 3.1: Resultat fra spørringen i figur-3.10 kjørt mot RDF grafen vist i figur-3.8

WHERE treffer en del av RDF-grafen man evaluerer spørringen imot vil dette resultatet bli tatt med i det endelige resultatsettet.

I figur-3.10 er et eksempel på hvordan en enkel SELECT spørring kan se ut. Vi definerer her først to prefikser, "ex" og "rdf". Prefikser benyttes for å unngå fulle URI'er hver gang man refererer til ressurser og angis i SPARQL spørringer ved å benytte seg av nøkkelordet PREFIX. Variablene benyttet i SELECT delen, "?navn" og "?produsert", hentes ut fra resultatene som samsvarer med grafmønsteret definert i WHERE delen. Grafmønsteret benyttet i WHERE i figur-3.10 består av tre tripler. Først definerer vi "?x" til å være et subjekt som må være av typen "ex:Film". Når en ressurs i RDF-grafen man spør mot samsvarer med dette vil "?x" bli bundet til denne ressursen. For videre å hente ut variablene "?navn" og "?prod" benytter vi oss av variablen "?x". Hvis "?x" har en verdi for predikatet "ex:navn" vil denne verdien bli bundet til variabelen "?navn", mens variabelen "?produsert" blir bundet til verdien av predikatet "ex:produsert" hvis "?x" har en verdi for dette. Hvis alle bestanddelene av grafmønsteret i WHERE delen blir bundet, blir resultatet av dette tatt med i det endelige resultatsettet. Tabell-3.1 viser resultatet fra spørringen i figur-3.10 kjørt mot RDF-grafen vist i figur-3.8

Bundne variabler er variabler som har en verdi. I tabell-3.12 ser vi for eksempel at variabelen i et resultat er bunnet til literalet "K-PAX" mens variabelen "?originalnavn" ikke er bundet.

```
{
 { ?film rdf:type ex:Film .
   ?film ex:navn ?navn . }
 { }
 ?film ex:publisert ?publisert .
}
```

Figur 3.11: Undermønstre i SPARQL-spørringer

3.2.2 Komplekse grafmønstre

RDF-grafer følger ikke alltid et fastsatt skjema. Det kan hende at variabler man er ute etter forekommer for noen treff men er ikke representert andre steder i RDF-grafen. Det kan også hende at variabler er representert på en litt forskjellig måte forskjellige steder innenfor en RDF-graf. I slike tilfeller hvor en variabel man er ute etter ikke eksisterer, vil man miste dette resultatet ettersom variablen ikke blir bundet. For å løse denne typen problemer kan man benytte seg av nøkkelordene OPTIONAL og UNION.

Undermønstre Undermønstre kan benyttes i grafmønstre. Disse mønstrene er avgrenset av krøllparanteser, {}, og er definert innenfor det allerede eksisterende grafmønsteret i WHERE delen av en spørring. Slike undermønstre kan bli brukt til å innføre restriksjoner på kun deler av det totale grafmønster i WHERE [1]. Figur-3.11 viser hvordan man kan benytte seg av undermønstre. Tomme undermønstre har ingen innvirkning på SPARQL-spørringer.

OPTIONAL I de tilfellene man ikke er sikker på om en variabel er med i RDF-grafen eller ikke, men vil ha den med i resultatet hvis den er tilstede, kan man benytte seg av nøkkelordet OPTIONAL. Hvis variabler som er definert innenfor et OPTIONAL grafmønster finnes i RDF-grafen vil verdien av denne variabelen bli tatt med i det endelig resultatet. Hvis dette ikke er tilfelle vil istedet variabelen få en tom verdi. Figur-3.12 viser en eksempel SPARQL SELECT spørring som benytter seg av OPTIONAL for å hente ut verdien for predikatet "ex:navn" og verdien for predikatet "ex:originalnavn" hvis dette er tilgjengelig.

Resultatet av denne spørringen kjørt mot RDF-grafen i figur-3.8 vil føre til et resultat som vist i tabell-3.2. Vi ser her at to av resultatene har verdier for predikatet "ex:originalnavn", mens den siste får en tom verdi for denne variablen.

```
PREFIX ex: <http://www.example.com/movies/> .
PREFIX rdf: <http://www.example.com/movies/> .
SELECT ?navn, ?originalnavn
WHERE
{
 ?x rdf:type ex:Film ;
    ex:navn ?navn .
 OPTIONAL { ?x ex:originalnavn ?originalnavn }
}
```

Figur 3.12: SELECT spørring med OPTIONAL

navn	original
"Den Innerste Sirkel"	"The Good Shepherd"
"Haisommer"	"Jaws"
"K-PAX"	

Tabell 3.2: Resultat fra spørringen i figur-3.12

UNION For å definere alternative mønstre kan man benytte seg av nøkkelordet UNION. Dette nøkkelordet benyttes for å slå sammen resultatet av flere mønstre, hvor minst et av mønstrene som inngår i UNION må eksistere i RDF-grafen man spør mot for at resultatet skal være med i det endelige resultatsettet [1]. I figur-3.13 viser et eksempel på bruk av UNION hvor navnet på en film bindes til verdien av "ex:originalNavn" og/eller "ex:navn". Hvis både verdier for variablene "?navn" og "?originalnavn" er knyttet til samme subjekt vil vil spørringen returnerer to resultater, en for hver av variablene.

```
PREFIX ex: <http://www.example.com/movies/> .
PREFIX rdf: <http://www.example.com/movies/> .
SELECT ?navn, ?produsert
WHERE
{
  ?x rdf:type ex:Film .
  { ?x ex:originalnavn ?navn . } UNION
  { ?x ex:navn ?navn . }
}
```

Figur 3.13: SELECT spørring med UNION

3.2.3 Filtrering av resultater

I mange tilfeller vil det for mange praktiske operasjoner ikke være nok å benytte seg av metodene beskrevet tidligere i kapitelet. Noen ganger vil det være nyttig å kunne hente ut data fra RDF-grafer som samsvarer med gitte restriksjoner. Dette kan for eksempel være å kun hente ut resultater hvor verdien av en variabel ligger innenfor et gitt spekter. For å angi restiksjoner benyttes nøkkelordet FILTER. FILTER angir restriksjoner for hele grafmønsteret hvor nøkkelordet er definert. Under er eksempler på noen filtreringsmetoder [1].

- For å undersøke om en variabel er lik eller ulik en fast verdi eller en annen variabel.

```
{
  ?film ex:name ?name .
  FILTER (?name = "Haisommer")
}
```

- For å ta med resultater som kun har verdier for en variabel større enn Ã©n gitt verdi.

```
{
  ?film ex:produsert ?prod .
  FILTER (?prod <= 2000)
}
```

- Sannhetsfunksjonen "isURI" benyttes for å teste om en variable er en URI eller ikke.

```
{
  ?film ex:name ?name .
  FILTER (isURI(?film))
}
```

- For å teste om en tekststreng samsvarer med et regulært uttrykk kan man benytte seg av sannhetsfunksjonen "REGEX". Hvis det regulære uttrykket er tilstede i variabelen man tester mot svarer filteret ja.

```
{
  ?film ex:name ?name .
  FILTER (REGEX(?name, "Haisom")
}
```

Man kan benytte seg av vilkårlig mange filtreringsvilkår innenfor et filtreringsmønster ved å benytte seg av logisk konjunksjon (&&) og disjunksjon (||). Negasjon (!) benyttes i de tilfellene man ønsker å filtrere bort resultater. I tillegg til dette kan man også benytte seg av aritmetiske operasjoner for å kombinerer forskjellige verdier (+, -, /, *) [1].

3.2.4 Vanlige SPARQL modifikatorer

En modifikator er et argument man kan gi en spørring som ikke påvirker spørringen logisk, men istedet formen og størrelse på et resultatsett av en evaluert spørring. En rekke forskjellige modifikatorer kan benyttes for å begrense det endelige resultatet av SPARQL-spørringer. Dette kan være spesielt nyttig med desentraliserte vevmiljøer, hvor man ikke kan være sikre på hvor store resultatsett man får returnert fra nett-tjeneren når man sender en spørring. SPARQL inkluderer derfor modifikatorer for å kontrollere detaljer om form og størrelse på hva som skal returneres i resultatsettene [1].

Sortering av resultatsett

Ofte vil det være nyttig å få data fra spørringer sortert etter en gitt rekkefølge. For eksempel tilfeller hvor man ønsker kun å prosessere en del av et datasett, kan det være lønnsomt at dataene er sortert slik at man får resultater som ligger nære hverandre etter sorteringskriteriene [1]. Ved hjelp av modifikatoren ORDER BY kan man sortere etter en eller flere kriterier i forskjellige rekkefølger angitt av sorteringsmodifikatorene ASC og DESC [17]. Hvis man ikke benytter seg av ASC eller DESC blir en standard sortering benyttet for variablen. URI'er blir her betraktet som literaler typet til "xsd:string" (tekststrenger). Figur-3.14 viser et eksempel på sortering ut ifra forskjellige verdier i forskjellige rekkefølger.

ASC - Når man bruker sorteringsmodifikatorene ASC på tekststrenger vil den sorterte rekkefølgen bli alfabetisk. Mot tallverdier blir rekkefølgen økende.

DESC - Sorteringsmodifikatoren DESC benyttet mot en tekststreng vil føre til en reversert alfabetisk sortering. Mot tallverdier blir rekkefølgen synkende.

Sortering av resultatsett trenger ikke å basere seg på variabler som er med i det endelige resultatsettet. Man kan istedet benytte seg av variabler som bare blir definert i grafmønsteret i WHERE delen av spørringen.

```
PREFIX ex: <http://www.example.com/movies/> .
PREFIX rdf: <http://www.example.com/movies/> .
SELECT ?navn, ?produsert
WHERE {
  ?x rdf:type ex:Film ;
     ex:produsert ?produsert ;
     ex:navn ?navn .
} ORDER BY DESC(?produsert) ASC(?navn)
```

Figur 3.14: SELECT spørring med ORDER BY

```
SELECT DISTINCT *
WHERE
{
 ?s ?p ?o .
} ORDER BY ?s LIMIT 10 OFFSET 100
```

Figur 3.15: SELECT spørring LIMIT og OFFSET

Utdrag fra resultatsett

En viktig del av spørrespråk er å kunne velge ut kun en del av resultatsettet fra en spørring. I SPARQL er dette gjennomført ved modifikatorene LIMIT og OFFSET. Nøkkelordene tillater at man kan få tilbake en delmengde av et resultatsett. LIMIT tilsier hvor mange resultater resultatsettet skal inneholde, mens OFFSET sier hvor mange resultater man skal se bort ifra før man begynner bygger opp resultatsettet [1],

Eksemplet i figure-3.15 vil resultere i et datasett som inneholder 10 resultater hvor første resultat kommer etter man har sett bort fra 100 resultater, og resultatsettet er sortert etter subjektet i triplene.

Distinkte rader

En annen måte for å produsere mer håndterbare resultatsett ut ifra en spørring er å luke ut rader som blir gjentatt i resultatsettet. For å oppnå dette kan man benytte seg av nøkkelordet DISTINCT som settes foran variablene man ønsker å hente ut i SELECT delen av en spørring [1]. I figur-3.15 vil alle triplene kun forekomme en gang. Hvis man kjører den samme spørringen uten DISTINCT vil man ikke kunne vite på forhånd om alle radene som returneres er distinkte eller ikke uten å undersøke resultatsettet.

Det er både positive og negative sider ved å benytte seg av DISTINCT.

Det positive er at vi kan fjerne duplikater og minske båndbredden[6] benyttet av endepunkter ettersom resultatsettet returnert inneholder mindre data. Det negative er at man kan skjule dårlig formaterte spørringer som kan medføre en økt eksekveringstid for å få et resultatsett fra et SPARQL-endepunt, samt man får ikke noen oversikt over hvor mye data faktisk funnet når spørringen blir evaluert mot en RDF-graf [17].

3.2.5 Navngitte grafer

En spørring i SPARQL eksekveres mot en RDF-graf og ingen eller flere navngitte RDF-grafer som alle er identifisert av URI'er. Dette muliggjør at en spørring kan evalueres på et sett av forskjelige RDF-grafer, RDF-datasett[17].

Ved å benytte seg av navngitte RDF-grafer kan man avgrense mengden data en spørring blir benyttet mot. Man vil da kun benytte spørremønstre mot de navngitte RDF-grafene og ikke alle RDF-grafer som er tilgjengelige i RDF-datasettet, noe som vil kunne føre til en reduksjon i tiden brukt for å eksekvere en spørring. Nøkkelorder GRAPH benyttes for å angi hvilken RDF-graf et grafmønster skal evalueres mot. Figur-3.16 viser et enkelt eksempel hvor en spørring blir kjørt mot RDF-grafen angitt av URI'en `http://www.mindswap.org/2004/owl/mindswappers`.

```
SELECT *
WHERE
{
  GRAPH <http://www.mindswap.org/2004/owl/mindswappers> {
    ?s ?p ?o .
  }
}
```

Figur 3.16: SELECT spørring utført mot en navngitt graf

3.2.6 Andre typer SPARQL spørringer

SELECT-spørringer er ikke den eneste måten man kan kjøre spørringer mot RDF-grafer i SPARQL. Under er en kort beskrivelse av andre metoder, i tillegg til eksempler for å demonstrere hvordan de kan se ut.

- ASK er en boolsk spørretype som benyttes for å stille spørsmål mot en RDF-graf. ASK returnerer en sannhetsverdi (ja eller nei), alt ettersom

[6]Med båndbredde menes den tilgjengelige overføringskapasiteten man kan benytte seg for overføring av data mellom lokasjoner over et nettverk.

mønsteret man definerer i WHERE delen er representert i RDF-grafen man spør mot eller ikke. Denne typen spørringer er nyttige og raske, ettersom SPARQL spørringen blir stoppet så fort den har mulighet til å komme med et svar [17]. ASK spørringen under returnerer nei kjørt mot RDF-grafen i figur-3.8, og ja hvis vi heller hadde spurt etter navnet "Den Innerste Sirkel".

```
PREFIX ex: <http://www.example.com/Filmer/> .
ASK
WHERE
{
  ?film ex:navn "K-PAX" .
}
```

- CONSTRUCT benyttes for å lage nye RDF grafer basert på et mønster definert av CONSTRUCT nøkkelordet. Variablene benyttet for å bygge den nye RDF-grafen blir definert i grafmønsteret i WHERE delen på samme måte som ved SELECT-spørringer [17]. Eksemplet under viser hvordan en CONSTRUCT spørring kan se ut.

```
PREFIX ex: <http://www.example.com/Filmer/> .
PREFIX rdf: <url> .
CONSTRUCT
{
  [] rdf:type ex:Movie ;
       ex:name ?navn ;
} WHERE {
  ?film rdf:type ex:Film ;
        ex:originalnavn ?navn ;
}
```

Denne spørringen utført mot RDF-grafen vist i figur-3.8 vil returnere en RDF-graf lik den vist under.

```
@prefix ex: <http://www.example.com/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
  ns#> .

 [] rdf:type ex:Movie ;
    ex:name "Jaws" ;

 [] rdf:type ex:Movie ;
    ex:name "The Good Shepherd" ;
```

- DESCRIBE returnerer en RDF-graf om en ressurs man ber om å få beskrevet. Returdataene er ikke fastsatt ut ifra en SPARQL-spørring hvor klienten må vite noe om strukturen av RDF-datasettet, men er istedet bestemt av SPARQL endepunktet ettersom svaret på denne typen spørringer ikke er definert i SPARQL spesifikasjonen [20]. På denne måten trenger ikke klienten å vite hvordan RDF-grafen man spør mot ser ut. Grafmønsteret i WHERE delen benyttes av endepunktene som restriksjon på hvordan data skal returneres [17]. Under er et eksempel på hvordan en DESCRIBE spørring kan se ut.

```
DESCRIBE *
WHERE
{
  ?s ?p ?o .
}
```

  Ettersom svaret på en DESCRIBE spørring ikke er definert i SPARQL spesifikasjonen kan det variere fra endepunkt til endepunkt. Standarden har i den senere tid blitt at man returerer en beskrivese av en ressurs hvis denne er bundet. Hvis man treffer en ikke bundet ressurs, i.e. en blank node, vil det returneres en beskrivelse av neste tilgjengelige bundende ressurs i grafen hvor den ikke bundede ressursen er subjekt. Dette er metoder benyttets av for eksempel DBpedia.

3.2.7 SPARQL endepunkter

SPARQL endepunkter er en tjeneste som tar i mot og evaluerer spørringer uttrykket i form av SPARQL spørrespråket. Endepunktene tillater personer og maskiner å sende spørringer mot RDF-datasett, som ofte representerer et spesifisert domene, hvor resultatet av spørringen returneres ut ifra SPARQL spørreformen. SELECT spørringer vil føre til at data returneres i form av XML, mens CONSTRUCT spørringer returneres i from av RDF-grafer. SPARQL endepunkter følger spessifikasjonene gitt av SPARQL protokollen.

Det eksistere flere tilgjengelige endepunkter som for eksempel DBpedia[7], `data.gov.uk`[8] og voiD store[9] (se kapittel 3.3.2) [17].

[7] `dbpedia.org/sparql` – endepunk for å sende spørringer mot RDF strukturert data fra wikipedia

[8] `data.gov.uk/sparql` – endepunkt for å evaluere spørringer mot åpne offentlige data i Storbritania

[9] `void.rkbexplorer.com/sparql` – endepunkt for å finne informasjon om datasett beskrevet gjennom VoID

3.3 Metadata

Metadata kan defineres som data som gir informasjon om en eller flere aspekter ved data. Dette kan blant annet være meningen bak dataene eller hvem som har publisert dataene. Det er ikke noe absolutt skille mellom metadata og vanlig data, hvor metadata i et dokument kan være data innenfor et annet dokument. Det er ingen begrensning på hvordan metadata skal se ut og kan benyttes til å beskrive alt fra mennesker og konsepter til å beskrive annen metadata [17].

Målet med metadata i dokumenter på veven er å beskrive innholdet i et maskinlesbart format, som gir ekstra informasjon om hvordan dataene kan benyttes av andre en de som produserte dokumentet . For å gi andre mulighet til å vurdere kvaliteten av publisert data og evaluere om de vil stole på innholdet i et datasett, burde enkel proveniens metadata som hvem som har publisert dokumentet, hvordan det ble opprettet og når det ble publisert være tilstede [16, 22]. I forhold til gjenbruk av åpne data vil det være nyttig å publiserer metadata som gir et vist inntrykk av hva et datasett inneholder og hvordan et datasett eventuelt er lenket sammen med andre datasett. Ved å kombinere metadata og data reduserer man problemene som ofte oppstår når data blir gjenbrukt, ettersom betydningen av data og forholdet mellom data ikke forandres.

3.3.1 Enkel proveniens metadata, Dublin Core terms

Metadata som beskriver opprinnelse av et dokument er viktig for forståelsen av innholdet i et dokument. The Dublin Core Metadata Initiative er et prosjekt med mål om å lage metadatastandarder for en rekke applikasjoner [17, 23]. Hovedsakelig benyttes standardene mot ressurser knyttet til veven, men er ikke begrenset til dette. Vokabularet Dublin Core er en ofte brukt standard i RDF dokumenter og definerer et sett med metadatatermer for å assosiere metadata med ressurser og datasett. Man regner ofte Dublin Core som et sett av 15 forskjellige metadataelementer som beskriver opprinelse og eierhistorik for dokumenter. Ingen av disse termene er påkrevet og kan bli gjentatt flere ganger. I figur-3.17 benyttes Dublin Core for å legge ved metadata om tittel, beskrivelse, kilde og datoen datasettet sist var modifisert, til et datasett.

3.3.2 Vocabulary of Interlinked Datasets

Vocabulary of Interlinked Datasets (VoID) er et RDF-skjema vokabular for å uttrykke metadata om RDF-datasett. VoID definerer termer og praksiser

```
@prefix rdf:<http://www.w3c.org/1999/02/22-rdf-syntax-ns#>.
@prefix dc: <http://purl.org/dc/terms/> .
@prefix void: <http://rdfs.org/ns/void#> .

:dataset a void:Dataset;
    dc:title "Dataset";
    dc:description "Eksempel datasett";
    dc:source <http://example.org/data/>;
    dc:modified "2011-01-01"^^xsd:date;
```

Figur 3.17: VoID datasett og Dublin Core metadata

for å katalogisere og gi statistisk metadata om datasett, i tilleg til å gi informasjon om hvordan datasett er lenket sammen. VoID er ment som en bro mellom de som publiserer og de som benytter seg av RDF datasett og kan benyttes for å følgende forskjellige metadatastrukturer[16, 24].

- *Generell metadata* – benyttet for å gi generell metadata om datasett. Denne typen metadata hjelper potensielle brukere for å undersøke om et datasett er egnet til sitt bruk eller ikke. Generell metadata inkluderer informasjon som opprinnelse, eierhistorikk, publikasjonsdato, osv. Dublin Core egner seg godt som vokabular for å gi slik metadata.

- *Tilgangs metadata* – benyttes for å beskrive metoder for hvordan man får tilgang til RDF triplene i et datasett. Eksempelvis kan man benytte seg av predikatet "void:sparqlEndpoint" for å angi lokasjonen til et sparql hvor endepunkt datasettet er tilgjengelig.

- *Strukturell metadata* – benyttes for å angi metadata som beskriver strukturen på et datasett, som skjema og den interne strukturen på datasettet. Dette kan for eksempel være hvilke vokabularer som er benyttet i datasettet gitt av predikatet "void:vokabulary", eller gi eksempler på ressurser i et datasett ved å benytte seg av predikatet "void:exampleResource". Denne typen metadata kan være nyttig når man utforskjer eller kjører spørringer mot et datasett

- *Beskrive lenking av datasett* – VoID gir muligheten til å legge ved metadata som beskriver hvordan datasett er lenket sammen. Man kan også angi at et datasett er et subsett av et annet.

Figur-3.17 definerer generell metadata om et datasett ved hjelp av Dublin Core og VoID.

Kapittel 4

Konvertering av data til RDF

Mye data publisert på veven er tilgjengelig i formater som ikke er egnet til gjenbruk eller lenking. Tilgjengelig data finnes overalt i form av strukturert HTML eller XML dokumenter, dokumenter som inneholder tegneseparerte verdier, data lagret i relasjonsdatabaser eller data publisert i regneark. Data lagret i slike formater egner seg ikke til bruk på den semantiske veven ettersom man ikke kan trekke ut hvordan dataene i dokumentene er relatert til hverandre, og bestanddelene er ikke tilgjengelige gjennom referanser og må først transformeres til RDF. Denne transformeringen varierer i vanskelighetsgrad ut ifra hvor lukket formatet dokumentet er på og tilgjengeligheten av programmeringsgrensesnitt (API'er) for traversering av innholdet. Det er for eksempel lettere å transformere XML dokumenter eller dokumenter med tegnseparerte verdier enn dokumenter i regneark.

Neste avsnitt tar for seg hvordan tabulære data kan representeres som RDF. Videre i kapittelet gjennomgås det allerede eksisterende programmer for å transformere data lagret i forskjellige formater til RDF. I tillegg vises det applikasjoner og språk for å eksponere data lagret i relasjonsdatabaser til RDF. Det finnes i tillegg til de applikasjonene som blir gjennomgått flere applikasjoner for å transformere data til RDF, men disse blir ikke gjennomgått i denne oppgaven.

4.1 Tabulære data og RDF

Regneark, dokumenter med tegnseparerte verdier, og relasjonsdatabaser representerer alle formater hvor verdiene er uttrykket i form av tabeller. Slike tabeller uttrykker mening ved at alle radene beskrives med verdier hentet fra kolonner. Å uttrykke slike tabeller i form av RDF er ganske enkelt, ettersom RDF kan sees som en abstraksjon av tabulære data. Kombinasjonen

rad, kolonne og celle kan uttrykkes i form av RDF-tripler hvor subjektet representerer en rad innenfor en tabell. Objektene er angitt ut ifra innholdet i celler innenfor rader og kobles til subjektene gjennom predikater som er gitt av kolonnene i en tabell. Den generelle formelen for å transformere tabulær data til RDF blir da som følger:

1. Hver rad innenfor et regneark representerer et subjekt
2. Hver kolonne/kolonneoverskrift kan representere et predikat
3. Hver celle kan oversettes til et objekt

Figur-4.1 viser en tabell som inneholder filmer uttrykket ved kommaseparerte verdier, hvor den første raden inneholder kolonneoverskrifter. Hver rad innenfor dokumentet beskriver en film og kolonneoverskriftene uttrykker hvordan verdiene i cellene innenfor en rad beskriver en film. I tillegg ser vi at den første kolonnen inneholder en nøkkel som er unik innenfor tabellen for hver film. Dette gjør at man kan anvende verdien av denne kolonnen for å identifiserer hver film.

```
id_film,navn,produsert,sjanger
1,Jaws,1975,thriller
2,K–Pax,2006,drama
3,Grand Torino,2008,drama
```

Tabell 4.1: Kommaseparert representasjon av datasettet gitt i tabell-4.2

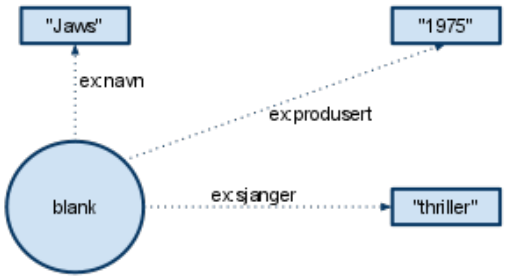
4.2 Representasjon av tabulære data

For å transformere dokumenter til RDF har man behov for å vite hvordan dataene er strukturert innenfor dokumentet. Kunnskap om dataene i dokumentet er viktig for å kunne skape RDF dokumenter som beholder den opprinnelige betydningen. Transformeringen av dokumenter kan resultere i forskjellige RDF-grafer ut ifra hvordan man benytter celleverdier som enten navngitte ressurser eller literaler. Det er ingen fasit på hvordan transformeringen av et dokument gjennomføres eller hvilke URI'er man skal bruke for å navngi dataelementer. En god regel i dette siste henseende er å så langt som mulig benytte seg av allerede eksisterende vokabularer. På denne måten unytter man den semantiske veven og ikke oppretter ressurser som allerede er laget.

id_film	navn	produsert	sjanger
1	Jaws	1975	thriller
2	K-PAX	2006	drama
3	Grand Torino	2008	drama

Tabell 4.2: Databaseskjema for filmer uttrykket tabulært

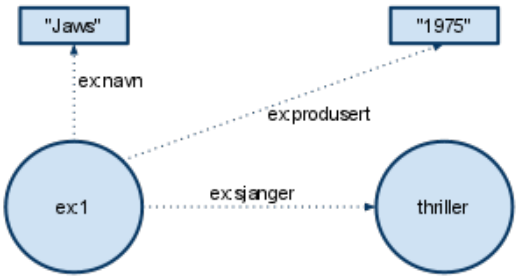
Tabulær data kan representeres som RDF i forskjellige former som utnytter den semantiske veven i forskjellige grad. Eksempelvis vil transformering av dataelementene gitt i tabell-4.2 kunne transformeres forskjellig. Hvis man under transformering av innholdet i denne tabellen knytter alle celleverdiene til et subjekt definert som en blank node som literaler vil man ende opp med en terminert stjerneformet graf. Slike RDF-grafer kan ikke benyttes av andre ettersom de ikke inneholder noen navngitte ressurser som kan refereres eksternt. Samtidig vil man ikke kunne utnytte ressurser som er beskrevet og lokaslisert i andre datasett ut ifra navngitte objekter. Å representere dataelementene i tabell-4.2 på denne metoden er i seg selv ikke feil, men man mister mye av strukturen og gjenbruksverdien man kunne utnyttet. Figur-4.1 viser hvordan resultaten av transformeringen av en rad i tabell-4.2 vil se ut.



Figur 4.1: Stjerneformet RDF representasjon av en film fra datasettet i tabell-4.2

Hvis man i stedet hadde transformert datasettet i tabell-4.2 og benyttet seg av allerede eksisterende vokabularer og navngitte ressurser i de tilfellene dette er mulig, ville man ha konstruert en RDF-graf som kan refereres av andre og som i tillegg refererer til andre eksterne ressurser ut ifra URI'er. Figur-4.2 viser hvordan en transformert rad fra datasettet kan representeres i form av en graf, når man benytter seg av både navngitte subjekter og navngitte objekter der dette er hensiktsmessig.

Innhold innenfor en rad i en tabell kan være relatert til raden i forskjellig grad.



Figur 4.2: RDF representasjon av en film fra datasettet i tabell-4.2 med et navngitt subjekt og objekt

4.2.1 Flate og multidimensjonale tabeller

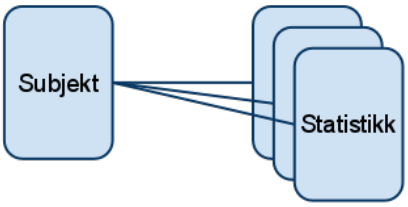
Den enkleste formen for tabeller er de tabellene hvor data i en rad hører sammen og kan knyttes direkte opp mot et subjekt. Slike tabeller refereres ofte til som flate og er de enkleste tabellene å lage transformeringsregler for. Dette fordi man kun trenger å lage en regel for hver kolonne man ønsker å ha med i det endelig transformerte RDF dokumentet.

I noen tabeller er det derimot ikke slik at alle verdier kan knyttes direkte mot et subjekt og samtidig gi mening. Det kan være tilfeller hvor data først må grupperes før disse grupperingene kan knyttes mot subjektet. Tabellen vist i figur-4.3 er et eksempel på dette. Der ser vi verdiene for en statistikkvariabel er gruppert etter år. Å knytte disse verdiene direkte mot subjektet vil ikke gi noen mening ettersom det ikke er noen indikasjon på når de gjelder for. Ved å gruppere verdiene og årstallene og knytte denne grupperingen mot subjektet vil derimot gi mer mening. Slike tabeller refereres ofte til som multidimensjonale.

region	Statistikkvariabel	2006	2007	2008
Akershus	Samlet inntekt under 150 000 kr	6	5	4
Akershus	Samlet inntekt 150 000 - 249 999 kr	11	10	9

Tabell 4.3: Multidimensjonal tabell – Utdrag fra tabell-5.4 om husholdninger etter størrelse på samlet inntekt fordelt på region og år

Multidimensjonale tabeller kan sees i sammenheng med relasjonsdata-basedesign hvor subjektet og all ekstra informasjon som hører direkte til subjektet ligger i en tabell, mens for eksempel de statistiske verdiene ligger i en annen tabell med subjekt identifikatoren som fremmednøkkel.



Figur 4.3: Multidimensjonale tabeller - Relasjonsdesing

4.3 Transformering av tegnseparerte dokumenter

Dokumenter med tegnseparerte verdier er en form for tabulære data. En linje i et slikt dokument representerer en rad hvor verdiene er delt i kolonner ut ifra tegnet som benyttes som skille. Dokumenter med innhold i dette formatet er programmeringsmessig de enkleste dokumentene å hente ut verdier ifra. De fleste programmeringsspråk inneholder allerede funksjonalitet for å lese filer og dele innholdet i hver linje ut ifra et gitt tegn.

Det finnes flere programmer for å transformere dokumenter med data lagret i dette formatet til RDF. Fra enkel transformering gitt av for eksempel ConvertToRDF[25] til mer avansert transformering gitt av RDF123[26]. I kapitlene 4.3.1 og 4.3.2 er en gjennomgang av disse transformasjonsprogrammene.

4.3.1 ConvertToRDF

ConvertToRDF[25] er et enkelt program laget for å transformere dokumenter som inneholder tegnseparerte verdier. Programmet implementerer den enkleste transformasjonsmodellen, nevnt i kapittel 4.3, hvor rader blir subjekter, kolonner blir predikater og celler blir objekter. I tilfellet med ConvertToRDF er objektene utypede literaler.

Programmet benytter seg av regler gitt i et templat for hvordan transformeringen av et dokument til RDF gjennomføres. I templatet kan man definere prefikser for hvilke vokabularer man ønsker å benytte seg av under transformeringen og hvordan subjektene konstruert under transformeringen skal types gjennom "rdf:type". Man kan enten angi at subjektene skal være blanke noder, eller noder som har "rdf:ID"[1] satt til innholdet i en gitt kolonne. Celleverdier knyttes mot de konstruerte subjektene gjennom definerte predikater, som kan utnytte prefiksene definert i templatet, ut ifra kolonner i form av utypede literaler. For å spesifisere hvilke kolonner celleverdier

[1]"rdf:ID" konkatineres med "xml:base" i XML for å lage en RDF-link

skal hentes ifra benyttes kolonneoverskrifter og ikke kolonnenummer. Dette medfører at alle dokumenter som skal transformeres med ConvertToRDF må ha en rad innenfor dokumentet som inneholder unike overskrifter.

Transformeringen i ConvertToRDF skjer per rad. Dette vil si at en rad uttrykker et subjekt og celleverdiene innenfor raden direkte beskriver subjektet i form av utypede literaler. Dette gjør programmet egnet til å uttrykke flate tabeller, men kommer til kort når det gjelder transformering av dokumenter med rikere innhold eller multidimensjonale tabeller. Alt som skal være med i det endelige RDF dokumentet som blir produsert av programmet må være en del av det tegnseparerte dokumentet. Dette vil si at man ikke kan definere metadata som beskriver dokumentet man har transformert, eller som beskriver subjektene konstruert under transformeringen. Man kan med andre ord ikke legge til informasjon.

Formatet benyttet for å definere templater til ConvertToRDF er enkelt, men følger ingen standarder og er konstruert spesielt til programmet. For å angi de forskjellige komponentene benyttes nøkkelord. "IPT" for å angi prefikser, "USE" for å angi typen til subjektene, "MAP" for å angi predikatene og hvilke kolonner de referer til, og "MAP ID" for å definere hvilken kolonne som skal benyttes til å sette "rdf:ID" til subjektene.

```
IPT http://www.example.com/ ex
USE ex:Film

MAP ex:id          "id_navn"
MAP ex:navn        "navn"
MAP ex:produsert   "produsert"
MAP ex:sjanger     "sjanger"
```

Figur 4.4: Eksempeltemplat for mapping ved ConvertToRDF

Figur-4.4 viser et eksempel på et templat for ConvertToRDF for å transformere datasettet i tabell-4.1. Her angir vi prefikset "ex" for `http://www.example.com/` og at "rdf:type" for alle subjektene skal settes til "ex:Film". Verdiene fra kolonnene med overskrift "id", "navn", "produsert" og "sjanger" skal kobles til subjektet gjennom predikatene "ex:id", "ex:navn", "ex:produsert" og "ex:sjanger".

4.3.2 RDF123

RDF123[26] er et kraftig transformasjonsprogram som benytter seg av et templat på samme måte som ConvertToRDF for å transformere dokumenter

med kommaseparerte verdier til RDF, men i motsetning til dette kan man uttrykke mer komplekse RDF-grafer ut ifra dokumentet man transformerer.

Transformeringen av dokumenter baserer seg på en templatgraf og ikke regler som ved ConvertToRDF. Dette vil si at man definerer hvordan man vil at RDF-grafene konstruert under transformeringen skal se ut. For å angi hvilke kolonner som skal knyttes til subjektet benytter man seg av variabler som refererer til de angitte kolonnene. Eksempelvis benyttes "$1" for å angi innholdet i den første kolonnen i dokumentet man transformerer. I de tilfellene variablene ikke inneholder noen verdi, vil triplet variabelen forekommer i ikke bli tatt med under transformeringen. I tillegg til å kunne benytte seg av verdier fra et dokument er det implementert et enkelt funksjonsbibliotek i RDF123 hvor man kan utføre forskjellige handlinger på eller ut ifra variablene. Eksempler på noe av den tilgjengelige funksjonaliteten i RDF123 er:

- Konkatenering av verdier og konstanter. For eksempel konkatenering av et prefiks definert i templatet og en verdi hentet fra en gitt kolonne. Dette fører til at man konstruerer navngitte subjekter og objekter.
- Utføre forskjellige handlinger ut ifra resultatet av likhetstester. Dette fører til at man for eksempel kan konstruerer forskjellige RDF-grafer basert på innhold i en gitt kolonne.
- Subtraksjon og addering av to verdier enten fra variabler eller konstante verdier gitt i templatet.

For å angi at man skal benytte seg av variabler eller funksjoner i templatet benytter man seg av nøkkelordet/navnerommet "Ex:" etterfulgt av variabelen man vil hente verdien ifra eller et funksjonsuttrykk. Hvis man ønsker for eksempel lage et navngitt objekt kan dette uttrykkes i templatgrafen som "<Ex:foaf+$2>". Programmet muliggjør også tilknytting av metadata til det ferdig transformerte dokumentet. Metadataene kan da defineres enten i dokumentet man transformerer ved å benytte seg av spesielle markeringer, eller de kan defineres i templatet.

Transformering av et dokument skjer per rad på samme måte som ConvertToRDF. For hver rad som skal transformeres vil RDF123 prøve å hente verdier og bytte ut uttrykkene angitt av "Ex:" så langt dette er mulig. Programmet kan i motsetning til ConvertToRDF benyttes til å transformere både flate og multidimensjonale tabeller, ettersom man kan konstruere navngitte eller blanke noder som har tilknyttede literaler eller objekter (se kapittel 4.2.1). Dette gjør at dokumenter transformert med RDF123 kan benyttes til lenking av data på den semantiske veven.

For å uttrykke templatene benytter RDF123 seg av Turtle serialisering. Ettersom Turtle er en serialiseringsform som er godt egnet for menneskelig lesning, og samtidig en de facto standard, egner den seg godt til å uttrykke templatgrafene benyttet av RDF123.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
  .
@prefix ex:  <http://www.example.com/> .

<Ex:ex+'filmer/'+$1> rdf:type ex:Film ;
   ex:navn "Ex:$2" ;
   ex:produsert "Ex:$3" .
   ex:sjanger "Ex:ex+'sjanger/'+$3" .
```

Figur 4.5: Eksempeltemplat for transformering ved RDF123

Eksemplet i figur-4.5 viser hvordan et templat for RDF123 kan uttrykkes for å transformere datasettet i tabell-4.1. Subjektet i grafen vil være en konkatenering av prefikset "ex", teststrengen "filmer/" og verdien som ligger i den første kolonnen, og vil være typet til "ex:Film". Til subjektet benyttes predikatet "ex:navn" for å angi navnet på subjektet som en typet literal ut ifra verdien i den andre kolonnen. Predikatet "ex:produsert" benyttes for å angi når subjektet var produsert som en utypet literal ut ifra verdien i den tredje kolonnen mens redikatet "ex:sjanger" knytter subjektet til en navngitt ressurs som representerer sjangeren til subjektet.

4.4 XLWrap for transformering av regneark

XLWrap[27, 28] er et program for å transformere regneark til RDF-grafer ut ifra et templat hvor dokumentet man transformerer kan være lokalisert lokalt eller eksternt igjennom veven. I motsetning til flere andre transformasjonsprogrammer er det i XLWrap innebygget støtte for flere formater som Microsoft Excel og OpenDocument regneark som kommaseparerte verdier.

Transformering av dokumenter med XLWrap består av to hovedkomponenter. Navngitte templatgrafer som uttrykker en eksempelgraf for hvordan RDF-grafene produsert under transformeringen av dokumenter skal se ut og prosesseringsinstruksjoner for hvordan templatgrafer flyttes rundt for å konstruere RDF-grafene.

Templatgrafer

Templatgrafene benyttet av XLWrap defineres som navngitte grafer i TriG syntaks[2]. Dette gjøres ved å opprette en navngitt blank node etterfulgt av krøllparanteser, hvor innholdet av krøllparantesene er en RDF-graf som angir hvordan man ønsker å benytte seg av verdier hentet fra dokumentet man transformere, for å konstruere én RDF-graf i det ferdig transformerte RDF dokumentet. XLWrap inneholder mye funksjonalitet for hvordan man kan benytte seg av verdier for å konstruere RDF-grafer. Under er noen eksempler på dette.

- Tekstkonkatenering av konstanter og eller verdier uttrykkes ved "&". Dette kan være nyttig når man skal benytte seg av celler for å konstruere navngitte ressurser.
- Man kan uttrykke aritmetiske funksjoner for utregning ved "+", "-", "*" og "/".
- Likhetstesting, og logiske konjunksjon og disjunksjon.

For å angi et uttrykk i XLWrap benyttes literaler typet til "xl:expr". Cellereferanser i et slikt uttrykk angis som en konkatenering av kolonnenavnet og raden cellen tilhører. For da å angi et uttrykk for å hente verdier fra celle "A2" benyttes literalet ""A2"^^xl:expr". Muligheten til å konkatenere tekstkonstanter og celleverdier gjør at man kan opprette navngitte objekter. Slike objekter opprettes ved å konstruere en blank node som inneholder predikatet "xl:uri". Uttrykket av dette literalet vil da bli benyttet som navngitt ressurs. Figur-4.6 viser et eksempel på en templatgraf hvor subjektet konstrueres som en navngitt ressurs ut ifra `http://www.example.com/filmer/` og verdien hentet fra celle "A2".

Prosesseringsinstrukser.

Prosesseringsinstrukser benyttes i XLWrap for å angi hvilket regneark en templatgraf skal benyttes for, og hvordan denne templatgrafen skal flyttes rundt i regnearket for å utføre transformeringen. Hvis man ikke angir hvordan en templatgraf skal flyttes vil den kun bli applisert en gang under transformeringen. Slike instrukser angis av predikatet "xl:template" og kan forkomme vilkårlig mange ganger i et templatdokument. Man kan i instruksene uttrykke:

[2]Serialiserings syntaks for RDF – `http://www4.wiwiss.fu-berlin.de/bizer/TriG/`

```
:Filmer {
  [ xl:uri "'http://www.example.com/filmer/' & A2"^^xl:Expr
    ] rdf:type ex:Film ;
    ex:navn "B2"^^xl:Expr ;
    ex:produsert "C2"^^xl:Expr ;
    ex:sjanger [ xl:uri "'http://www.example.com/sjanger/'
      & D2"^^xl:Expr .
}
```

Figur 4.6: Templatgraf i TriG syntaks for XLWrap

- Lokasjonen til dokumentet man skal transformere gjennom predikatet "xl:fileName".
- Hvilken side i dokumentet man skal transformere gjennom predikatet "xl:sheetName" eller "xl:sheetNumber".
- Grafer som skal knyttes direkte til det ferdig transformerte dokumentet som kan benyttes til å uttrykke metadata. Dette angis av predikatet "xl:constantGraph".
- Hvilken navngitt templatgraf som skal benyttes under transformeringen gjennom predikatet "xl:templateGraph".
- En sekvens av instruksjoner for hvordan templatgrafen eller deler av templatgrafen flyttes i dokumentet for å konstruere RDF-grafen. Denne sekvensen gis av predikatet "xl:transform".

Hvordan forskjellige bestanddeler av en templatgraf flyttes gis som en sekvens som inneholder forskjellige flytte spesifikasjoner. Når templatgrafer flyttes i et dokument vil cellereferansene angitt i templatgrafen referer til nye celler ut ifra spesifikasjonene.

- Hvordan templatgrafen, eller deler av templatgrafen flyttes til nye rader, angis av klassen "xl:RowShift".
- Hvordan deler av en templatgraf kan gjenbrukes over flere kolonner, angis av klassen "xl:ColShift".
- Hvordan templatgrafen eller deler av templatgrafen kan benyttes over flere sider innenfor et dokument angis av klassen "xl:SheetShift".
- Hvordan en templatgraf skal repeteres over flere dokumenter eller sider og angis av klassene "xl:FileRepeat" og "xl:SheetRepear".

Figur-4.7 viser et enkelt eksempel på hvordan man kan definere prosesseringsinstrukser for hvordan man transformerer et gitt dokument.

```
{
  [] rdf:type xl:Mapping ;
    xl:template [
      xl:fileName "file:filmer.xls" ;
      xl:templateGraph :Filmer ;
      xl:transform [ a xl:RowShift ]
    ] .
}
```

Figur 4.7: Prosesseringsinstrukes for et templat i XLWrap

Navngivning av templatgrafer gjør det mulig å benytte seg av forskjellige templater for å utføre forskjellige transformeringer som skal være med i det ferdig transformerte dokumentet. Dette kan for eksempel være to templatgrafer for å transformere et dokument, eller to templatgrafer for å transformere to forskjellige dokumenter, som gjør at man kan konstruere RDF datasett ut ifra forskjellige kilder. TriG syntaksen brukt av XLWrap ligner mye på Notation3 men er ikke en W3C standard og brukes kun for å benytte seg av navngitte grafer.

Transformeringen av dokumenter er da ikke radspesifikke som ved RDF123 og ConvertToRDF. XLWrap prøver å applisere templatgrafen ut ifra cellereferansene gitt i grafen. Når dette er gjort vil templatgrafen eller deler av templatgrafen bli flyttet rundt i dokumentet ut ifra sekvensene angitt av "xl:transform" hvor cellereferansene også flyttes. På denne måten kan XLWrap benyttes til å transformere både flate og multidimensjonale tabeller.

Figur-4.8 viser et eksempel på et templat benyttet av XLWrap for transformerer datasettet i tabell-4.2 til RDF.

4.5 Eksponering av data i relasjonsdatabaser som RDF

Data representert i relasjonsdatabaser utgjør en stor mengde data for den semantiske veven. RDF har den fordelen at man kan representere data modellert etter de samme prinsippene som i relasjonsdatabaser og egnes derfor til å eksponere slike data. Det har blitt utviklet en rekke verktøy og applikasjoner for å eksponerer data i relasjonsdatabaser i form av RDF som man kan navigere eller kjøre spørringer mot, gjennom for eksempel SPARQL

```
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns
  #> .
@prefix ex:     <http://www.example.com/> .
@prefix xl:     <http://purl.org/NET/xlwrap#> .

{ [] rdf:type xl:Mapping ;
    xl:template [
      xl:fileName "file:filmer.xls" ;
      xl:templateGraph :Filmer ;
      xl:transform [ a xl:RowShift ]
    ] .
}

:Filmer {
  [ xl:uri "'http://www.example.com/filmer/' & A2"^^xl:Expr
    ] rdf:type ex:Film ;
    ex:navn "B2"^^xl:Expr ;
    ex:produsert "C2"^^xl:Expr ;
    ex:sjanger [ xl:uri "'http://www.example.com/sjanger/'
      & D2"^^xl:Expr .
}
```

Figur 4.8: Eksempelmapping for transformering ved XLWrap

endepunkter[17]. Verktøy som dette muliggjør eksponering av relasjonelle data på den semantiske veven uten å endre strukturen på databasen som eksponeres ut ifra templater som transformerer tabeller og kolonner til klasser og predikater gitt av et vokabular.

Tim Berners-Lee publiserte i 1998 et notat[29] hvor han satt søkelys på hvordan data i relasjonsdatabaser utgjør en stor ressurs for gjenbruk og hvordan man kan eksponere dette igjennom for eksempel RDF. I tillegg til dette har W3C publisert R2RML[30] som er et transformasjonsspråk for hvordan man kan eksponere relasjonsdatabaser i form av RDF.

I neste avsnitt gjennomgår R2RML, mens D2RQ[31, 32] som er et verktøy for å eksponerer relasjonsdatabaser som RDF gjennomgås i kapittel 4.5.2.

4.5.1 Mapping av relasjonsdatabaser til RDF ved R2RML

RDB to RDF Mapping Language (R2RML)[30] er et pågående arbeid fra W3C og er et språk for å uttrykke mapping av relasjonelle databaser til

RDF datasett. Slike mappinger fungerer som et grenesnitt hvor relasjonelle data er eksponert i form av RDF ut ifra gitte vokabularer. R2RML uttrykker templatene for eksponering som RDF-grafer i Turtle serialisering og muliggjør mapping av forskjellige typer implementasjoner. Applikasjoner kan for eksempel benytte seg av R2RML for å eksponere relasjonelle data gjennom SPARQL endepunkter.

Hver R2RML mapping er laget for å eksponere et spesifikt databaseskjema ut ifra gitte vokabularer. Input til en mapping er en relasjonell database som følger et skjema definert i mappingen. Resultatet av denne mappingen er et RDF datasett som bruker predikater og typer fra de angitte vokabularene.

For å definere en mapping opprettes en blank node som er typet til "rr:TriplesMapClass" og inneholder predikatet "rr:SQLQuery" som angir en SQL spørring. Denne SQL spørringen benyttes som databaseskjema for mappingen. For å angi hvordan data kobles mot et subjekt benyttes predikatet "rr:predicateObjectMap" som angir en blank node. Denne blanke noden inneholder predikatet "rr:predicateMap" som definerer hvilket predikat som skal benyttes for å koble dataene mot subjektet, mens predikatet "rr:objektMap" angir hvordan objektet skal konstrueres enten som literaler eller navngitte ressurser ut ifra en gitt kolonne.

For å benytte seg av navngitte subjekter benyttes predikatet "rr:subjektMap" som viser til en blank node. Denne noden inneholder predikatet "rr:template" som angir en literal for hvordan subjektet skal konstrueres hvor verdiene innenfor krøllparanteser refererer til en kolonne. Et eksempel for hvordan dette literalet kan konstrueres er da `http://www.example.com/{id}`.

Figur-4.9 viser hvordan en mapping av databaseskjemaet vist i tabell-4.2 kan uttrykkes i et R2RML templat. Denne mappingen vil konstruere navngitte subjekter hvor verdiene fra kolonnene "navn", "sjanger" og "produsert" tilknyttes gjennom definerte predikater som typede literaler.

4.5.2 D2RQ

D2RQ[31, 32] benyttes for å eksponere relasjonsdatabaser ved å uttrykke regler for hvordan databaseskjemaer kan representeres som RDF. Disse reglene angir hvordan man bygger subjekter ut ifra verdier i en tabell og hvordan kolonner relateres til subjektene gjennom predikater. D2RQ er en plattform som består av tre deler:

- Et språk for hvordan man eksponerer relasjonsdatabaser som RDF ved å uttrykke relasjonen mellom et databaseskjema og en ontologi.

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix ex: <http://www.example.com/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<#TriplesMap1>
  a rr:TriplesMapClass;
  rr:SQLQuery """Select "id_film", "navn", "produsert", "
    sjanger" from Film""";

  rr:subjectMap [ rr:template "http://example.com/filmer/{
    id_film}" ];

  rr:predicateObjectMap [
    rr:predicateMap [ rr:predicate ex:id_film ];
    rr:objectMap    [ rr:column "id_film"  ] ];

  rr:predicateObjectMap [
    rr:predicateMap [ rr:predicate ex:navn ];
    rr:objectMap    [ rr:column "navn"; rr:datatype xsd:
      string ] ];

  rr:predicateObjectMap [
    rr:predicateMap [ rr:predicate ex:sjanger ];
    rr:objectMap    [ rr:column "sjanger"; rr:datatype xsd:
      string ] ];

  rr:predicateObjectMap [
    rr:predicateMap [ rr:predicate ex:produsert ];
    rr:objectMap    [ rr:column "produsert"; rr:datatype
      xsd:string ] ] .
```

Figur 4.9: Eksempelmapping av et databaseskjema uttrykket i R2RML

- En motor som implementerer en D2RQ graf i form av en Jena[3] graf, som er den enkleste representasjonsformen i Jena. D2RQ grafen pakker inn relasjonsdatabasen til en virtuell RDF-graf basert på eksponeringstemplatet og omdefinerer funksjonskall i Jena og Sesame[4]

[3]Java rammeverkt for å utvikle applikasjoner for den semantiske veven – `http://jena.sourceforge.net/`

[4]Rammeverk for å prosessere RDF data – `http://www4.wiwiss.fu-berlin.de/bizer/TriG/`

for å omskrive SPARQL spørringer til SQL spørringer som blir kjørt mot databasen. Resultatet av slike spørringer returneres enten i from av RDF tripler eller som et SPARQL resultatsett.

- En vevserver for å eksponere relasjonsdatabaser gjennom et SPARQL endepunkt.

For å angi hvordan data i relasjonsdatabaser eksponeres i D2RQ konstueres et templat som angir forskjellige regler. Eksempler på hva man kan uttrykke ved regler:

- Hvordan man kobler til databasen uttrykkes ved å konstruere et subjekt som er typet til "d2rq:Database". For dette subjektet kan man definere hvilken database man skal koble til og hvilken driver man skal benytte for tilkoblingen. I tillegg kan man uttrykke brukernavn og passord for å koble seg til databasen.

- En regel for hvordan subjektene for en tabell blir navngitt og typet. Dette gjøres ved å opprette et subjekt av typen "d2rq:ClassMap". Her kan man angi hvordan subjektet skal konstrueres, hvordan det skal types og hvilken "d2rq:Database" man skal benytte seg av.

- For å uttrykke hvordan relasjonen et subjekt og kolonnedataene i en tabell kan man konstruerer subjekter som er typet til "d2rq:PropertyBridge". Disse angir predikatet som skal benyttes for å koble dataene sammen, hvordan dataene skal representeres som enten literaler eller navngitte ressurser. I tillegg angis det hvilken "d2rq:ClassMap" de hører til.

For å referere til kolonner i en tabell benytter man seg av "@@" etterfulgt av navnet på tabellen man skal hente verdiene ifra konkatenert med et punktum og navnet på kolonnen i databaseskjemaet. For å avslutte kolonnereferansen benyttes "@@".

Figur-4.10 viser hvordan et templat for å eksponere data fra en tabell i en relasjonsdatabase kan se ut. Tabell 4.2 benyttes som "Film" databaseskjema for eksemplet. Subjektet skal bestå av et prefiks konkatenert med verdien av kolonnen "id_film" i "Film" databasen og skal types til "ex:Film". Navnet på filmen skal knyttes til subjektet som en typet literal hentet fra "navn" kolonnen og året filmen var produsert knyttes til subjektet igjennom predikatet "ex:produsert" som en typet literal hentet fra "produsert" kolonnen. Verdiene i "sjanger" kolonnen knyttes til subjektet gjennom predikatet "ex:sjanger" som et navngitt ressurs.

```
@prefix d2rq: <http://www.wiwiss.fu-berlin.de/suhl/bizer/
  D2RQ/0.1#> .
@prefix ex: <http://www.example.com/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

# Namespace of the mapping file; does not appear in mapped
  data
@prefix map: <file:///None/void/example.n3#> .

map:Database1 a d2rq:Database;
     d2rq:jdbcDSN "jdbc:mysql://localhost/database";
     d2rq:jdbcDriver "com.mysql.jdbc.Driver";
     d2rq:username "brukernavn";
     d2rq:password "passord".
     .
map:Film rdf:type d2rq:ClassMap;
     d2rq:dataStorage map:Database1.
     d2rq:class ex:Film;
     d2rq:uriPattern "http://www.example.com/filmer/@@Filmer
       .id_film@@" .
     .
map:navn rdf:type d2rq:PropertyBridge;
     d2rq:belongsToClassMap map:Film;
     d2rq:property ex:navn;
     d2rq:column "Filmer.navn";
     d2rq:datatype xsd:string .
     .
map:produsert rdf:type d2rq:PropertyBridge;
     d2rq:belongsToClassMap map:Film;
     d2rq:property ex:produsert;
     d2rq:column "Film.produsert";
     d2rq:datatype xsd:string .

map:sjanger rdf:type d2rq:PropertyBridge;
     d2rq:belongsToClassMap map:Film;
     d2rq:property ex:sjanger;
     d2rq:uriPattern "http://www.example.com/sjanger/@@Film.
       sjanger@@";
```

Figur 4.10: Eksempelmapping for å eksponere en tabell som inneholder filmer i D2RQ

Kapittel 5

Transformering av Microsoft Excel regneark

Mye data på veven blir publisert i formater som ikke egner seg til gjenbruk. Dette kan for eksempel være data lagret tekstlig i form av tegnseparerte vedier, eller tabulære data, i form av lukkede regnearkformater. En del av arbeidet med denne masteroppgaven har gått ut på å lage en applikasjon for å transformere Microsoft Excel regneark til gjenbrukbare RDF dokumenter ut ifra regler definert i et templat for hvordan transformeringen skal utføres. Resultatet av dette er applikasjonen RDFizer.

RDFizer er en applikasjon skrevet i Java og består av to deler, en prosesseringsmotor og en templatmotor. Templatmotoren henter ut regler definert i et templat som skal benyttes under transformeringen av regneark, og bygger opp en intern regelstruktur utifra dette. Prosesseringsmotoren benytter seg av denne templatmotoren for å anvende regler mot data hentet fra regnearket man transformerer. Ved å utvikle programmet på denne måten kan man enkelt benytte seg av templatmotoren og regeldefinisjonene utviklet for transformering til å utvikle andre applikasjoner for å transfomere dokumenter i andre formater en Microsoft Excel. Applikasjonen benytter seg av kjente åpne eksterne API'er for å kunne håndtere forskjellige aspekter ved transformeringen av regneark til RDF.

5.1 Java API'er benyttet av RDFizer

Jena benyttes i templatmotoren for å hente ut informasjon fra templatet som blir benyttet under transformeringen. Jena muligjøring å kjøre SPARQL spørringer mot et gitt dokument og benyttes i stor grad for å hente ut regeler som skal anvendes i forskjellige deler av transformasjonsprosessen. I tillegg

til dette benyttes Jena til å bygge opp det resulterende RDF dokumentet RDFizer konstruerer etter transformering av regneark er ferdig.

For behandling av Microsoft Excel regneark benyttes Apache POI[1]. Denne API'en benyttes for å kunne traversere rader innenfor et regneark og hente ut verdier fra cellene i hver rad ut ifra kolonnenummere.

5.2 RDFizer

RDFizer utnytter templatspråket og templatmotoren for å kunne transformere regneark til RDF. Templatspråket angir hvordan transformasjonen skal gjennomføres ved at man kan definere regler og instrukser. Disse regeldefinisjonen muliggjør styring av hvordan man oppretter subjekter og hvordan verdier fra regneark knyttes opp mot disse subjektene som enten literaler eller objekter. For å gi enkel kontroll over selve transformeringen av et regneark er det mulig å definere instruksjoner som sier noe om hvor man vil starte og hvor man vil avslutte transformeringen. I tillegg muligjør templatspråket at man knytter metadata direkte til subjektene som blir opprettet eller metadata som knyttes til det ferdig transformerte RDF dokumentet. I templatspråket benyttet av templatmotoren må man da kunne definere:

- hvordan subjektene skal bygges. Enten som navngitte objekter ut ifra en URI prefiks og en verdi hentet fra regnearket man transformerer, eller som blanke noder.
- metadata som skal knyttes mot subjektene som blir opprettet under transformeringen av et regneark.
- metadata som skal knyttes direkte til det ferdig transformerte RDF dokumentet.
- instruksjoner for å styre hvor innenfor et regneark man ønsker å starte og avslutte transformeringen.
- hvordan man benytter seg av celleverdier i et regneark for å konstruere typede literaler.
- hvordan man konstruerer navngitte objekter utfira celleverdier i et regneark sammen med URI prefikser, eller hvordan man oppretter blank noder.
- hvordan regler kan gjenbrukes over flere kolonner.

[1] Java API for Microsoft Dokumenter – `http://poi.apache.org/`

- om man skal benytte seg av verdier hentet fra regneark man transformerer eller bestemte verdier.
- hvilke predikater som skal benyttes for å knytte typede literaler, navngitte objektene, eller blanke nodene til subjektet.
- hvordan man kan knytte navngitte objekter eller typede literaler til objekter som igjen knyttes til subjektet gjennom gitte predikater.

Føringene listet over for hvordan man oppretter subjekter og hvordan verdier knyttes til disse subjektene gjennom predikater, kan sees i sammenheng med hvordan tabulære data kan representeres som RDF (se kapittel 4.1). Dette gjelder såfremt man kun transformerer verdier hentet fra kolonner i regnearket og hvor disse verdiene knyttes direkte til subjektet som enten typede literaler eller navngitte objekter.

- Subjektene blir opprettet ut ifra rader i regnearket man transformerer. Disse subjektene ender da opp som enten navngitte objekter eller blanke noder.
- Predikatene for hvordan verdier fra celler blir knyttet til subjektet er knyttet til kolonnen hvor verdiene er gitt.
- Objektene blir opprettet ut ifra verdier som er hentet fra celler. Disse blir da knyttet til subjektet som enten typede literaler eller navngitte objekter.

I de tilfellene hvor verdier ikke blir hentet fra regnearket vil denne koblingen falle bort. Predikatet vil ikke lengre være knyttet til noen spesifikk kolonne og verdiene benyttet for å konstruerer de navngitte objekter eller typede literaler vil ikke være hentet fra celler i regnearket.

5.3 Templatspråket

I kapittel 5.2 ble det listet opp føringer og instrukser som kan uttrykkes i templatspråket for hvordan transformeringen av regneark gjennomføres av RDFizer. Disse føringene og instruksene kan deles inn i fire hovedbestanddeler:

- *Subjektspesifikasjon*: om hvordan subjekter bygges.
- *Metadata:* om hvordan man definerer metadata som skal knyttes mot subjekter eller til det ferdig transformerte RDF dokumentet.

- *Prosesseringsinstrukser:* om hvordan man kan styre enkle aspekter av transformasjonen.
- *Ressursspesifikasjoner:* om hvordan man definerer regler for hvordan verdier fra regnearket eller bestemte verdier knyttes til subjekter som enten typede literaler eller objekter.

Videre i dette delkapittelet er det en gjennomgang av hovedbestanddelene av templatspråket og hvordan man kan benytte seg av disse for å definere regler og instrukes for hvordan man transformerer regneark I tillegg gis det eksempler for hvordan dette kan uttrykkes i templater. Kapittel 5.3.1 viser grammatikken i templatspåket definert i form av Utvidet Backus-Naur form ved hjelp av Turtle serialisering. Denne serialiseringen formen blir også benyttet for å beskrive de forskellige bestandelene av templatspråket. Vokabularet `http://www.example.com/` benyttet i eksemplene for hvordan de forskjellige bestandelene kan uttrykkes i templater og er kun brukt som eksempel.

5.3.1 Templatspråkets grammatikk

Tabell 5.1: Templatspråkets grammatikk beskrevet i Utvidet Backus-Naur form

template	"@prefix rdf: <" , rdf , "> ." , "@prefix rdfs: <" , rdfs , "> ." , "@prefix smc: <" , smc , "> ." , "@prefix xsd: <" , xsd , "> ." , "@prefix owl: <" , owl , "> ." , { prefix } , [sheet] , [selection] , subject , { datapred } , { objectpred } ,
prefix	"@prefix " , stringnow , ": <", url , "> ."
sheet	"[]" , rdf,"type" , smc,"Sheet" , term { url , string , term | url , url , term }
selection	{ offset | eof }
offset	smc,"selection" , " " , smc,"offset" , " " , typednum , term
eof	smc,"selection" , " " , smc,"eof" , " " , allnum , term

Fortsettelse på neste side

	Fortsettelse fra forrige side
subject	"[]" , " " , rdf,"type" , " " , smc,"subject" , term , [smc,"identifiedBy" , " [" , idspec , "]"] , term , [smc,"describedBy" , " [" , descspec , "]"] , term
idspec	rdf,"type" , " " , smc,"subjectId" , term , smc,"prefix" , " " , url , term , smc,"columnNumber" , " " , typednum
descspec	smc,"subjectType", url , term , [{ url , " " , { string | url } }]
datapred	url , " " , rdf,"type" , " " , owl,"DatatypeProperty" , term , rdfs,"range", " " , xsd , xsdtype , term , (valuepred | indexpred) , term
objectpred	url , " " , rdf , 'type' , " " owl,"ObjectProperty" , term , smc,"ObjectSpec", " " , objspec , [shift]
objspec	"[" , blankspec , "]" | "[", resspec , "]"
blankspec	rdf,"type" , " " , smc,"blankNode" , term { url , " " , espec } , term
resspec	rdf,"type" , " " , url , term , smc,"prefix" , " " , url , term , valuepred , term | indexpred , term , { url , " " , espec }
espec	"[" , rdf,"type" , " " , smc,"expression" , term , rdfs,"range", ' ' , xsd , xsdtype , term , indexpred "]" term | valuepred "]" term | "[" , rdf,"type" , " " , smc,"expression" , term , smc,"prefix", " " , url , term , indexpred "]" term | valuepred "]" term ,
shiftspec	smc,"shift" , " " , "[" , rdf,"type" , " " , smc,"expression" , " " , term , smc,"length" , " " , typednum , term , smc,"duration" , " " , typednum , "]", term
valuepred	smc,"value" , " " , string
indexpred	smc,"index" , " " , expression
expression	"C" , number | "C" , number , "-R" , number
typednum	“"”, number , “"” , xsd,"int"
allnum	“"”, ? all numbers ? , “"” , xsd,"int"
	Fortsettelse på neste side

	Fortsettelse fra forrige side
datatype	xsd , xsdtype
url	{ ws } , ?all valid url addresses?
term	{ ws } , { "." | ";" }
ws	?white space characters? | ""
number	?all positive numbers including 0?
string	“"” , ?all visible characters including white spaces? , “"”
stringnow	“"” , ?all visible characters? , “"”
xsdtype	?-all posible xsd datatypes?
rdf	{ ws } , "'http://www.w3.org/1999/02/22-rdf-syntax-ns#"
rdfs	{ ws } , "http://www.w3.org/2000/01/rdf-schema#"
owl	{ ws } , "http://www.w3.org/2002/07/owl#"
smc	{ ws } , "http://www.semicolon.no/excellence#"
xsd	{ ws } , "http://www.w3.org/2001/XMLSchema#"

5.3.2 Subjektspesifikasjon

I henhold til grammatikken må alle templater ha med en subjektspesifikasjon. Denne spesifikasjonen sier hvordan subjektene man knytter all annen data til skal se ut utifra hver rad i et regneark. Man kan enten benytte seg av verdier hentet fra en spesifikk kolonne sammen med en URI prefiks for å lage navngitte subjekter, eller man kan benytte seg av blanke noder. Muligheten til å definere subjekter som blanke noder kan være nyttig i de tilfellene hvor regneark man ønsker å transformere ikke inneholder en kolonne som kan benyttes som nøkkelkolonne.

Selve subjektdefinisjonen utføres ved at man lager en blank node av typen “smc:subject”. Hvis denne noden ikke har noen andre predikater knyttet til seg vil man benytte seg av blanke noder som subjekter under transformeringen av regnearket. Figur-5.1 viser hvordan subjektspesifikasjonen vil kunne se ut i et slikt tilfelle.

```
[] rdf:type smc:subjekt .
```

Figur 5.1: Subjektspesifikasjon - Blanke noder

Hvis man istedet for å benytte blanke noder som subjekter ønsker å benytte seg av verdier i en spesifikk kolonne for å lage navngitte subjekter, må man knytte en identifikasjonsspresifikasjon til definisjonen av subjektet igjennom predikatet “smc:identifiedBy” som en blank node. Denne spesifikasjonen må være av typen “smc:subjectId” og må inneholder predikatene “smc:prefix” og “smc:columnNumber”. “smc:columnNumber”

	Kundenummer	Fornavn	Etternavn	Telefonnummer	Telefonnummer
2	101	Ola	Norman	999191919	999191918
3	102	Per	Norman	999181818	999181817
4	103	Kari	Norman	999171717	999171716
5	104	Norman	Norman	999161616	999161615
6	105	Olga	Norman	999151515	999151514

Tabell 5.2: Eksempeldatasett hvor subjekt kolonnen er markert

definere hvilken kolonne man skal bruke for å konstruere subjektene, og "smc:prefix" definere URI prefisken som skal knyttes sammen med verdiene som ligger i subjektkolonnen for å lage de navngitte subjektene.

```
[] rdf:type smc:subject ;
  smc:identifiedBy [
    rdf:type smc:subjetId ;
    smc:columnNumber "0"^^xsd:int ;
    smc:prefix "http://www.exsample.com/kunde#" ] .
```

Figur 5.2: Subjektspesifikasjon - Navngitte subjekter

I subjektspesifikasjonen vist i figur-5.2 defineres subjekter utifra verdiene som ligger i den første kolonnen i datasettet i figur-5.2 sammen med prefikset `http://www.example.com/kunde#`.

5.3.3 Metadata

I mange tilfeller vil det være nyttig å kunne knytte ekstra informasjon som beskriver innholdet i et ferdig transformert RDF dokument. Dette kan for eksempel være hvem som har publisert dataene og hva dataene i dokumentet dekker. Fordelene ved å knytte metadata til dokumenter er at andre kan oppdage og eventuelt integrere eller lenke sammen egen datasett utifra denne informasjonen. I templatspråket gjøres dette ved å opprette en blank node som er av typen "smc:Sheet". All metadata som man ønsker å knytte til RDF dokumentet er da predikater av denne blanke noden. "smc:Sheet", slik som den er bygget opp i templatet, vil da direkte bli tatt med i det ferdig transformete RDF dokumentet. Figur-5.3 viser hvordan man kan benytte seg av "smc:Sheet" for å angi metadata som skal innkluderes i det ferdit transformerte RDF dokumentet. Her vil metadata om kilden, hvem som har publisert dataene og hva innholdet i RDF dokumentet dekker bli inkuldert.

Det vil ofte være nyttig å legge ved ekstra beskrivelser av subjektene man oppretter under transformeringen av et regneark. For å uttrykke dette i templatspråket knytter man en beskrivelsesspesifikasjon til subjektspesifikasjonen

```
[] rdf:type smc:Sheet ;
  dc:source "http://www.example.com/" ;
  dc:publisher "Eksempelregneark";
  dc:coverage "Kunder" .
```

Figur 5.3: Metadata for tilknyttning av ferdig transformert RDF dokument

gjennom predikatet "smc:describedBy" som en blank node. Denne beskrivelsesspesifikasjonen må ha med predikatet "smc:subjectType som brukes til å sette "rdf:type" for subjektene. Alt som i tillegg til dette er predikat av beskrivelsesspesifikasjonen vil bli knyttet til alle subjektene som blir konstruert under transformeringen av et regneark. Hvordan man uttrykker metadata som skal knyttes til subjektene er vist i figur-5.4.

```
[] rdf:type smc:subject ;
  smc:identifiedBy [
    rdf:type smc:subjetId ;
    smc:columnNumber "0"^^xsd:int ;
    smc:prefix "http://www.exsample.com/kunde#" ] ;
  smc:describedBy [
    smc:subjectType ex:Kunde ;
    ex:region "Akershus"^^xsd:string ] .
```

Figur 5.4: Subjektspesifikasjon med tilknyttet metadata

Hvis vi benytter oss av denne subjektspesifikasjonen mot datasettet vist i tabell-5.2 vil et eksempel for hvordan et subjekt kan se ut være som vist i figur-5.5.

```
<http://www.example.com/kunde#101> rdf:type ex:Kunde ;
  ex:region "Akershus"^^xsd:string .
```

Figur 5.5: Eksempelgraf for subjektspesifikasjon

5.3.4 Prosesseringsinstruksjoner

Regneark inneholder ofte en del informasjon som ikke direkte er en del av datasettet. Dette kan være metadata som beskriver datasettet, som publikasjonsdato eller hvem som har publisert datasettet. Ofte er det også

slik at man benytter seg av kolonnetitler for å beskrive innholdet i forskjellige rader, som i seg selv ikke er data. I andre tilfeller kan det hende man ikke ønsker å transformere hele regneark, men i stedet et utdrag av innholdet. For å gi mulighet til å begrense hvor man vil starte og hvor man vil avslutte transformeringen, er det mulig i templatspråket å uttrykke dette gjennom subjektet "smc:selection". "smc:selection" benytter seg av predikatet "smc:offset" for å definere hvor mange rader man ikke ønsker å ta med fra starten av et regneark under transformeringen. Predikatet "smc:eof" definerer hvilken rad man ønsker at transformeringen skal avsluttes på. I figur-5.6 viser hvordan dette kan uttrykkes i et templat hvor transformeringen skal starte på rad 2 og avsluttes på 10.

```
smc:selection smc:offset "1"^^xsd:int ;
  smc:eof "10"^^xsd:int ;
```

Figur 5.6: Prosesseringsinstrukser for å starte transformeringen på rad 2 og avslutte på rad 10

I tillegg til å kunne definere hvilken rad transfomasjonen skal avsluttes på kan man gi predikatet "smc:eof" en negativ verdi. På denne måten vil man avlsutte transformeringen når det et gitt antall rader igjen av regnearket. Hvis "smc:eof" ikke er satt eller er satt til 0 vil alle radene tilgjengelige i et regneark bli transformert.

```
smc:selection smc:eof "-10"^^xsd:int ;
```

Figur 5.7: Prosesseringsinstruks for å ikke ta med de siste radene i et regneark man transformerer

Figur-5.7 viser hvordan man kan uttrykke at man ikke ønsker å transformere de 10 siste radene i et regenark.

5.3.5 Ressursspesifikasjoner

For å knytte verdier fra celler i et regneark mot subjekter defineres regler for hvordan disse verdiene transformeres og hvordan de skal knyttes mot subjektene. Templatspråket muliggjør tilknyttning av typede literaler og objekter til subjekter. Dette gjøres ved å opprette et subjekt som benyttes som predikat for relasjonen mellom subjektet og ressursen regelen lager. For å angi om et predikat knytter et subjekt mot en literal eller et objekt defineres av hvordan predikatet er typet. Hvis predikatet er typet

til "owl:ObjectPrperty" vil denne bli benyttet for å opprette en blank eller navngitt node og knytte dette mot subjektet gjennom predikatet. Hvis predikatet er typet til "owl:DatatypeProperty" vil denne bli benyttet for å opprette en typet literal som tilknyttes subjektet gjennom predikatet. De neste seksjonene viser hvordan man definerer regler for å opprette objekter og literaler samt hvordan man kan definere hvilke verdier som skal benyttes for å opprette ressursene.

Variabler og konstanter

For å definere hvilke verdier man ønsker å ta med seg under transformeringen av regneark kan man benytte seg av "smc:index" eller "smc:value". "smc:value" brukes i de tilfellene man vil angi at et predikat skal knytte en konstant verdi som er definert av "smc:value" til alle subjektene som blir laget under transformeringen av regneark. "smc:index" brukes for å hente ut verdier fra spesifikke kolonner i et regneark. For å uttrykke hvilken kolonne man skal hente verdier fra benytter man seg av en tekststreng som begynner med nøkkelen "C" etterfulgt av kolonnenummeret man ønsker å benytte. "smc:index" kan i tillegg referer til bestemte celler innenfor et regneark hvis man benytter seg av en utvidet syntaks, se figur-5.10, hvor man etter å ha definert kolonnen legger ved "-" og nøkkelen "R" etterfulgt av nummeret på raden man hente den bestemte cellen fra. Det er viktig å legge merke til at i de tilfellene man benytter seg av bestemte kolonner må man ikke definere raden relativt til verdien av "smc:offset", men refere til den faktiske raden innenfor et regneark. Figur-5.8 viser hvordan man kan benytte "smc:index" for å hente ut verdier fra en spesifikk kolonner, mens figur-5.9 viser hvordan man med "smc:value" benytter seg av en konstant verdi. Figur-5.10 viser hvordan man kan benytte seg "smc:index" for å refere til en bestemt kolonne i en bestemt rad.

```
ex:year rdf:type owl:DatatypePropery ;
  rdf:range xsd:int ;
  smc:index "C1" .
```

Figur 5.8: "smc:index" for å hente verdier fra kolonner

Om man lager et objekt eller en typed literal ut ifra "smc:value" eller "smc:index" kommer an på hvilke andre predikater som er definert for objektet de er predikater for. Se seksjonene om literalspesifikasjon og objektspesifikasjon.

```
ex:year rdf:type owl:DatatypePropery ;
  rdf:range xsd:string ;
  smc:value "2010" .
```

Figur 5.9: "smc:value" for å benytte en kostant som verdi

```
ex:year rdf:type owl:DatatypePropery ;
  rdf:range xsd:int ;
  smc:index "C1–R1" .
```

Figur 5.10: "smc:index" for å referer til en bestemt kolonne

Literalspesifikasjon

Den enkleste transformasjonsreglen man kan uttrykke i templatspråket er for literaler. Dette er verdier som er hentet direkte fra et regneark, eller definert som en konstant, og skal knyttes mot subjektet. Dette defineres ved å opprette et subjekt som er typet til "smc:DatatypePropery". Verdien som skal brukes i literalet hentes ut fra predikatet "smc:index" eller "smc:value" og blir typet til datatypen definert av predikatet "rdfs:range". For å definere at verdiene plassert i kolonne 1 skal knyttes til et subjekt gjennom predikatet "ex:firstname" og ha datatypen "xsd:string" kan vi benytte reglen vist i figur-5.11.

```
ex:firstname rdf:type owl:DatatypePropery ;
  rdfs:range xsd:string ;
  smc:index "C1" .
```

Figur 5.11: Literalspesifikasjon

Når denne reglen blir benyttet mot datasettet vist i figur-5.2 sammen med subjektspesifikasjonen i figur-5.4 vil grafen kunne se ut som vist i figur-5.12 .

```
<http://www.example.com/kunde#101> rdf:type ex:Kunde ;
  ex:region "Akershus"^^xsd:string ;
  ex:firstname "Ola"^^xsd:string .
```

Figur 5.12: Eksempelgraf etter literalspesifikasjon

Literalspesifikasjoner som benytter seg av konstante verdier hentet fra "smc:value" kan sees på som metadata for et subjekt, beskrevet i kapittel

5.3.3, ettersom verdiene koblet mot subjektet ikke forandres basert på rader innenfor et datasett.

Objektspesifikasjon

For å lage beskrivende og og gode RDF dokumenter vil det ofte være behov for å kunne knytte blanke eller navngitte objekter til subjektene ut ifra verdier hentet fra regnearket. Navngitte objekter er byggestenen i lenkede data hvor man igjennom URI'ene benyttet som navn kan relatere ressurser som er lokalisert og eventuelt beskrevet andre steder.

For å lage objektregler definerer man et subjekt som er typet til "owl:ObjectPropery". Denne regelen benytter seg da av predikatet "smc:ObjectSpec" som viser til en blank node. Hvis denne noden er typet til "smc:BlankNode" vil det opprettes en blank node, ellers vil det ut ifra predikatene "smc:prefix" og "smc:value" eller "smc:index" opprettes en navngitt ressurs som knyttes mot subjekt. Figur-5.13 viser hvordan man oppretter navngitte ressurser.

```
ex:phonenumber rdf:type owl:ObjectPropery ;
  smc:ObjectSpec [
    rdf:type ex:PhoneNumber ;
    smc:prefix "http://www.example.com/phone/mobile/no/" ;
    smc:index "C3" ] .
```

Figur 5.13: Objektspesifikasjon

I tillegg til å representere enkle objekter tillater templatspråket at verdier fra celler i regenarket kan knyttes mot et definert objekt og ikke direkte til subjektet som enten er navngitte objekter eller typede literealer. Dette uttrykkes ved at man i den blanke noden "smc:ObjectSpec" oppretter predikater mot blanke noder som er typet til "smc:expression", hvor predikatene blir benyttet for å koble objektet mot navngitte ressurser eller typede literaler. Hvis en slik blanke node inneholder predikatet "smc:prefix" vil man opprette en navngitt ressurs ut ifra predikatet "smc:index" eller "smc:value". Dersom noden inneholder predikatet "rdfs:range" vil man lage et typet literal ut ifra "smc:index" eller "smc:value". Transformasjonen av data fra regneark ved denne metoden er nødvendig for å kunne representere multidimensjonale tabeller (se kapittel 4.2.1. Figur-5.14 viser hvordan man kan lage en blank node som knyttes mot subjektet gjennom predikatet "smc:phonenumber", hvor den blanke noden har to objekter knyttet til seg gjennom predikatet "ex:mobile".

```
ex:phonenumber rdf:type owl:ObjectPropery ;
  smc:ObjectSpec [
    rdf:type smc:BlankNode ;
    ex:mobile [
      rdf:type smc:expression ;
      smc:prefix "http://www.example.com/phone/mobile/no/"
        ;
      smc:index "C3" ] ;
    ex:mobile [
      rdf:type smc:expression ;
      smc:prefix "http://www.example.com/phone/mobile/no/"
        ;
      smc:index "C4" ] ;
```

Figur 5.14: Avansert objecktspesifikasjon

Hvis vi benytter oss av denne objektspesifikasjonen mot datasettet i figur-5.2 sammen med spesifikasjonene vist i figur-5.4 og figur-5.11 vil grafen kunne se ut som vist i figur-5.15.

```
<http://www.example.com/kunde#101> rdf:type ex:Kunde ;
  ex:region "Akershus"^^xsd:string ;
  ex:firstname "Ola"^^xsd:string .
  ex:phonenumber [
    ex:mobile <http://www.example.com/phone/mobile/no
      /999191919> ;
    ex:mobile <http://www.example.com/phone/mobile/no
      /999191918> ]
```

Figur 5.15: Eksempelgraf etter objektspesifikasjon

Gjenbruk av transformasjonsregler

Templatspråket tillater at objektspesifikasjoner definert for et gitt antall kolonner kan bli repetert langs den horisontale aksen. Objektspesifikasjonen muliggjør å konstruere grafer som uttrykker multidimensjonale tabeller. Ved å gjenta en objektspesifikasjon over flere rader tillater i tillegg å uttrykke "en til mange" relasjoner hvor et subjekt kobles til et flere objekter gjennom samme predikat. Se figur-4.3.

Dette uttrykkes i et templat ved å koble en blank node til en objektspesifikasjon igjennom predikatet "smc:shift". Denne blanke noden må være typet til

"smc:expression" og benytter seg av verdiene for predikatene "smc:duration" og "smc:length" for repetere en objektspesifikasjon. "smc:length" sier hvor mange kolonner objektspesifikasjonen skal flyttes for hver gang den blir gjenbrukt, mens "smc:duration" sier hvor mange gange objektspesifikasjonen skal gjenbrukes for hver rad. For en objektspesifikasjon som har "smc:shift" verdien "smc:length" lik 2 og "smc:duration" lik 3 vil bli gjennomført totalt 4 ganger. En gang for den originale transformasjonen, samt en gang for hver gang regelen blir applisert på nytt. Gjenbruk av objektspesifikasjoner er nyttig i de tilfeller hvor data er listet opp langs den horisontale aksen og samme spesifikasjon skal benyttes over flere kolonner. Dette kan for eksempel være statistisk data fordelt på år, eller stemmefordeling fordelt på politiske partier.

```
ex:mobile rdf:type owl:ObjectProperty ;
  smc:ObjectSpec [
    rdf:type ex : PhoneNumber ;
    smc:prefix "http://example.com/phone/mobile/no/" ;
    smc:index "C3" ] ;
  smc:shift [
    rdf:type smc:expression ;
    smc:length "1"^^xsd:int ;
    smc:duration "1"^^xsd:int ] .
```

Figur 5.16: Gjenbruk av en objektspesifikasjon

Eksemplet i figur-5.14 benyttes for å lage en blank node hvor mobilnummere fra to forskjellige kolonner tilknyttes før den blanke noden knyttes til subjektet. Ønsker man istedet å knytte mobilnummerene direkte til subjektet igjennom predikatet "ex:mobile" kan man gjøre dette ved å definere en objektregel som blir brukt to ganger. Transformasjonsregelen definert i figur-5.16 vil bli utført 2 ganger. En gang for den original transformasjonen og en gang etter shift operasjonen er gjennomført. Hvis vi benytter oss av denne transformasjonsregelen på datasettet i figur-5.2 sammen med spesifikasjonene i figur-5.4 og figur-5.11 vil grafen kunne se ut som vist i figur-5.17.

5.4 Uttrykkskraft

Dokumenter man ønsker å konvertere kommer i mange forskjellige former med varierende innhold. Det er derfor viktig at templatspråket inneholder nok uttrykkskraft til å dekke de fleste behov samt gi brukerne mulighet til å styre hvilke deler man ønsker å konvertere.

```
<http://www.example.com/kunde#101> rdf:type ex:Kunde ;
  ex:region "Akershus"^^xsd:string ;
  ex:firstnake "Ola"^^xsd:string .
  ex:mobile <http://www.example.com/phone/mobile/no
    /999191919> ;
  ex:mobile <http://www.example.com/phone/mobile/no
    /999191918> .

<http://www.example.com/phone/mobile/no/999191919>
  rdf:type ex:PhoneNumber .
<http://www.example.com/phone/mobile/no/999191918>
  rdf:type ex:PhoneNumber .
```

Figur 5.17: Eksempelgraf etter gjenbruk av en objektspesifikasjon

Transformasjonsreglene beskrevet i kapittel 5.3 tillater oppbygning av subjekter enten som blanke noder eller navngitte ressurser, samt regler for transformering av celleverdier enten til objekter eller typede literaler som man knytter mot subjektene. Dette alene gir nok uttrykkskraft til å transformere og representere flate tabeller.

Transformasjonsreglene gir samtidig mulighet til å konstruere mer komplekse objekter. Dette er objekter som enten er navngitte ressurser eller blanke noder som igjen har typede literaler eller navngitte objekter knyttet til seg. Ved hjelp av de avanserte objektreglene har man nok uttrykkskraft til å transformerer og representere begrensede multidimensjonale tabeller.

Med begrensede multidimensjonale tabeller henvises det til hvor dype RDF grafer som kan konstrueres ved å benytte seg av transformasjonsreglene i templatspråket. Ettersom objektspesifikasjonen ikke har en rekursiv struktur slik at man kan utbrodere et objekt i en omgang vil man ikke kunne representere dypere grafer enn vist i figur-5.18. En rekursiv objektspesifikasjon hvor man ikke har noen begrensning for hvor dype grafer som kan uttrykkes ved objektspessifikasjoner er en foreslått utvidelse av RDFizer (se kapittel 6.4).

Figur 5.18: Dybdebegrensning for representasjon av multidimensjonale tabeller

5.5 Eksempler på templater

Forskjellen mellom flate og multidimensjonale tabeller er beskrevet i kapittel 4.2.1. I neste avsnitt vises det et eksempel på et templat for å transformere en flat tabell, mens kapittel 5.5.2 viser eksempel på et templat for å transformere en multidimensjonal tabell.

5.5.1 Flate tabeller - Partifinansiering

Alle partier i Norge er lovpålagt etter partiloven å oppgi sine inntekter til et sentralt register. Alle partier finansieres på forskjellige måter og partifinansiering.no publiserer en oversikt over dette i Excel regneark. For å vise hvordan transformeringen av flate tabeller kan gjennomføres har jeg valgt å bruke et utdrag fra dette dokumentet som eksempel. Har i datasettet, vist i tabell-5.3, gjort små endringer i forhold til det originale dokumentet for å gjøre det mer oversiktelig. Dette innebærer endring av overskrifter samt fjerning av enkelte kolonner. Under er en liste over de dataene i partifinansieringsdatasettet vi ønsker å ta med i det ferdig transformerte RDF dokumentet.

- Partilagets navn
- Partikode
- Statlig støtte
- Kommunal støtte
- Kapitalinntekter
- Bidrag fra privatpersoner

I henhold til grammatikken for templatspråket defineres først hvilke prefikser vi ønsker å benytte. En del av disse prefiksene er påkrevet slike som "rdf", "smc", "rdfs", "xsd" og "owl". I tillegg til dette ønsker vi å benytte oss av DublinCore (se kapitel 3.3.1) for metadata og vokabularet Friend of a Friend[2] for navn. I fravær av URI'er for offenlige sektor benyttes en fiktiv URI for å representere dette gjennom `http://www.difi.no/vocab/partier#` [3] og benytter seg av prefiksen "difi". For representasjon av skattbare verdier benyttes skatteetaten[4] som vokabular med prefiks "skd". Se figur-5.19 for

[2] `http://www.foaf-project.org/`
[3] `http://www.difi.no/` - Direktoratet for forvaltning og IKT
[4] `http://www.skatteetaten.no`

	A	B	C	D	E	F	G
1							
2	**Partilagets navn**	**Partikode**	**Kommunenr**	**Statsstøtte**	**Kommunalstotte**	**Kapitalinntekter**	**Bidrag privat**
3	Kristelig Folkeparti	001		16,615,864	0	865,606	2,690,282
4	Østfold Kristelig Folkeparti	001		251,702	84,337	0	0
5	Halden KrF	001	0101	12,563	0	102	0
6	Moss KrF	001	0104	4,791	14,437	567	0
7	Sarpsborg KrF	001	0105	26,143	0	38	0
8	Fredrikstad KrF	001	0106	20,629	16,484	64	26,593
9	Aremark KrF	001	0118	3,472	9,794	127	0
10	Marker KrF	001	0119	0	15,320	126	0
11	Rømskog KrF	001	0121	0	0	0	0
12	Trøgstad KrF	001	0122	3,159	5,447	34	0
13	Spydeberg KrF	001	0123	0	0	0	0
14	Askim KrF	001	0124	0	11,928	25	0
15	Eidsberg KrF	001	0125	12,061	0	1,049	2,733
16	Skiptvet KrF	001	0127	9,057	0	78	0
17	Rakkestad KrF	001	0128	9,710	10,398	51	0
18	Råde KrF	001	0135	5,483	19,790	138	0
19	Rygge KrF	001	0136	0	0	0	0
20	Våler KrF	001	0137	0	0	0	0
21	Hobøl KrF	001	0138	3,569	5,648	920	0
22	Akershus Kristelig Folkeparti	001		291,229	0	8,878	56,075
23	Vestby KrF	001	0211	0	0	0	0
24	Ski KrF	001	0213	7,482	5,312	0	0
25	Ås KrF	001	0214	4,953	18,125	99	550
26	Frogn KrF	001	0215	0	0	0	0
27	Nesodden KrF	001	0216	0	0	0	0

Tabell 5.3: Flate tabeller - Partifinansiering

definisjonen av prefikser i templatet. Ettersom datasettet i tabell-5.3 ikke inneholder noen kolonne som egner seg til bruk som subjekter, defineres subjektene som blanke noder.

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
   .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix smc:  <http://www.semicolon.no/excellence#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix dc:   <http://purl.org/dc/elements/1.1/> .
@prefix difi: <http://www.difi.no/vocab/partier#> .
@prefix skd:  <http://www.skattedirektoratet.no/vocab/> .
@prefix foaf: <http://xmlns.foaf/0.1/> .

[] rdf:type smc:subject .
```

Figur 5.19: Definering av prefikser og subjekt

I tillegg til å definere subjektet ønsker vi å inkludere metadata for å beskrive det ferdig transformerte dokumentet og benytter oss av DublinCore for å definere dette, se figur-5.20. I tillegg til dette defineres noen enkle prosesseringsinstrukser for å ikke inkludere den første blanke raden samt raden som inneholder tittel for kolonnene.

```
[] rdf:type smc:Sheet ;
 dc:source "http://www.partifinansiering.no";
 dc:publisher "http://www.regjeringen.no/nb/dep/fad.html";
 dc:coverage "Kommunevis, 2009".

smc:selection smc:offset "2"^^xsd:int .
```

Figur 5.20: Metadata of prosesseringsinstrukser

Videre ønsker vi å knytte noen av verdiene i dokumentet til subjektet som navngitte ressurser. I dette eksemplet ønsker vi å gjøre dette for "kommunenummer" og "partikode" vist i figur-5.21.

```
difi:partikode rdf:type owl:ObjectPropery ;
  smc:ObjectSpec [
    smc:prefix "http://www.difi.no/vocab/partier/" ;
    smc:index "C1" ] .

difi:knum rdf:type owl:ObjectPropery ;
  smc:ObjectSpec [
    smc:prefix "http://www.difi.no/vocab/
      forvaltningsenheter/kommune/knum#"
    smc:index "C3" ] .
```

Figur 5.21: Objektspesifikasjon - Navngitte ressurser

De siste dataene vi ønsker å knytte mot subjektet under transformasjonen av datasettet er literalene. Her er vi ute etter partiets navn, den statlige og kommunale støtten motatt, kapitalinntektene samt bidrage fra privatpersoner vist i figur-5.22.

Når vi da appliserer disse reglene mot datasettet vist i tabell-5.3 vil RDFizer produsere grafer eksempelvis som den i figur-5.23.

```
foaf:name rdf:type owl:DatatypePropery ;
 rdfs:range xsd:string ;
 smc:index "C0" .

skd:statstotte rdf:type owl:DatatypePropery ;
  rdfs:range xsd:double ;
  smc:index "C3" .

skd:kommunestotte rdf:type owl:DatatypePropery ;
  rdfs:range xsd:double ;
  smc:index "C4" .

skd:kapitalinntekter rdf:type owl:DatatypePropery ;
  rdfs:range xsd:double ;
  smc:index "C5"

skd:privat rdf:type owl:DatatypePropery ;
  rdfs:range xsd:double ;
  smc:index "C6"
```

Figur 5.22: Literalspesifikasjon - Typede literaler

```
[] difi:partikode <http://www.difi.no/vocab/partier/001> ;
   difi:knum <http://www.difi.no/vocab/forvaltningsenheter/
     kommune/knum#0106> ;
   foaf:name "Fredrikstad KrF"^^xsd:string ;
   skd:statstotte "20,629"^^xsd:double ;
   skd:kommunalstotte "16,484"^^xsd:double ;
   skd:kapitallinntekter "64"^^xsd:double ;
   skd:private "26,593"^^xsd:double .
```

Figur 5.23: Eksempelgraf etter kovertering av data i tabell-5.3

5.5.2 Multidimensjonale tabeller - Statistikk fra Statistisk Sentralbyrå

I mange tilfeller er statistisk data fremstilt i multidimensjonale tabeller hvor for eksemel verdier er fordelt på år. For å vise hvordan man benytter seg av temlatspråket for å transformere data lagret i multidimensjonale tabeller er det her hentet inn et eksempel fra Statistisk Sentalbyrå[5]. Datasettet er hentet

[5]http://www.ssb.no/

fra tabell 07184 som tar for seg husholdninger etter størrelse på samlet inntekt fordelt på region og år, vist i figur-5.4. Under er en liste over de dataene vi er ute etter å ha med i det ferdig transformerte RDF dokumentet.

- Region
- Statistikkvariabel
- Grupperte statistiske data

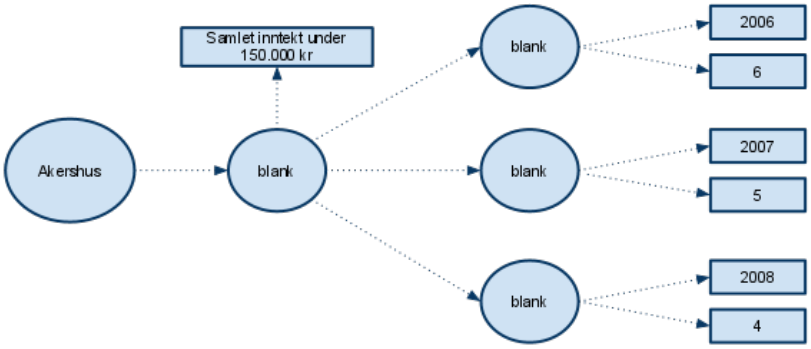
	A	B	C	D	E
1	Husholdninger, etter region, statistikkvariabel og tid				
2					
3					
4	**Region**	**Statistikkvariabel**	**2006**	**2007**	**2008**
5	Akershus	Samlet inntekt under 150 000 kr	6	5	4
6	Akershus	Samlet inntekt 150 000 - 249 999 kr	11	10	9
7	Akershus	Samlet inntekt 250 000 - 349 999 kr	13	11	10
8	Akershus	Samlet inntekt 350 000 - 449 999 kr	12	12	11
9	Akershus	Samlet inntekt 450 000 - 549 999 kr	10	9	9
10	Akershus	Samlet inntekt 550 000 - 749 999 kr	19	17	16
11	Akershus	Samlet inntekt 750 000 - 999 999 kr	16	17	17
12	Akershus	Samlet inntekt 1 000 000 kr og over	15	18	22
13	Oslo	Samlet inntekt under 150 000 kr	11	10	9
14	Oslo	Samlet inntekt 150 000 - 249 999 kr	17	15	14
15	Oslo	Samlet inntekt 250 000 - 349 999 kr	17	15	14
16	Oslo	Samlet inntekt 350 000 - 449 999 kr	14	14	14
17	Oslo	Samlet inntekt 450 000 - 549 999 kr	9	10	10
18	Oslo	Samlet inntekt 550 000 - 749 999 kr	13	13	13
19	Oslo	Samlet inntekt 750 000 - 999 999 kr	10	11	11
20	Oslo	Samlet inntekt 1 000 000 kr og over	10	12	14

Tabell 5.4: Multidimensjonale tabeller - Statistikk fra SSB

Ettersom datasettet i tabell-5.4 ikke inneholder noen kolonner med verdier som egner seg til subjekt benytter vi oss av en blank node. Det kan diskuteres om den første kolonnen kan benyttes som subjekt. Grafen vist i figur-5.24 kunne da ha vært en ønskelig representasjon etter at datasettet var transformert, men som vist i kapittel 5.4 er det begrenset hvor dype multidimensjonale tabeller som kan transformeres med RDFizer. Figur-5.25 viser prefiksdefinisjonene og subjektspesifikasjonen. Her benytter "ssb" som prefiks for et fiktivt vokabular på samme måte som med "difi" i forrige avsnitt.

På samme måte som eksemplet i kapittel 5.5.1 med flate tabeller ønsker vi å legge ved noen enkle prosesseringsinstrukser for å hindre transformeringen av de innledende radene. Disser radene inneholder informasjon som ikke skal være med i det endelige transformerte dokumentet. Vi vil i tillegg definere metadata som skal knyttes mot det ferdig transformerte RDF dokumentet vist i figur-5.26.

Vi ønsker i figur-5.27 å knytte statistikkvariablen i klartekst opp mot subjektet gjennom predikatet "ssb:varname".



Figur 5.24: Alternativ representasjon av datasette i tabell-5.4

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
   .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix smc:  <http://www.semicolon.no/excellence#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix dc:   <http://purl.org/dc/elements/1.1/> .
@prefix ssb:  <http://www.ssb.no/vocab/> .

[] rdf:type smc:subject .
```

Figur 5.25: Prefikser og subjektspesifikasjon

```
[] rdf:type smc:Sheet ;
  dc:source "http://statbank.ssb.no/";
  dc:publisher "http://www.ssb.no";
  dc:coverage "Region,Statistikkvariabel".

smc:selection smc:offset "3"^^xsd:int .
```

Figur 5.26: Metadata og prosesseringsinstrukser

```
ssb:varname rdf:type owl:DatatypePropery ;
  rdfs:range xsd:string ;
  smc:index "C1" .
```

Figur 5.27: Literalspesifikasjon

I tillegg til dette ønsker vi å lage en enkel objektspesifikasjon, figur-5.28, som transformerer verdiene i den første kolonnen om til et objekt som knyttes

```
ssb:region rdf:type owl:ObjectPropery ;
  smc:ObjectSpec [
    smc:prefix "http://www.example.com/norway/region/" ;
    smc:index "C0" ] .
```

Figur 5.28: Enkel objektsepsifikasjon

til subjektet gjennom predikatet "ex:region".

Transformeringen av statistikkolonnene skal lages som blanke noder med årstall og verdi knyttet til seg. Disse blanke nodene skal igjen kobles til subjektet gjennom predikatet "ssb:stat". Siden vi her skal lage den samme reglen for alle de tre siste kolonnene kan vi benytte oss av "smc:shift" predikatet for å gjenta objektspesifikasjonen et gitt antall ganger. I dette tilfellet vil regelen appliserers 3 ganger, en for den originale transformasjonene og to ganger mens reglen flyttes. Objektspesifikasjonen er vist i figur-5.29.

```
ssb:stat rdf:type owl:ObjectPropery ;
  smc:ObjectSpec [
    rdf:type smc:blankNode ;
    ex:year [
      rdf:type smc:expression ;
      rdfs:range xsd:string ;
      smc:index "C2–R3" ] ;
    ssb:value [
      rdf:type smc:expression ;
      rdfs:range xsd:int ;
      smc:index "C2" ]] ;
  smc:shift [
    rdf:type smc:expression ;
    smc:length "1"^^xsd:int ;
    smc:duration "2"^^xsd:int ] .
```

Figur 5.29: Avansert objektspesifikasjon

Etter disse transformeringsreglene har blitt applisert til datasettet fra SSB vist i tabell-5.4, blir den resulterende grafen seendes ut som vist i figur-5.30.

```
[] ssb:region <http://www.example.com/norway/region/
  Akershus> ;
  ssb:varname "Samlet inntekt under 150.000 kr"^^xsd:string
     ;
  ssb:stat [
    ex:year "2006"^^xsd:string ;
    ssb:value "6"^^xsd:int ] ;
  ssb:stat [
    ex:year "2007"^^xsd:string ;
    ssb:value "5"^^xsd:int ] ;
  ssb:stat [
    ex:year "2008"^^xsd:string ;
    ssb:value "4"^^xsd:int ] .
```

Figur 5.30: Eksempelgraf etter konvertering av datasettet i tabell-5.4

5.6 Virkemåte

RDFizer tar imot Microsoft Excel regnearket man ønsker å transformere og et templat som spesifiser hvordan transformeringen skal gjennomføres. Prosesseringsmotoren tar imot disse filene og sender templatet til templatmotoren.

Templatmotoren vil benytte SPARQL spørringer for å hente ut subjektspesifikasjonen (hvordan subjektene skal konstrueres), prosesseringsinstrukser (for enkel styring av transformeringen) og metadata (som skal knyttes til det ferdig transformerte RDF dokumentet). Når dette er hentet vil templatmotoren trekke ut reglene som bestemer hvordan data skal knyttes til subjektene. Disse reglene blir prosessert og lagt i en struktur ut ifra hvilke type ressurser de knytter til subjektene. Prosesseringsmotoren vil da begynne å traversere regnearket som skal transformerers ut ifra prosesseringsinstruksene hentet fra templatmotoren. For hver nye rad i regnearket vil prosesseringsmotoren konstruere et subjekt ut ifra spesifikasjonene hentet fra templatmotoren og benytte seg av regelstrukturen bygget opp i templatmotoren for å knytte verdier til subjektet. Verdiene fra regnearket vil bli knyttet til subjektet som enten blanke noder, navngitte objekter eller typede literaler.

Når prosesseringsmotoren er ferdig med å transformere regnearket vil metadataene angitt i templatet bli hentet fra templatmotoren og knyttet til det ferdig transformerte RDF dokumentet, og RDF dokumentet returneres.

5.7 Sammenligning

I kapittel 4 ble det gjennomgått flere applikasjoner for å transformere trabulære data til RDF. I likhet med ConvertToRDF benytter RDFizer seg av regelbasert transformering av rader, kolonner og celler til RDF. Dette gir en enkel forståelse av hvordan subjekter opprettes og hvordan data fra regnearket knyttes til subjektene igjennom predikater som objekter eller literaler.

Et av kravene for templatspråket benyttet av RDFizer er at det skal benytte seg av en serialiseringsmetode for RDF som er standard, eller de facto standard. I XLWrap benyttes TriG, mens i ConvertToRDF benyttes et eget designet format for å representere hvordan transformasjonen av data gjennomføres. Templater benyttet av RDFizer uttrykkes i Turtle som ikke er en standrad i seg selv, men en de facto standard, som er basert på Notation3 serialisering. Ved å ikke benytte seg av TriG syntaks for å uttrykke templatgrafer vil man ikke kunne oppnå den samme funkasjonaliteten som man kan uttrykke ved TriG i XLWrap, men standarder er viktig for å utvikle den semantiske veven slik at data ikke er avhengig av egne motorer for å evaluere og uttrykke RDF-grafer.

RDF123 og ConvertToRDF baserer seg på transformering av data i rader over til subjekter og objekter. RDFizer kan uttrykke det samme, men muligheten til å hente verdier fra bestemte celler innenfor et regneark muliggjør datatransformering på et annet plan enn disse transformasjonsprogrammene. Dette vil si at man kan uttrykke verdier som ikke er til stedet i raden som blir prosessert, men heller i en bestemt rad.

Kapittel 6

Videreutvikling av RDFizer

I forrige kapittelet ble det gjennomgått hvilke muligheter som er tilgjengelig for transformering av Excel regneark ved å benytte seg av RDFizer. Under utviklingen av dette programmet har det kommet frem flere aspekter som kan være ønskelig å videreutvikle eller implementere for å gi større fleksibilitet i hvordan transformeringen av regneark gjennomføres. I mange utviklingsprosesser dukker det ofte opp behov som medfører ekstra arbeid og utsettelser hvis de skal bli gjennomført. Under testing av RDFizer ble det funnet flere begrensninger og tanker om utvidelser som kan være nyttige å gjennomføre for å utvide uttrykkskraften og funksjonaliteten i applikasjonen. For å kunne gjennomføre utviklingen av applikasjonen har noen av disse utvidelsene blitt sett bort ifra og har ikke blitt implementert i den endelige versjonen av RDFizer.

Videre i dette kapittelet er en gjennomgang av noen av de sentrale ideene for utvidelse av funksjonalitet og uttrykkskraft i RDFizer. I de tilfellene dette er aktuelt vil det være inkludert hvordan dette kan representeres i templatspråket.

6.1 Ekskludering

I templatspråket kan man definere hvor man vil starte og hvor man vil avslutte transformeringen av et regneark ved å sette verdier for prosesseringsinstruksene "smc:offset" og "smc:eof". På denne måten ekskluderer man rader i starten og slutten man ikke ønsker skal være med i resultatet av transformeringen. I noen situasjoner kan det derimot hende at rader som forkommer innenfor den delen av et dokument gitt av "smc:offset" og "smc:eof" ikke skal være med i resultatet. I slike tilfeller kan det være nyttig å kunne definere hvilke rader dette gjelder som en del av prosesseringsinstruksene i templa-

tet. I figur-6.1 viser et forslag for hvordan man kunne representert dette i et templat.

```
smc:selection smc:exclude "10"^^xsd:int ;
smc:selection smc:exclude "11"^^xsd:int ;
smc:selection smc:exclude "12"^^xsd:int ;
```

Figur 6.1: Ekskludering av spesifikke rader som ikke skal være med etter transformeringen av et regneark

På samme måte som ved andre prosesseringsinstrukser kan "smc:selection" benyttes som subjektet i trippelet hvor man definerer hvilke rader som ikke skal inkluderes. Predikatet benyttet for å angi verdien kan være "smc:exclude" og kan forekomme som predikat for subjektet uendelig mange ganger. Verdiene gitt av "smc:exclude" burde refererer til de faktiske radnummerene innenfor et regneark og er ikke satt relativ til verdien av "smc:offset". I figur-6.1 vil verdiene som ligger i radene 10, 11 og 12 ikke bli tatt med i det endelige resultatet av transformeringen.

6.2 Formater

I kapittel 4 har det blitt gjennomgått flere forskjellige applikasjoner og metoder for å representere og transformere data fra forskjellige formater til RDF. Måten RDFizer er bygget opp gjør at man kan trekke ut templatmotoren som evaluerer templater og benytte denne i utviklingen av andre verktøy og applikasjoner. Dette medfører at motoren enkelt kan benyttes som basis i transformasjonsapplikasjoner som transformerer dokumenter i andre formater en Excel. Begrensningen for utvikling av slike applikasjoner vil kun være basert på tilgjengeligheten av utviklings API'er for de spesifikke formatene.

En applikasjon for konvertering av tegnseparerte dokumenter vil da ikke være vanskelig å utvikle, ettersom man kan benytte seg av allerede eksisterende funksjonalitet i Java for å lese dokumenter og dele linjer ut ifra gitte skilletegn. Ettersom flere benytter seg av gratis programmer kan det også være ønskelig å kunne konvertere regneark som for eksempel er laget i OpenOffice.

6.3 Faste cellereferanser

Templatspråket gir muligheten til å definere at objektregler skal gjenbrukes for andre kolonner enn de kolonnene regelen allerede er definert for gjennom predikatet "smc:shift". Når en slik regel blir gjenbrukt vil alltid kolonnereferansene bli flyttet i tråd med verdiene definert av "smc:shift". Dette kan i noen tilfeller være uønsket og en funksjonalitet hvor man i templatspråket kunne definert at "smc:index" alltid skal referere til en fast kolonne, kan være ønskelig. Et forslag her er å markere dette på samme måte som man markerer kolonner og rader for predikatet "smc:index", men benytte seg av en annen nøkkel, for eksempel "S".

```
ex:firma rdf:type owl:ObjectPropery ;
 scm:ObjectSpec [
    rdf:type ex:Firma ;
    smc:prefix "http://www.example.com/" ;
    smc:index "S3" ;
    ex:telefon [
      rdf:type smc:expression ;
      rdfs:range xsd:string ;
      smc:index "C4" ] ] ;
 smc:shift [
    smc:length "1"^^xsd:int ;
    smc:duration "2"^^xsd:int ] .
```

Figur 6.2: Faste cellereferanser

Figur-6.2 er et eksempel for hvordan dette kan benyttes når man definerer en transformasjonsregel for et objekt som flyttes ut ifra "smc:shift". Her vil objektet generert alltid være en konkatenering av den gitte prefiksen og celleverdien som ligger i kolonnene gitt av "S3". I et hypotetisk datasett vil dette kunne produsere et objekt av celleverdien i kolonne 3 og knytte telefonnumre som ligger i kolonne 4, 5 og til dette objektet.

6.4 Multidimensjonale tabeller

I RDFizer og templatspråket kan man uttrykke multidimensjonale tabeller som beskrevet i kapittel 5.4, om uttrykkskraft. Som beskrevet i dette kapittelet er det restriksjoner på hvor dype multidimensjonale tabeller som kan transformeres og vi ser allerede i dette kapittelet et behov for større uttrykkskraft. En utvidelse av denne strukturen for å kunne uttrykke vilkårlig

dype grafer ut ifra tabeller, hvor et subjekt knyttes til et objekt som igjen knyttes til et objekt, osv. vil være ønskelig. Dette vil være med på å løfte uttrykkskraften til RDFizer betraktelig og vil muliggjøre transformering av regneark hvor data må grupperes flere ganger for hver rad.

```
ex:a rdf:type owl:ObjectPropery ;
 scm:ObjectSpec [
   rdf:type ex:Type ;
   smc:prefix "http://www.example.com/prefixa/" ;
   smc:index "C3" ;
   ex:b [
     rdf:type smc:expression ;
     smc:prefix "http://www.exampel.com/prefixb/" ;
     smc:index "C4" ;
     ex:c [
       rdf:type smc:expression ;
       rdfs:range xsd:string ;
       smc:index "C5" ] ] ] .
```

Figur 6.3: Multidimensjonale tabeller

En måte å representere dette på i templatspråket, hvor vi benytter oss av den allerede eksisterende strukturen, vil da kunne være som vist i figur-6.3.

6.5 Funksjonsbibliotek

RDFizer tillater ikke beregning av verdier slikt som for eksempel aggregering, gjennomsnitt, konkatenering eller konvertering av datatyper. Slik funksjonalitet kan i mange situasjoner være nyttig. Et eksempel hvor slik funksjonalitet kunne vært nyttig er i Byantikvarens gule liste[1]. I denne listen benyttes European Terrestrial Reference System (ETRS89) for å uttrykke posisjonene for verneverdige bygg. Hvis man ønsker å benytte seg av disse koordinatene i systemer som kun benytter seg av lenge og breddegrader, longitude and latitude, kunne det være ønskelig å benytte seg av funksjonalitet som gjør dette for seg. Eksempler på hva som kan inngå i et funksjonsbibliotek kan være:

- mulighet til å konkatenere celleverdier hentet fra to forskjellige kolonner. Dette kan for eksempel være nyttig hvis man får oppgitt et fylkesnummer og et kommunenummer hvor ingen av dem kan benyttes

[1]Byantikvarens gule liste er en oversikt over registrerte verneverdige kulturminner og kulturmiljøer i Oslo. – `http://www.byantikvaren.oslo.kommune.no/gul_liste/`

alene under transformeringen som subjekt. Konkatenering av disse verdiene er godt egnet til akkurat dette formålet.

- konvertering av verdier fra et format til et annet. Dette kan for eksempel være en kolonne som inneholder dato og klokkeslett som man i stedet ønsker representert kun som dato, eller konvertering mellom desimaltall og heltall.

I tillegg til dette kunne en mekanisme hvor en bruker selv definerer funksjoner, som blir benyttet under transformeringen av celleverdier til literaler eller objekter være nyttig. Dette kan for eksempel være matematiske uttrykk hvor variabler i uttrykket blir byttet ut med verdier hentet fra celler. På denne måten er ikke brukeren begrenset til funksjoner allerede definert i et funksjonsbiblioteket. En representasjon hvor vi ønsker å konvertere en celleverdi lagret i en kolonne fra desimaltall til heltall kan da være som vist i figur-6.4.

```
ex:heltall rdf:type scm:DatatypePropery ;
  rdfs:range xsd:int ;
  smc:index "C2" ;
  smc:function "double_to_int" .
```

Figur 6.4: Konvertering av desimaltall til heltall

Figur-6.5 viser et eksempel på hvordan man eventuelt kunne definert en matematisk funksjon som evaluerers før det blir knyttet til subjektene.

```
ex:heltall rdf:type scm:DatatypePropery ;
  rdfs:range xsd:int ;
  smc:math "C2*C3" .
```

Figur 6.5: Egendefinert matematisk uttrykk.

Ved å implementerer et funksjonsbibliotek eller muligheten til å definere egne funksjoner i RDFizer vil føre til at man mister mye av den todelte strukturen applikasjonen har i dag. Templatmotoren utfører ingen funksjoner på celleverdiene, noe som innføringen av slike muligheter vil forandre. Den todelte strukturen er viktig hvis man ønsker å benytte seg av motoren i andre applikasjoner enn RDFizer (se kapittel 6.2).

6.6 Validering av templater

Slik RDFizer fungerer i dag eksisterer det ikke noen form for validering av templatet før det blir benyttet under transformering av et regneark. Dette fører til at applikasjonen prøver å transformere et regneark ut ifra et templat uten noen indikasjon på om templatet er riktig strukturert og inneholder de instruksene som er påkrevet eller ikke. En ontologi for evaluering av templater før det blir benyttet for transformering kunne derfor vært ønskelig.

Tillegg A

Javadokumentasjon til RDFizer

Videre er det en dokumentasjon av klassene benyttet i RDFizer applikasjonen, og er generert gjennom javadoc. Klassene som benyttes av applikasjonen er:

- ***RDFizer*** dokumenter på side 88 – Denne klassen er benyttes som tilkoblingspunkt for applikasjonen og tar for seg traversering av Microsoft Excel regneark og henter ut verdier fra rader. Denne klassen benytter seg av klassene ***Template*** og ***TemplateRule*** for å transformere et regneark til RDF.
- ***RDFizer.RDFizerHolder*** dokumenter på side 91 – Denne klassen returnerer en instans av klassen ***RDFizer***.
- ***Template*** dokumenter på side 92 – Denne klassen henter ut regler som som skal benytter for å transformerer et regneark fra et gitt templat. Klassen henter ut reglene for hvordan verdier skal knyttes til subjektene og bygger opp en regelstruktur ved å benytte seg av klassen ***TemplateRule***.
- ***TemplateRule*** dokumenter på side 96 – Denne klassen blir benyttet av klassen ***Template*** for å holde reglene som er hentet fra et transformasjonstemplat. Disse regelen blir igjen benyttet av klassen ***RDFizer*** for å transformere et regneark.
- ***TemplatRule.Tyles*** dokumenter på side 104 – Denne nummertypen benyttes av klassen ***TemplateRule*** for å identifiserer de forskjellige regeltypene. Dette kan være enten blank node, navngitte objekter eller typede literaler.

no.semicolon.excellence

Class RDFizer

java.lang.Object
  └ **no.semicolon.excellence.RDFizer**

public class **RDFizer**
extends java.lang.Object

Class for transforming a Excel document to RDF using a given template file. The RDFizer uses the rules extracted by the Template class to transform the cell values in the document.

Nested Class Summary

static class	**RDFizer.RDFizerHolder** Holder for a RDFizer instance.

Constructor Summary

RDFizer() Empty constructor for RDFizer.

Method Summary

com.hp.hpl.jena.rdf.model.Model	**convertSheet**(java.lang.String inputFile) Converts an Excel spreadsheet to rdf according to the specifications in a template file.
static RDFizer	**getInstance**() Return a instance of RDFizer.
java.lang.String	**getUpdateRequest**() Returns the update request createt from the metaTriples.
static void	**main**(java.lang.String[] argv) To run the program from commandline.
void	**setTemplate**(Template template) Sets the RDFizers template.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

RDFizer

```
public RDFizer()
```

Empty constructor for RDFizer.

Method Detail

getInstance

```
public static RDFizer getInstance()
```

Return a instance of RDFizer.

Returns:
a RDFizer instance.

---

setTemplate

```
public void setTemplate(Template template)
```

Sets the RDFizers template.

Parameters:
`template` - the template to use with the RDFizer.

---

getUpdateRequest

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

Returns the update request createt from the metaTriples.

Returns:
the update request createt from the metaTriples.

---

convertSheet

```
public com.hp.hpl.jena.rdf.model.Model convertSheet(java.lang.String inputFile)
                                             throws java.io.IOException,
                                                    org.apache.poi.openxml4j.exceptions.InvalidFormatException
```

Converts an Excel spreadsheet to rdf according to the specifications in a template file.

Parameters:

`inputFile` - the spreadsheet to convert

Returns:

An RDF representation of selected columns from input file.

Throws:

`java.io.IOException`

`org.apache.poi.openxml4j.exceptions.InvalidFormatException`

---

main

```
public static void main(java.lang.String[] argv)
```

To run the program from commandline.

Parameters:

`argv` - arguments to program

---

no.semicolon.excellence

Class RDFizer.RDFizerHolder

java.lang.Object
  └ **no.semicolon.excellence.RDFizer.RDFizerHolder**

Enclosing class:
RDFizer

---

public static class **RDFizer.RDFizerHolder**
extends java.lang.Object

Holder for a RDFizer instance.

Constructor Summary

RDFizer.RDFizerHolder()

Method Summary

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

RDFizer.RDFizerHolder

public **RDFizer.RDFizerHolder**()

---

no.semicolon.excellence

Class Template

java.lang.Object
  └ **no.semicolon.excellence.Template**

public class **Template**
extends java.lang.Object

Class for extracting rules from a given template. These rules are used to create the finished RDF document.

Method Summary

java.util.ArrayList<TemplateRule>	**getCells**() Return the Rules that are extracted from the template file.
int	**getEof**() Enables the RDFizer to skip rows from end of file
com.hp.hpl.jena.rdf.model.StmtIterator	**getMetaTriples**() Retrieves information about the template.
int	**getOffset**() Enables the RDFizer to skip rows from 0 to offset
int	**getSubjectColumn**() Extracts the index of the column that subjects are to be constructe from.
com.hp.hpl.jena.rdf.model.Resource	**getSubjectDescription**(com.hp.hpl.jena.rdf.model.Resource subject) Add description to a subject resource.
java.lang.String	**getSubjectPrefix**() Returns a prefix that is to be used to construct an RDF subject for each row in the spreadsheet.
com.hp.hpl.jena.rdf.model.Model	**getTemplate**() Return the Jena Model corresponding to the RDF template file.
boolean	**hasMetaTriples**() Test whether the template has meta triples

boolean	**hasSubjectSpec**() Tests whether the template contains a description of the row subject.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

getSubjectDescription

public com.hp.hpl.jena.rdf.model.Resource **getSubjectDescription**(com.hp.hpl.jena.rdf.model.Resource subject)

Add description to a subject resource.

Parameters:
`subject` - the subject resource you want to describe.
Returns:
the subject with description added.

---

getMetaTriples

public com.hp.hpl.jena.rdf.model.StmtIterator **getMetaTriples**()

Retrieves information about the template. These are triples that are not instrumental to the conversion process. They are distinguishable as properties of an anonymous smc:Sheet object.

Returns:
A statement iterator containing the metatriples.

---

getOffset

public int **getOffset**()

Enables the RDFizer to skip rows from 0 to offset

Returns:
the row number at which processing of the sheet is to start.

---

getEof

public int **getEof**()

Enables the RDFizer to skip rows from end of file

Returns:
the row number at which processing of the sheet is to stop.

---

getSubjectColumn

```
public int getSubjectColumn()
```

Extracts the index of the column that subjects are to be constructe from.

Returns:
the index of the subject column

---

getSubjectPrefix

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

Returns a prefix that is to be used to construct an RDF subject for each row in the spreadsheet.

Returns:
the prefix as a possibly empty string.

---

getTemplate

```
public com.hp.hpl.jena.rdf.model.Model getTemplate()
```

Return the Jena Model corresponding to the RDF template file.

Returns:
the Jena Model corresponding to the RDF template file.

---

getCells

```
public java.util.ArrayList<TemplateRule> getCells()
```

Return the Rules that are extracted from the template file.

Returns:
the Rules that are extracted from the template file.

---

hasSubjectSpec

```
public boolean hasSubjectSpec()
```

Tests whether the template contains a description of the row subject.

Returns:
true if yes, false if no.

---

hasMetaTriples

```
public boolean hasMetaTriples()
```

Test whether the template has meta triples

Returns:
true if yes, false if no

---

no.semicolon.excellence

Class TemplateRule

java.lang.Object
└ **no.semicolon.excellence.TemplateRule**

public class **TemplateRule**
extends java.lang.Object

A class for holding transformation rules extracted from a template. These rule could represent blank nodes, objects or literals. See the different constructors on how the different types are created.

Nested Class Summary

static class	**TemplateRule.Types** Representing the different types of rules the TemplateRule can represent.

Constructor Summary

TemplateRule(com.hp.hpl.jena.rdf.model.Resource predicate) Constructor for creating a Rule that will create a blank node.
TemplateRule(com.hp.hpl.jena.rdf.model.Resource predicate, com.hp.hpl.jena.rdf.model.Resource datatype, java.lang.String index, java.lang.String value) Constructor for creating a Rule that will create a literal from provided index and data type.
TemplateRule(com.hp.hpl.jena.rdf.model.Resource predicate, java.lang.String index, java.lang.String prefix, com.hp.hpl.jena.rdf.model.Resource type) Constructor for creating a Rule that will create an object using provided index and prefix value.

Method Summary

void	**addSLI**(com.hp.hpl.jena.rdf.model.Resource predicate, com.hp.hpl.jena.rdf.model.Resource dtype, java.lang.String index)

	Add a child rule to the object.
void	**addSLV**(com.hp.hpl.jena.rdf.model.Resource predicate, com.hp.hpl.jena.rdf.model.Resource dtype, java.lang.String value) Add a child rule to the object.
void	**addSPI**(com.hp.hpl.jena.rdf.model.Resource predicate, java.lang.String prefix, java.lang.String index) Add a child rule to the object.
void	**addSPV**(com.hp.hpl.jena.rdf.model.Resource predicate, java.lang.String prefix, java.lang.String value) Add a child rule to the this rule.
java.util.ArrayList<TemplateRule>	**getCells**() Returns the child rules for this rule.
int	**getCol**() Returns the column the rule applies to.
com.hp.hpl.jena.rdf.model.Resource	**getDatatype**() Returns the datatype for the rule.
int	**getDuration**() Returns how many times the rule should be applied.
int	**getLength**() Returns how many cells the shift mechanism jumps each time it is applied.
com.hp.hpl.jena.rdf.model.Resource	**getPredicate**() Returns the predicate for the rule.
java.lang.String	**getPrefix**() Returns the prefix for the rule.
com.hp.hpl.jena.rdf.model.Resource	**getRDFtype**() Returns the rdf:type of the rule.
int	**getRow**() Returns which row the rule applies to.
TemplateRule.Types	**getType**() Returns the type of the rule.
java.lang.String	**getValue**() Returns the static value of this rule.
void	**setDuration**(int duration) Sets how many times the rule should be applied.
void	**setLength**(int length) Sets how many columns the rule should

	jump while shifting.
void	**setValue**(java.lang.String value) Returns the static value of the rule.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

TemplateRule

```
public TemplateRule(com.hp.hpl.jena.rdf.model.Resource predicate)
```

Constructor for creating a Rule that will create a blank node.

Parameters:

`predicate` - the predicate for the rule.

TemplateRule

```
public TemplateRule(com.hp.hpl.jena.rdf.model.Resource predicate,
                    java.lang.String index,
                    java.lang.String prefix,
                    com.hp.hpl.jena.rdf.model.Resource type)
```

Constructor for creating a Rule that will create an object using provided index and prefix value.

Parameters:

`predicate` - the predicate for the rule.
`index` - which cell the rule applies to.
`prefix` - the prefix for the rule.
`type` - the rdf type for the rule.

TemplateRule

```
public TemplateRule(com.hp.hpl.jena.rdf.model.Resource predicate,
                    com.hp.hpl.jena.rdf.model.Resource datatype,
                    java.lang.String index,
                    java.lang.String value)
```

Constructor for creating a Rule that will create a literal from provided index and data type.

Parameters:

`predicate` - the predicate for the rule.
`datatype` - the data type for the rule.
`index` - which cell the rule applies to.
`value` - the rule represents a static value.

Method Detail

addSPV

```
public void addSPV(com.hp.hpl.jena.rdf.model.Resource predicate,
                   java.lang.String prefix,
                   java.lang.String value)
```

Add a child rule to the this rule. The child rule will create a object with a fixed value.

Parameters:

`predicate` - the predicate for the child rule.
`value` - the fixed value for the child rule.
`prefix` - the prefix for the child rule.

---

addSPI

```
public void addSPI(com.hp.hpl.jena.rdf.model.Resource predicate,
                   java.lang.String prefix,
                   java.lang.String index)
```

Add a child rule to the object. The child rule is a object with value extracted from the index param.

Parameters:

`predicate` - the predicate for the child rule.
`index` - which cell the rule applies to.
`prefix` - the prefix for the child rule.

---

addSLV

```
public void addSLV(com.hp.hpl.jena.rdf.model.Resource predicate,
                   com.hp.hpl.jena.rdf.model.Resource dtype,
```

```
                java.lang.String value)
```

Add a child rule to the object. The child rule is a literal with a fixed value given by the value param.

Parameters:

`predicate` - the predicate for the child rule.
`dtype` - the data type of the child rule.
`value` - the fixed value for this rule.

---

addSLI

```
public void addSLI(com.hp.hpl.jena.rdf.model.Resource predicate,
                   com.hp.hpl.jena.rdf.model.Resource dtype,
                   java.lang.String index)
```

Add a child rule to the object. The rule added is a literal with value extracted from a index parameter.

Parameters:

`predicate` - the predicate for the child rule.
`dtype` - the data type of the child rule
`index` - which cell the rule applies to.

---

getValue

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

Returns the static value of this rule. If the rule does not contain a static value an empty string is returned.

Returns:

the static value of the rule.

---

getType

```
public TemplateRule.Types getType()
```

Returns the type of the rule.

Returns:

the type of the rule.

See Also:

TemplateRule.Types

---

getPrefix

public java.lang.String **getPrefix**()

Returns the prefix for the rule. If the rule does not have a prefix an empty string is returned.

Returns:
the preifx for the rule.

See Also:
TemplateRule.Types

---

getPredicate

public com.hp.hpl.jena.rdf.model.Resource **getPredicate**()

Returns the predicate for the rule.

Returns:
the predicate for the rule.

---

getDatatype

public com.hp.hpl.jena.rdf.model.Resource **getDatatype**()

Returns the datatype for the rule. If the rule does not have a datatype null is returned.

Returns:
the datatype for the rule.

---

getRDFtype

public com.hp.hpl.jena.rdf.model.Resource **getRDFtype**()

Returns the rdf:type of the rule. If the rule does not have a type null is returned.

Returns:
the rdf:type of the rule.

---

getRow

```
public int getRow()
```

Returns which row the rule applies to. This used together with the column value is used to represent a static column. If the rule does not spesify a row -1 is returned.

Returns:
which row the rule should be applied to.

---

getCol

```
public int getCol()
```

Returns the column the rule applies to. If the rule does not spesify a column -1 is returned.

Returns:
the column the rule applies to.

---

getDuration

```
public int getDuration()
```

Returns how many times the rule should be applied.

Returns:
how many times the rule should be applied.

---

getLength

```
public int getLength()
```

Returns how many cells the shift mechanism jumps each time it is applied.

Returns:
how many cells the shift mechanism jumps.

---

getCells

public java.util.ArrayList<TemplateRule> **getCells**()

Returns the child rules for this rule. These rules are used to add additional information to the resource created by this rule.

Returns:
the child rules for this rule.

---

setLength

public void **setLength**(int length)

Sets how many columns the rule should jump while shifting.

Parameters:
`length` - how many columns the rule should jump while shifting.

---

setDuration

public void **setDuration**(int duration)

Sets how many times the rule should be applied.

Parameters:
`duration` - how many times the rule should be applied.

---

setValue

public void **setValue**(java.lang.String value)

Returns the static value of the rule.

Parameters:
`value` - the static value for the rule.

---

no.semicolon.excellence

Enum TemplateRule.Types

```
java.lang.Object
  └ java.lang.Enum<TemplateRule.Types>
      └ no.semicolon.excellence.TemplateRule.Types
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Comparable<TemplateRule.Types>

Enclosing class:

TemplateRule

```
public static enum TemplateRule.Types
extends java.lang.Enum<TemplateRule.Types>
```

Representing the different types of rules the TemplateRule can represent.

Enum Constant Summary

BLANKNODE The rule will create a blank node.
DATATYPE The rule will create a Literal.
OBJECTTYPE The rule will create a Object.

Method Summary

static TemplateRule.Types	**valueOf**(java.lang.String name) Returns the enum constant of this type with the specified name.
static TemplateRule.Types[]	**values**() Returns an array containing the constants of this enum type, in the order they are declared.

Methods inherited from class java.lang.Enum
clone, compareTo, equals, finalize, getDeclaringClass, hashCode, name, ordinal, toString, valueOf

Methods inherited from class java.lang.Object
getClass, notify, notifyAll, wait, wait, wait

Enum Constant Detail

DATATYPE

public static final TemplateRule.Types **DATATYPE**

The rule will create a Literal.

OBJECTTYPE

public static final TemplateRule.Types **OBJECTTYPE**

The rule will create a Object.

BLANKNODE

public static final TemplateRule.Types **BLANKNODE**

The rule will create a blank node.

Method Detail

values

public static TemplateRule.Types[] **values**()

Returns an array containing the constants of this enum type, in the order they are declared. This method may be used to iterate over the constants as follows:

```
for (TemplateRule.Types c : TemplateRule.Types.values())
    System.out.println(c);
```

Returns:

an array containing the constants of this enum type, in the order they are declared

---

valueOf

```
public static TemplateRule.Types valueOf(java.lang.String name)
```

Returns the enum constant of this type with the specified name. The string must match *exactly* an identifier used to declare an enum constant in this type. (Extraneous whitespace characters are not permitted.)

Parameters:

`name` - the name of the enum constant to be returned.

Returns:

the enum constant with the specified name

Throws:

`java.lang.IllegalArgumentException` - if this enum type has no constant with the specified name

`java.lang.NullPointerException` - if the argument is null

Bibliografi

[1] Pascal Hitzler, Markus Krotzler, and Sebastian Rudolph. *Foundation of Semantic Web Technologies*. CRC Press, 2009.

[2] Rådet Europa-Parlamentet. Europa-parlamentets og rådets direktiv 2003/98/ef af 17. november 2003 om videreanvendelse af den offentlige sektors informationer. `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32003L0098:DA:NOT#texte`, November 2003.

[3] Universitetet i Bergen. Fakta først – vidrebruk av datakilder i offentlig sektor: potensial og hindring. `http://voxpublica.no/wp-content/uploads/2010/01/fakta_foerst_rapport.pdf`, Januar 2010.

[4] The British Government. Opening up government. `http://data.gov.uk`. Besøkt Februar 2011.

[5] IT og Telestyrelsen. digitaliser.dk. `http://digitaliser.dk/`. Besøkt Februar 2011.

[6] The American Government. Data.gov – empovering people. `http://data.gov/`. Besøkt Februar 2011.

[7] Arbeids- og Administrasjonsdepartementet. Fra bruk til gjenbruk. `http://www.regjeringen.no/upload/kilde/aad/rap/2004/0004/ddd/pdfv/220715-fra_bruk_til_gjenbruk_endelig-web-2-31-08-2004.pdf`, August 2004.

[8] Justis og politidepartementet. Lov 2006-05-19 nr 16: Lov om rett til innsyn i dokument i offentleg verksemd (offentleglova). `http://www.lovdata.no/all/hl-20060519-016.html#1`, Mai 2006.

[9] Moderniseringsdepartementet. enorge 2009 – det digitale spranget. `http://www.regjeringen.no/upload/FAD/Vedlegg/IKT-politikk/enorge_2009_komplett.pdf`, Juni 2005.

[10] Fornyingsdepartementet. data.norge.no – fornyingsdepartementets blogg om viderebruk av offentlige data. `data.norge.no`. Besøkt April 2011.

[11] Avinor. Avinor.no – gratis flydata fra avinor. `flydata.avinor.no`. Besøkt Mars 2011.

[12] Tom Heath and Christian Bizer. *Linked Data – Evolving the Web into Global Data Space*. Morgan and Claypool, første utgave edition, 2011.

[13] Tim Berners-Lee. Linked data – desing issues. `http://www.w3.org/DesignIssues/LinkedData.html/`, Juni 2009.

[14] Frank Manola and Eric Miller. Rdf primer – w3c recommendation. `http://www.w3.org/TR/rdf-primer/`, Februar 2004.

[15] Tim Berners-Lee. Putting government data online. `http://www.w3.org/DesignIssues/GovData.html`, Juni 2009.

[16] Chrisstian Bizer, Tom Heath, and Tim Berners-Lee. Linked data – the story so far. `http://tomheath.com/papers/bizer-heath-berners-lee-ijswis-linked-data.pdf`, Januar 2010.

[17] John Hebeler, Matthew Fisher, Ryan Blace, and Andrew Perez-Lopex. *Semantic Web Programming*. WILEY, 2009.

[18] The World Wide Web Consortium. About w3c. `http://www.w3.org/Consortium/`. Besøkt April 2011.

[19] Elias Torres, Lee Feigenbaum, and Kendall Grant Clark. Sparql protocol for rdf – w3c recommendation. `http://www.w3.org/TR/rdf-sparql-protocol/`, Januar 2008.

[20] Eric Prud'hommeaux and Andy Seaborne. Sparql query language for rdf – w3c recommendation. `http://www.w3.org/TR/rdf-sparql-query/`, Januar 2008.

[21] Toby Seagaran, Colin Evans, and Jamie Taylor. *Programming the Semantic Web*. O'REILLY, første utgave edition, 2009.

[22] Tim Berners-Lee. Metadata architecture. `http://www.w3.org/DesignIssues/Metadata`, Januar 1997.

[23] The Dublin Core Metadata Initiative. Dublin core metadata element set, version 1.1. `http://dublincore.org/documents/dces/`, Oktober 2010.

[24] Keith Alexander, Richard Cyganiak, Michael Hausenblas, and Jun Zhao. Describing linked datasets with the void vocabulary. `http://www.w3.org/TR/void/`, Juni 2009.

[25] Michael Grove. Mindswap convert to rdf tool. `http://www.mindswap.org/~mhgrove/ConvertToRDF/`. Besøkt Mars 2011.

[26] Lushan Han and Cynthia Parr and Joel Sachs and Anupam Joshi. RDF123: a mechanism to transform spreadsheets to RDF. `http://ebiquity.umbc.edu/paper/html/id/368/RDF123-a-mechanism-to-translate-spreadsheets-to-RDF`, August 2007.

[27] Andreas Langegger. Xlwrap – spreadsheet to rdf wrapper. `http://xlwrap.sourceforge.net/`. Besøkt Mars 2011.

[28] Andreas Langegger. Xlwrap – mapping design patterns. `http://xlwrap.sourceforge.net/patterns.html`. Besøkt Mars 2011.

[29] Tim Berners-Lee. Relational databases on the semantic web. `http://www.w3.org/DesignIssues/RDB-RDF.html`, September 1998.

[30] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF Mapping Language. `http://www.w3.org/TR/r2rml/`, Mars 2011.

[31] Chris Bizer, Richard Cyganiak, Jörg Garbers, Oliver Maresch, and Christian Becker. The d2rq platform v0.7 – treating non-rdf relational databases as virtual rdf graphs, spesification. `http://www4.wiwiss.fu-berlin.de/bizer/d2rq/`, November 2010.

[32] Chris Bizer, Richard Cyganiak, Jörg Garbers, Oliver Maresch, and Christian Becker. The d2rq plattform – treating non-rdf databases as virtual rdf graphs. `http://www4.wiwiss.fu-berlin.de/bizer/d2rq/spec/`, August 2009.