UNIVERSITETET I OSLO
Institutt for informatikk

Distribusjon og integrasjon av folkeregisterdata med semantisk teknologi

Masteroppgave

Henrik Wingerei

14. mai 2012

Sammendrag

Norge er i en unik situasjon når det kommer til muligheter for distribusjon og samhandling av offentlige data. Dette skyldes blant annet at Norge, som et av få land, har ett sentralt register som inneholder informasjon om alle som har bodd eller bor i Norge: Det sentrale folkeregister. Potensialet som ligger i mulighetene for informasjonsutveksling og samhandling vanskeliggjøres imidlertid av dagens distribusjonsmodell. Dette kommer konkret til uttrykk hos NAV hvor distribusjonsmodellen kompliserer identifiseringen av personer. I tillegg eksisterer det en rekke andre registre i det offentlige Norge som brukes som supplement til data som ligger i Folkeregisteret, og det er knyttet utfordringer til integrasjon av disse heterogene kildene.

Basert på noen av dagens modeller fra Folkeregisteret og NAV, presenterer oppgaven en modell for distribusjon som anvender semantiske teknologier. Oppgaven viser hvordan folkeregisterdata kan konverteres til RDF på en konservativ måte og videre hvordan ontologier kan utvikles basert på dokumentasjon fra Skattedirektoratet og NAV. Til slutt vises det hvordan en ny distribusjonsløsning som distribuerer folkeregisterdataene gjennom et SPARQL-endepunkt og et RESTfullt grensesnitt, kan implementeres.

Den nye distribusjonsløsningen som er presentert vil gi en fleksibel løsning som blant annet gjør det enklere å integrere heterogene kilder, med et grensesnitt som gjør det mulig å stille kraftige, vilkårlige spørringer. I tillegg vil ontologier med eksplisitte begrepsdefinisjoner og relasjoner mellom disse muliggjøre resonnering, noe som kan føre til at ny kunnskap oppdages.

Arbeidet har vist at det å benytte semantiske teknologier til en distribusjonsløsning krever opplæring og erfaring av både utviklere og brukere av et slikt system. For virkelig å kunne utnytte semantiske teknologier er det viktig at det brukes tid på å utvikle ontologier som fanger semantikken til dataene. Dette er imidlertid et tidkrevende og vanskelig arbeid, og det er heller ikke alt som er enkelt eller mulig å modellere.

Oppgaven viser at distribusjonsløsningen som er presentert gjør at etater og andre aktører kan få tilgang til oppdaterte data, og gjøre det enklere å integrere disse dataene mot andre kilder. I tillegg kan identifiseringsprosessen hos NAV utføres basert på et større og bedre datagrunnlag, samt gi større muligheter for å kontrollere denne prosessen ved å kunne stille vilkårlige kontrollspørringer.

Takk til

Først og fremst vil jeg takke mine to veiledere Martin G. Skjæveland og Arild Waaler. Arild for å komme med en interessant og utfordrende oppgave, samt komme med gode løsninger til vinklinger på oppgaven når ikke alt ble som vi planla. Martin for mange gode og lærerike samtaler og tilbakemeldinger når det gjelder oppgavens innhold, samt gode innspill og ideer knyttet til tekniske utfordringer.

Videre vil jeg takke Berit Bergen fra NAV for å komme med et interessant case som har blitt til en viktig del av oppgaven. En takk rettes også til Boris Schürmann, leder for moderniseringsprogrammet for Det sentrale folkeregister i Skatteetaten, for interessen han har vist for prosjektet og Torunn Robøle fra Skattedirektoratet for testdata fra Folkeregisteret og dokumentasjon av datamodellen bak Det sentrale folkeregister.

Til slutt vil jeg også takke mine studievenner for en flott studietid, og spesielt «høgskolegjengen» — dere vet hvem dere er. Sist, men ikke minst, vil jeg takke min samboer Camilla og mine foreldre for all støtte og oppmuntring dere har kommet med gjennom arbeidet med oppgaven.

Innhold

Figurer

Tabeller

Forord

Tankene rundt denne oppgaven startet i november 2010 da jeg kom i kontakt med min biveileder, Arild Waaler, angående aktuelle oppgaver innenfor fagfeltet semantiske teknologier. En av oppgavene som han foreslo var knyttet til Det sentrale folkeregister (DSF), og jeg syntes dette virket som et spennende og aktuelt case å jobbe videre med. Utgangspunktet var å bruke semantiske teknologier til å dokumentere dagens Folkeregister, og det ble involvert andre masterstudenter som skulle fokusere på denne biten, mens det etterhvert modnet frem en idé om at jeg skulle se på distribusjon av folkeregisterdata. Etterhvert ønsket vi å lage et case hvor DSF, semantiske teknologier og distribusjon var sentrale temaer.

Vi har gjennom hele prosessen hatt god kontakt med moderniseringsprogrammet for Folkeregisteret, som ønsket et slikt case velkommen og var positive til at vi kunne lage en «proof-of-concept»-løsning. Med utgangspunkt i ønsket vi hadde kom vi i januar 2011 i kontakt med Skattedirektoratet (SKD), og startet samtaler med dem for å undersøke mulighetene for å få tilgang til materiale. Det viste seg at dette var mer utfordrende og tidkrevende enn ventet, men i mars 2011 fikk vi tilgang til en brukerhåndbok for Folkeregisteret. Denne fungerer som en sluttbrukermanual for brukere av Folkeregisteret.

Utover våren 2011 fikk vi ikke tilgang til noe mer materiale, men i august 2011 fikk vi tilgang til en dump med anonymiserte folkeregisterdata sammen med en beskrivelse av disse dataene. Materialet som ble mottatt viste seg å ikke være nok til å kunne dokumentere Folkeregisteret på den måten vi ønsket, og fokuset var nå å lage en demonstrator som viste distribusjonsmulighetene semantiske teknologier gir, ved å ta utgangspunkt i dataene som vi mottok. I tillegg hadde vi nå et ønske om å prøve å få på plass en sluttbruker av folkeregisterdata, og dermed kunne lage en «end-to-end»-løsning. Det vil si å vise hvordan den nye distribusjonsløsningen faktisk kunne bli anvendt av eksisterende systemer hos sluttbruker.

For å prøve å komme i kontakt med aktuelle sluttbrukere holdt jeg blant annet en presentasjon for prosjektet Semicolon II[1] i september 2011. Her snakket om jeg status for oppgaven og formidlet ønsket vi hadde om å

[1] `http://www.semicolon.no/`

komme i kontakt med aktuelle sluttbrukere. Vi fikk noen gode ideer her, og vi fikk også tilgang til mer dokumentasjon av DSF fra SKD, men det førte dessverre ikke til noe konkret knyttet til ønsket om sluttbruker.

I november 2011 kom vi i kontakt med NAV, som syntes prosjektet var veldig spennende og vi fikk til et møte med dem i desember samme år. Her presenterte jeg hva jeg hadde gjort hittil, og motivasjonen for å finne en sluttbruker og dermed kunne lage en «end-to-end»-løsning. Underveis i møtet kom vi frem til et problem som kunne være interessant å jobbe videre med, som var knyttet til identifisering av personer i forbindelse med saksbehandlinger.

Etter dette innledende møtet fikk vi til et nytt møte på Institutt for informatikk (Ifi) i januar 2012, hvor Berit Bergen fra NAV forklarte mer rundt detaljene knyttet til problemet som ble skissert på det første møtet. Etter å ha diskutert litt internt på Ifi, kom vi frem til at dette problemet kunne egne seg godt til å vise hvordan den nye distribusjonsløsningen for Folkeregisteret kan gjøre det enklere å integrere heterogene kilder. En annen stor motivasjon for å kunne jobbe med NAV er at de har et ønske om å kunne stille vilkårlige spørringer mot DSF, noe som ikke er mulig i dag.

På bakgrunn av dette fikk vi jobbet frem problemstillingen og problembeskrivelsen som blir presentert i oppgaven. Som vist har det vært en lang prosess å komme frem til det fokuset og retningen oppgaven som presenteres har. Det er også verdt å merke seg at oppgaven har skiftet fokus flere ganger, og mye arbeid har blitt gjort som ikke kommer til direkte uttrykk i denne rapporten.

Kapittel 1

Innledning

Denne masteroppgaven presenterer en ny distribusjonsmodell for Det sentrale folkeregister, som benytter semantiske teknologier. Oppgaven vil vise hvordan en ny distribusjonsmodell kan bli utviklet ved hjelp av disse teknologiene, og videre vise hvordan en spesifikk datakilde kan integreres mot den nye løsningen gjennom et konkret case. Caset vil gjennom oppgaven bli referert til som identifiseringsproblemet, og blir presentert i kapittel 2.

Dette kapittelet beskriver i seksjon 1.1 kort bakgrunnen for problemet, og videre blir problemstillingen presentert i seksjon 1.2. Til slutt i seksjon 1.3 blir oppbyggingen av resten av oppgaven beskrevet.

1.1 Bakgrunn

I Norge er det store mengder persondata som forvaltes av offentlig sektor, og disse dataene brukes svært ofte i forbindelse med saksbehandling. Med persondata mener vi *opplysninger og vurderinger som kan knyttes til en enkelt-person* jf. personopplysningsloven paragraf 2 nr 1. Det sentrale folkeregister er et offentlig register som inneholder persondata om personer som har bodd eller bor i Norge. Det eksisterer en rekke forskjellige datasystemer i forskjellige offentlige etater som bruker persondata, og som derfor har behov for å hente ut og bruke data fra Folkeregisteret. I tillegg har mange etater egne datakilder som brukes som komplement til dataene fra Folkeregisteret, og det er derfor behov for å kombinere data fra flere av disse kildene for å innhente den nødvendig informasjonen for å kunne behandle en sak. Dette medfører at det foregår sammenstilling av data på tvers av både datasystemer og etater.

Norge er i en ganske unik situasjon ved å ha ett sentralt register som inneholder informasjon om alle personer som bor eller har bodd i Norge, og Det sentrale folkeregister gir et stort potensial for utnyttelse av personressurser i forhold til andre land som ikke har et slikt register [1, side 8]. For å utnyt-

te denne informasjonen kreves det en god distribusjonsmodell som legger til rette for gjenbruk og enkel tilgang til dataene, samtidig som sikkerheten ivaretas. I dag distribueres folkeregisterdata via frittstående distributører som mottar daglige overføringer fra Folkeregisteret med alle oppdateringene som er skjedd i databasen siste døgn [2, side 22].

Det er imidlertid flere som mener at potensialet som ligger i mulighetene for informasjonsutveksling og samhandling ikke er utnyttet. Flere rapporter [1, 2, 80] peker blant annet på hvordan denne distribusjonsmodellen sammen med dagens prismodell fører til dårlig utnyttelse av informasjonsmulighetene og et underforbruk av folkeregisterdata. I artikkelen *Dyr og dårlig datahåndtering* [44] påstår Holte at måten det offentlige håndterer grunndata om hver enkelt av oss kan karakteriseres som *dyrt og dårlig*. Holte sier videre hvordan vi som innbyggere blir spurt om den samme informasjonen gjentatte ganger og at denne informasjonen blir lagt inn i forskjellig systemer som i liten grad utveksler informasjon seg imellom.

Det er også knyttet flere utfordringer til sammenstilling av ulike datakilder. Datakildene som skal kombineres er ofte heterogene, det vil si at kildene har forskjellig struktur eller er på forskjellig format, og kan derfor ikke kombineres uten å bearbeide en eller flere av kildene. Tradisjonelle teknologier som stormaskinløsninger og relasjonelle databaser, som det er mye av i offentlig sektor, er lite fleksible og det er ofte store kostnader knyttet til å endre eller utvide strukturen til datakildene for å få dem til å passe sammen med andre datakilder.

Semantiske teknologier og såkalte lenkede data har av flere blitt presentert som teknologier som vil være med på å skape bedre distribusjon og informasjonsutveksling, og dermed gjøre det enklere å integrere datakilder på tvers av systemer og etater. Disse teknologiene inneholder et sett med W3C[1]-standarder og tilbyr blant annet en datamodell, et spørrespråk og et modelleringsspråk. Målet med oppgaven er å anvende semantisk teknologi for å utvikle en ny distribusjonsmodell for Folkeregisteret, og deretter vise gjennom et konkret case hvordan denne nye modellen kan bidra til bedre samhandling på tvers av etater, og føre til enklere dataintegrasjon av heterogene kilder.

1.2 Problemstillinger

Problemstillingen for oppgaven er:

> *Hvordan er semantiske teknologier egnet til å utvikle en ny, forbedret distribusjonsmodell for Folkeregisteret, og hvordan vil denne modellen egne seg til å integrere andre heterogene datakilder?*

Andre spørsmål som vil bli belyst:

[1] World Wide Web Consortium — `http://www.w3.org/`

- Hvordan kan man sikre at rådataene blir beholdt intakte gjennom konverteringsprosessen til RDF?

  Dette spørsmålet blir diskutert detalj i i seksjon 4.2.2 på side 53

- Hvordan kan ontologier bli utviklet basert på dokumentasjon fra Skattedirektoratet og NAV?

  Dette spørsmålet blir diskutert i detalj i seksjon 4.3 på side 61

- Hvordan kan en demonstrator implementeres og hvilke gevinster vil denne gi?

  Dette spørsmålet blir diskutert i detalj i kapittel 5 og 6.

- Hvilke begrensninger gir semantiske teknologier?

  Dette spørsmålet blir diskutert i detalj i kapittel 7.

1.3 Oppbygging av oppgaven

Resten av oppgaven er strukturert som følger:

Kapittel 2 beskriver bakgrunnen for problemet, de forskjellige aktørene og det konkrete caset. Kapittelet diskuterer også problemer med dagens distribusjonsmodell i større detalj og skisserer et forslag til ny løsning.

Kapittel 3 presenterer semantiske teknologier i detalj, og har et særlig fokus på de delene av teknologien som den nye distribusjonsmodellen benytter.

Kapittel 4 beskriver konvertering av rådata til RDF og utviklingen av ontologiene, og hvordan disse kan integreres.

Kapittel 5 beskriver implementasjonsdetaljene til demonstratoren.

Kapittel 6 beskriver gevinstene med den nye løsningen, og viser gjennom flere eksempler resultater som kan oppnås med den nye distribusjonsmodellen sammenlignet med dagens løsning.

Kapittel 7 beskriver utfordringene og begrensningene med demonstratoren og semantiske teknologier generelt som er blitt erfart gjennom utviklingen av løsningen.

Kapittel 8 oppsummerer arbeidet og presenterer mulig videre arbeid.

Kapittel 2

Bakgrunn

Dette kapittelet beskriver bakgrunnen og motivasjonen for å utvikle en ny distribusjonsmodell for Folkeregisteret, og vil også beskrive et case som vi vil anvende for å vise hvordan denne nye distribusjonsmodellen kan brukes til integrasjon av en annen datakilde. Dette caset vil vi referere til som *identifiseringsproblemet*. Caset er godt egnet for å demonstrere mange sider av semantiske teknologier, og hvordan teknologiens særegenheter og muligheter passer godt til å løse deler av identifiseringsproblemet.

Først presenteres de partene som leseren bør ha kunnskap om for å forstå beskrivelsen av problemet. Dette er Det sentrale folkeregister, Skattedirektoratet og NAV. For alle partene gis det en kort generell introduksjon, og hva slags rolle de forskjellige aktørene har i denne sammenhengen. Etter å ha presentert partene blir selve identifiseringsproblemet presentert, og til slutt i kapittelet blir utfordringene med dagens løsning analysert og hovedtrekkene rundt den nye løsningen presentert.

2.1 Det sentrale folkeregister

Det sentrale folkeregister (DSF) er et offentlig register over alle personer som bor eller har vært bosatt i Norge. Informasjonen i registeret brukes som grunnlag for blant annet skatte- og valgmanntallet, og som grunnlag for befolkningsstatistikken, noe som er viktig for planleggingen av offentlige tjenester [78].

Ifølge folkeregisterloven paragraf 1 skal det være ett sentralt Folkeregister for Norge, og kravet for at en person skal være registrert i Folkeregisteret er at personen:

a) er eller har vært bosatt i Norge,

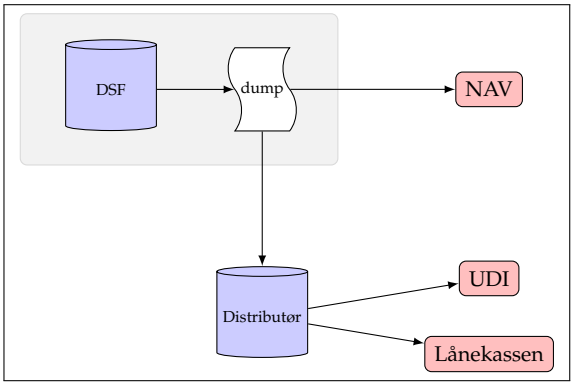
b) er født i Norge,

c) har fått tildelt fødselsnummer eller D-nummer.

DSF inneholder informasjon om både personer som er bosatt i Norge, og personer som ikke er bosatt i Norge. Paragraf 4 i folkeregisterloven sier at for enhver som er bosatt i Norge, fastsettes et fødselsnummer. Fødselsnummeret har elleve siffer og er delt opp i to bolker på seks og fem siffer. De seks første sifrene er fødselsdatoen til personen, mens de fem siste sifrene kalles for et personnummer. Personer som ikke er bosatt i Norge, men som skal være midlertidig i Norge på grunn av for eksempel jobb, får tildelt et D-nummer (dagnummer). Det er bygd opp på samme måte som fødselsnummeret, bortsett fra at fødselsnummeret er modifisert ved at det legges til 4 på det første sifferet.

Mange offentlige etater bruker informasjon fra DSF, og det er derfor et stort behov for å hente ut data fra Folkeregisteret. Figur 2.1 viser et overordnet bilde over hvordan distribueringen av folkeregisterdata fungerer i dag. Data distribueres i de fleste tilfeller gjennom en distributør, og fungerer ved at distributøren daglig mottar en overføring fra Folkeregisteret med de siste døgns endringer [2, side 22]. Distributøren er pliktig til å oppdatere sine data med overføringen fra DSF hver natt, slik at aktører som abonnerer på folkeregisterdata får tilgang til et register med daglig oppdaterte data [2, side 22]. Det er også noen store aktører, for eksempel NAV, som mottar denne overføringen direkte fra Folkeregisteret.

Offentlig myndighet kan innhente enkeltoppslag fra DSF vederlagsfritt, men hvis det er behov for en masseutsendelse, må mottaker dekke utgiftene som påløper [2, side 23]. Innrapportering og meldinger til Folkeregisteret som navne- og adresseendringer er det ikke knyttet gebyrer til [2, side 23].



Figur 2.1: Overordnet bilde av dagens situasjon for distribuering av folkeregisterdata.

2.2 Skattedirektoratet

Skattedirektoratet (SKD) er et forvaltningsorgan som er underlagt Finansdepartementet, og SKD fungerer som hovedkontor for Skatteetaten [68]. I 1991 ble folkeregistrering underlagt SKD [72], og SKD er dermed ansvarlig

for ajourhold av Folkeregisteret, og tildeling og administrering av fødselsnummer og D-nummer.

SKD har en egen gruppe som er ansvarlig for tildeling av fødselsnummer og D-nummer, den såkalte ID-forvaltningsgruppen. Denne er ansvarlig for å behandle saker som omhandler personer som ønsker å få tildelt et D-nummer eller et fødselsnummer. Før en person får tildelt et slikt nummer er det nødvendig å gjennomføre en identifisering av personen, for å finne ut om vedkommende allerede ligger i DSF, eller om personen skal registreres for første gang i registeret.

Det finnes mange situasjoner hvor det er nødvendig å gjennomføre en slik identifisering. Hvis en person skal jobbe midlertidig i Norge er det nødvendig for denne personen å få et D-nummer, slik at han eller hun blant annet kan motta skattekort. Personen sender da inn en søknad om å få opprettet et D-nummer, og denne saken blir tatt i mot av en ID-forvalter hos SKD. Basert på opplysninger som personer sender inn, som fødselsdato og navn, finner ID-forvalteren ut om vedkommende allerede er registrert i DSF, eller om personen må registreres.

2.3 NAV

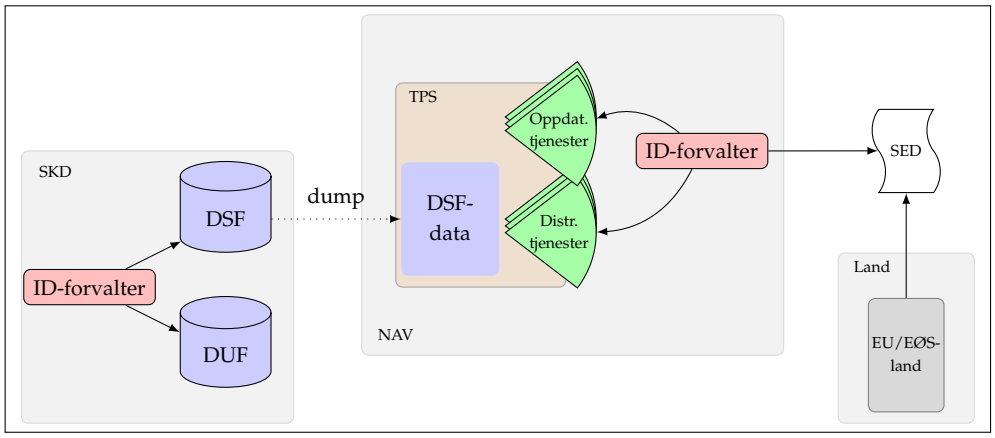
Arbeids- og velferdsforvaltningen (NAV) er en landsdekkende offentlig virksomhet som ble opprettet 1. juli 2006 ved sammenslåingen av Aetat og trygdeetaten. NAV er organisert med et direktorat i Oslo og egne NAV-kontor i alle fylker og kommuner i Norge [73, 71]. NAV har hele befolkningen som brukergruppe, og forvalter ordninger som dagpenger, arbeidsavklaringspenger, pensjon, barnetrygd og kontantstøtte [55]. NAV behandler en rekke saker hvor folkeregisterdata brukes i saksbehandlingen, og NAV er en stor bruker av folkeregisterdata.

Som nevnt er det SKD som er ansvarlig for identifiseringsprosessen av personer i Norge, og ansvarlig for å tildele D-nummer og fødselsnummer til personer som søker om dette. Men NAV har også en rekke egne ID-forvaltere, og disse brukes i forbindelse med såkalt lovvalg — prosessen med å bestemme medlemsland til en person. NAV har et samarbeid med EU- og EØS-land når det gjelder utveksling av personinformasjon, og hvis en saksbehandler skal behandle en sak som omhandler en person som bor i Norge som ikke har norsk statsborgerskap, er det tre faser en saksbehandler må gjennom.

1. Identifisere personen
2. Bestemme medlemsland — lovvalg
3. Behandle sak

2.4 Identifiseringsproblemet

I forbindelse med en slik saksbehandling er det særlig punkt *1) Identifisere personen* som er interessant og som vi vil analysere nærmere. Figur 2.2 viser en oversikt over identifiseringsprosessen hos henholdsvis SKD og NAV.



Figur 2.2: Oversikt over identifiseringsprosessen.

En ID-forvalter hos SKD anvender en rekke registre, i tillegg til DSF, for å stadfeste en persons identitet. Ett av disse er Datasystem for utlendings- og flyktningsaker (DUF), som er et register for alle asylsøkere og flyktninger i Norge. Et DUF-nummer er et 12-sifret identifikasjonsnummer som tildeles utlendinger som søker opphold i eller innreise til Norge. En ID-forvalter i SKD bruker data fra både DSF og DUF (og andre registre som ikke er med i figur 2.2) til å fastsette om en person eksisterer i et av registrene, og om personen må opprettes i DSF.

Mens ID-forvalterne i SKD jobber mot flere registre, jobber ID-forvalterne i NAV mot sitt hovedsystem Tjenestebasert Persondatasystem (TPS). TPS er bygd på en tjenesteorientert arkitektur og tilbyr en rekke oppdaterings- og distribusjonstjenester som ulike fagsystemer bruker for å hente ut og oppdatere data, og er laget for å sikre en enhetlig distribusjons- og oppdateringsløsning. NAV mottar hver natt en dump med et uttrekk av Folkeregisteret som blir lagt inn i TPS. I tillegg til denne dumpen inneholder TPS all annen informasjon som NAV lagrer, som blant annet informasjon om personers pensjon og arbeid. Det er først og fremst uttrekket av DSF som ID-forvaltningen på NAV jobber mot når de skal identifisere personer.

NAV sin identifiseringsprosess foregår ved at de mottar informasjon om personen de skal identifisere fra personens hjemland. Denne informasjonen består blant annet av navnet og fødselsdatoen til personen, og ofte mors og fars navn, samt et ID-nummer. Dette ID-nummeret kan enten være et nasjonalt ID-nummer som unikt identifiserer personen i det landet han eller hun kommer fra (som Norges fødselsnummer), men det kan også være et lokalt ID-nummer som kun blir brukt i den etats systemer som

NAV kommuniserer med. For utveksling av denne informasjonen brukes formatet Structured Electronic Data (SED). Dette er et standardformat for transaksjoner og er en del av en flyt som foregår mellom NAV og de ulike landene. SED-ene blir sendt frem og tilbake og lagres for hver gang de blir oppdatert.

Det finnes en rekke typer SED, men den vi skal jobbe med heter A-SED og er en serie med SED-er som brukes til kommunikasjon med andre EØS-land og Sveits i forbindelse med lovvalg. Denne serien består av ni forskjellig SED [54]:

A001 Forespørsel om unntak

A002 Svar på forespørsel om unntak

A003 Beslutning angående gjeldende lovgivning

A003A Beslutning angående gjeldende lovgivning — papirversjon

A004 Svar på beslutning om gjeldende lovgivning

A005 Forespørsel om ytterligere informasjon

A006 Svar på forespørsel om ytterligere informasjon

A007 Foreløpig beslutning angående gjeldende lovgivning

A008 Melding om relevant informasjon

Veiledning til A-SED [54] beskriver hvordan hver av disse SED-ene er en del av en flyt, og for A-serien eksisterer det fire flyter: FA001, FA002, FA003 og FA004. Den mest interessante for identifiseringsproblemet er flyt FA002 som benyttes ved beslutning om lovvalg og inneholder SED A003, A005, A006, A004, A007 og A003A. Rekkefølgen innenfor denne bestemte flyten må opprettholdes, men flere av SED-ene er valgfrie. A003 sendes av et EØS-land til NAV, og NAV bruker dataene i denne SED-en til å foreta en identifisering. Hvis NAV får foretatt en identifisering basert på denne informasjonen, sendes en A006 tilbake og flyten avsluttes. Flyten avsluttes også automatisk hvis NAV ikke foretar seg noe innen to måneder. Hvis NAV ønsker mer informasjon kan de sende en A005, hvor det spesifiseres hva det ønskes informasjon om, og en A005 besvares med en A006. En A007 benyttes ved uenighet om lovvalget. A003A er en papirversjon av A003 og skal ikke benyttes.

For å gjennomføre identifiseringen utfører saksbehandlerne hos NAV manuelle fonetiske søk mot TPS, for å lete etter treff på attributtene som NAV har mottatt fra de utenlandske etatene. Hvis saksbehandleren finner personen i TPS, regnes personen som identifisert og informasjonen som er mottatt knyttes opp mot den eksisterende personen i TPS. Hvis saksbehandleren ikke finner personen i TPS, sendes en «opprett d-nummer»-sak til SKD, men da sendes kun navnet og fødselsdatoen med. SKD mottar så denne informasjonen fra NAV, og gjennomgår igjen en identifiseringsprosess basert på sine datakilder. Hvis denne prosessen fører til at personen ikke blir identifisert, blir det lagt inn en ny oppføring i DSF

med denne personens data. NAV vil så få denne informasjonen inn i sitt system ved den nattlige dumpen med folkeregisterdata som de mottar.

Ved identifiseringsprosessen hos NAV er det to uønskede scenarioer som kan oppstå.

Én id brukes til å identifisere flere personer Dette forekommer hvis saksbehandleren finner en person i TPS, men denne personen er egentlig ikke korrekt person. Dette kan skje hvis det ligger en annen person i TPS med mange attributter som er identiske med den personen saksbehandleren prøver å identifisere. Dette fører dermed til at en person som SKD burde ha registrert i DSF, ikke blir registrert.

Én person blir identifisert med flere id-er Dette forekommer hvis saksbehandleren ikke klarer å finne personen i TPS, men personen eksisterer egentlig. Dette kan skje hvis de attributtene som benyttes i søket ikke er lagret for denne personen i TPS, men personen eksisterer i andre systemer som NAV ikke har tilgang til å søke i. Dette fører igjen til at NAV sender en «opprett d-nummer»-sak til SKD på en person som eksisterer fra før, og det er dermed en risiko for at en person blir lagt inn flere ganger i DSF.

Det er flere grunner til at de to scenarioene er uønsket. For det første så vil dette resultere i at DSF ikke stemmer med den virkelige verden. Det vil si at selv om man tror at en person er registrert i DSF, så er personen faktisk ikke det. Dette kan føre til at personer som ønsker å få behandlet en sak hos en offentlig etat får problemer med dette, fordi saksbehandlerne ikke klarer å finne personen i folkeregisterdataene. Det er også et problem hvis en person blir registrert to ganger, fordi personer enkelt kan utnytte dette ved for eksempel å motta dobbel stønad fra NAV, og å avgi stemme ved kommune- og fylkestingsvalg og stortingsvalg flere ganger. NAV har også avdekket organisert svindel hvor personer har operert med flere identiteter, blant annet hvor en mor har mottatt trygd for barn som ikke eksisterer [2, side 27].

2.5 Utfordringer med dagens løsning

Det er flere utfordringer og problemer knyttet til både dagens pris- og distribusjonsmodell av folkeregisterdata og hvordan dette påvirker identifiseringsproblemet.

Ifølge rapporten *Utveksling av grunndata på personinformasjonsområdet* [2] er det flere brukere av folkeregisterdata som ikke er fornøyd med den tilgangen de har i dag og ønsker en bedre tilgang. Rapporten sier at tilgangen til registeret er for dårlig, og at etater ikke oppdaterer sine opplysninger fordi oppslag og tilkobling til registeret er for dyrt. Dette underbygges av rapporten *Fra bruk til gjenbruk* [1] som blant annet sier at prisen som ligger til grunn for distribusjon av folkeregisterdata har vært så høy at det har hindret etater i å ta ut all den informasjon som de mener

de trenger for å utføre sine oppgaver. I tillegg er det flere kommuner som reagerer på at de pålegges å legge inn data gratis, men at de må betale for å hente dataene ut igjen.

I tillegg til prisproblematikken er det flere problemer knyttet til selve distribusjonsmodellen. Som nevnt inneholder dumpen med data som blir distribuert til NAV og distributører kun et uttrekk av DSF. I tillegg er det flere etater som skulle ønske at DSF inneholdt flere dataelementer og det medfører ofte dobbeltarbeid der etater selv må hente inn data der DSF ikke inneholder den nødvendige informasjonen [2, side 11]. I tillegg er det mange etater som har egne registre, såkalte skyggeregistre, som inneholder informasjonen som etatene trenger, og hvor DSF ikke er tilstrekkelig ajourført.

En konsekvens av at det eksisterer flere slike kopier av Folkeregisteret, er at systemene som forvalter de forskjellige kopiene ofte er bygd opp på forskjellige måter når det gjelder arkitektur, men også hvordan dataene er lagret og modellert. Siden systemene påvirkes av hvilke systemleverandører som leverer systemene, skaper dette problemer for semantisk interoperabilitet [80, side 31]. Interoperabilitet er evnen distribuerte systemer har til å utveksle tjenester og data, mens semantisk interoperabilitet sikrer at disse utvekslingene gir mening. Det vil si at forespørreren og tilbyderen har en felles forståelse for meningen til de forespurte tjenestene eller dataene [39]. Tett knyttet til dette er også behovet for metadata, og i denne sammenhengen anser vi metadata som opplysninger om hva et dataelement betyr, eller hvilket innhold det har [2, side 115]. For å få til økt samhandling mellom etater og systemer er det viktig få til en effektiv gjenbruk av opplysninger, og en viktig forutsetning for et slikt gjenbruk er at det er klart definert hva et dataelement betyr [2, side 41].

Det er også et problem at NAV og andre etater ikke får jobbet direkte mot den sentrale databasen med folkeregisterdata. Hvis for eksempel saksbehandlerne i NAV ønsker å oppdatere informasjonen til en person i DSF, er ikke dette mulig å gjøre direkte mot DSF hos SKD, men må gjøres via TPS og NAV sin «kopi» av DSF. Dette fører til at data legges inn i NAV sine systemer istedenfor i Folkeregisteret, og at NAV sitt datasett og Folkeregisteret ikke er synkronisert. I tillegg kan det ta tid før meldinger som er sendt inn til DSF når saksbehandlerne hos NAV. For eksempel kan det ta lang tid før en dødsmelding som er meldt inn til DSF blir distribuert ut til NAV [2, side 41].

Når det gjelder identifiseringsproblemet er det flere sider som er problematiske, og det er både en teknisk og ikke-teknisk side av problemet. Den ikke-tekniske siden går på at rollefordelingen ikke er klart definert mellom SKD og NAV. SKD er ansvarlig for identifiseringsprosessen, men NAV har også sin egen identifiseringsprosess, og det er ingen klar fordeling av ansvar og arbeidsoppgaver mellom disse to etatene. Denne uklare ansvar- og arbeidsfordelingen er en veldig viktig del av problemet, og for å få en komplett løsning av identifiseringsproblemet, er det nødvendig med en analyse av rollefordeling og deretter legge frem forslag til hvordan rolle- og

arbeidsfordeling bør være. En slik analyse er derimot utenfor omfanget av denne oppgaven, som vil fokusere på de teknologiske delene av problemet.

En av de teknologiske utfordringene er at identifiseringsprosessen er en manuell prosess, det vil si at saksbehandlere manuelt må søke etter personene i TPS basert på data som kommer i SED-filene. Dette tar mye tid, og det kan også være større sannsynlighet for feil, enn hvis det hadde vært en automatisert prosess. Videre er det et integrasjonsproblem, nemlig det å koble sammen data fra to forskjellige kilder. På den ene siden er det en datakilde, SED, som er på XML-format, mens det på den andre siden er en relasjonell database som ligger til grunn for TPS, hvor folkeregisterdataene ligger. Identifiseringsproblemet påvirkes også av de generelle utfordringene knyttet til distribusjonsmodellen for folkeregisterdataene. Dette gjelder først og fremst det å ikke ha tilgang til helt oppdaterte data, samt at DSF er mer innholdsrikt enn det NAV får tilgang til via TPS. Begge disse punktene kan føre til at identifiseringsprosessen feiler fordi det kan være informasjon som ikke har blitt overført til NAV sine systemer, eller det kan være informasjon som NAV ikke får tilgang til, men som ligger lagret i DSF.

Generelle utfordringer knyttet til dagens distribusjonsmodell, samt hvordan disse utfordringene konkret påvirker identifiseringsproblemet, kan oppsummeres med følgende punkter:

- Dumpen med folkeregisterdata som blir distribuert til NAV og distributører inneholder kun et uttrekk av DSF. Dette fører til dobbeltarbeid der etater selv må hente inn data som DSF ikke inneholder. Dette fører videre til opprettelse og forvalting av såkalte skyggeregistre, som inneholder informasjon som etatene trenger, men hvor dataene fra DSF ikke strekker til. Når det gjelder identifiseringsproblemet fører dette til utfordringer for ID-forvaltningen i NAV, fordi de har færre dataelementer å basere identifiseringen på enn ID-forvaltningen hos SKD.
- Saksbehandlere i blant annet NAV får ikke jobbet direkte mot den sentrale databasen med folkeregisterdata, og det kan ta tid før ulike type meldinger når NAV og distributørene. Dette fører igjen til at beslutninger kan tas på feil datagrunnlag.
- Det er store utfordringer knyttet til semantisk interoperabilitet og dataintegrasjon. Dette kommer konkret til uttrykk i identifiseringsproblemet, hvor to heterogene kilder, DSF og SED, skal kombineres.
- Det er et behov for å kunne beskrive metadata på en måte som gjør det mulig å gjenbruke denne informasjonen på tvers av systemer og etater. Det er også viktig at disse metadataene beskriver dataelementer på en måte som utvetydig definerer hva de ulike dataelementene betyr.
- Identifiseringsprosessen er en manuell prosess, det vil si at saksbehandlere manuelt må søke etter personene i TPS basert på data som

kommer i SED-filene. Dette tar mye tid, og det kan også være større sannsynlighet for feil, enn hvis det hadde vært en automatisert prosess.

2.6 Overordnet beskrivelse av ny løsning

Som vi har vist er det flere rapporter som har pekt på problemer og utfordringer med både pris- og distribusjonsmodell for dagens folkeregister. Blant annet på bakgrunn av dette har programmet *Modernisering av Folkeregisteret* blitt opprettet, som er ansvarlig for utviklingen av et nytt folkeregister. Dette er et arbeid som har som mål å være ferdig i 2015, og programmet bruker blant annet rapporten om utveksling av grunndata [2] som grunnlag.

Målet for denne oppgaven er ikke å lage et nytt Folkeregister, eller argumentere for en ny prismodell, men å se på hvordan en ny distribusjonsmodell for folkeregisterdata kan bidra til økt samhandling og en bedre tilgang til oppdaterte folkeregisterdata. Oppgaven presenterer en ny distribusjonsmodell for Folkeregisteret som bruker semantiske teknologier. Semantiske teknologier er et sett med teknologier som blant annet egner seg godt til å benyttes i systemer hvor dataintegrasjon er en sentral utfordring, og hvor det er behov for å spørre etter data fra en rekke heterogene datakilder. For å vise hvordan data fra flere kilder kan kombineres ved hjelp av semantiske teknologier, viser oppgaven hvordan DSF- og SED-datene kan distribueres på en måte som gjør det mulig å integrere disse heterogene datakildene på en enkel og elegant måte. Den nye løsningen som vi presenterer vil fokusere på tre hovedpunkter: distribusjon, spørringer og modell for dataintegrasjon.

2.6.1 Distribusjon

For å bedre distribusjonen av folkeregisterdata er det nødvendig med en ny distribusjonsmodell for Folkeregisteret som er fleksibel og gjør det mulig for flere aktører å hente ut oppdaterte og korrekte data. I tillegg bør distribusjonsmodellen muliggjøre integrering av folkeregisterdata med andre heterogene kilder, som SED i identifiseringsproblemet. Distribusjonsmodellen bør være generell nok til at flere aktører kan benytte seg av den samme distribusjonskanalen, men samtidig kunne tilpasses til de forskjellige aktørers behov. Disse kravene underbygges av rapporten om utveksling av grunndata, hvor det stilles en rekke krav til arkitekturen til et nytt Folkeregister. Her heter det blant annet at *«Arkitekturen må ivareta tilgang til oppslag i Folkeregisteret, og mulighet til å foreta endringer i Folkeregisteret [...] Arkitekturen må tillate integrasjon [...] Arkitekturen må legge til rette for at det er enklest mulig å gjøre endringer i innhold, funksjonalitet og integrasjoner mot andre løsninger».*

Semantiske teknologier kan bidra til å lage en slik fleksibel modell ved å representere folkeregisterdataene på dataformatet Resource Description Framework (RDF). RDF blir forklart i detalj i seksjon 3.2.1 på side 20, men kort fortalt er RDF et språk for å beskrive strukturert informasjon, og fungerer som grunnsteinen innenfor semantiske teknologier. Ved å konvertere folkeregisterdataene til RDF kan det lages en distribusjonsløsning som er fleksibel, og som gjør det mulig for mange aktører å hente ut oppdaterte data, samtidig som det er mulig å tilpasse dataene for ulike behov. Videre vil det kunne skapes et rammeverk som gjør det enkelt å kombinere heterogene datakilder som også er distribuert som RDF.

2.6.2 Spørringer

For å kunne lage en fleksibel og god distribusjonsmodell er det nødvendig med et spørregrensesnitt som fungerer på tvers av applikasjoner, hvor vilkårlige spørringer kan stilles mot dataene. Siden distribusjonsmodellen skal benyttes av flere ulike aktører, er det nødvendig at spørregrensesnittet ikke stiller krav til et bestemt rammeverk eller programmeringsspråk, men benytter seg av en mekanisme for datakommunikasjon som kan benyttes av en rekke rammeverk og programmeringsspråk.

Et ønske som de ansatte i NAV har, er å kunne stille ulike kontrollspørringer mot folkeregisterdataene i forbindelse med identifiseringsprosessen, og det er derfor viktig at det er mulig å stille vilkårlige spørringer mot dataene uten at disse spørringene må være konfigurert på forhånd. For å kunne hente data fra både DSF og SED er det også viktig at spørregrensesnittet gjør det mulig å stille spørringer på tvers av disse heterogene datakildene.

Semantiske teknologier muliggjør distribusjon av dataene på en fleksibel måte, med et spørregrensesnitt som fungerer på tvers av applikasjoner, og som gjør det mulig å stille vilkårlige spørringer mot dataene. Spørrespråket som benyttes heter SPARQL og beskrives i detalj i seksjon 3.2.2 på side 26. SPARQL benytter seg av HTTP-protokollen for dataoverføring og tilbyr derfor et generisk grensesnitt som ikke stiller spesifikke krav til programvare eller rammeverk. Dette i motsetning til mer tradisjonelle grensesnitt som ofte er mindre fleksible, og hvor applikasjonen som ønsker å konsumere dataene må benytte seg av spesifikke språk og rammeverk.

2.6.3 Modell for dataintegrasjon

Ved hjelp av RDF og SPARQL kan en bedre distribusjonsmodell for Folkeregisteret tilbys, men det vil fortsatt eksistere utfordringer når det gjelder integrasjon av DSF med andre datakilder. Dataintegrasjon er et velkjent problem, og denne oppgaven definerer dataintegrasjon som problemet med å kombinere data fra forskjellige kilder og gi brukeren et enhetlig bilde av disse dataene [49]. Det eksisterer en rekke metoder for dataintegrasjon, men som regel baserer løsningen seg på et globalt skjema

som modellerer sammenhengen mellom de lokale kildene. Videre trengs en metode for å omformulere spørringer gitt mot det globale skjemaet, til spørringer mot de forskjellige lokale kildene.

Et problem med dataintegrasjon er at begreper kan brukes forskjellig i forskjellig kilder. For eksempel kan begrepet lønn bety årslønn i ett system, mens det i et annet system kan bety månedslønn. For å overføre data mellom disse systemene har den konvensjonelle løsningen vært å bruke dedikert programvare tilpasset hver enkelt overføring, og forholdet mellom data fra henholdsvis sender og mottaker har vært «hard-kodet» inn i programvaren [65]. Dette fører til liten mulighet for gjenbruk, og overføringen burde bli representert på en mer generisk måte. En naturlig tanke er å lage en felles modell hvor det er enighet om begreper på tvers av etater, og at alle systemer bruker denne felles modellen. Problemet med dette er at det er vanskelig å komme til en slik enighet når det gjelder alle begreper. Et alternativ er derfor å la de forskjellige etatene ha forskjellige modeller, men at det brukes et modelleringsspråk som gjør det mulig å definere relasjoner mellom de ulike begrepene.

Semantiske teknologier tilbyr modelleringsspråket Web Ontology Language (OWL) som blir presentert i detalj i seksjon 3.2.3 på side 33, og OWL gjør det mulig å lage en modell over datasettet som eksplisitt og matematisk definerer begreper og relasjonene mellom dem. Til grunn for en slik modell ligger det en formell semantikk som gjør det mulig å utføre resonnering over datasettene. Resonnering er prosessen med å utlede implisitt informasjon som ikke er eksplisitt lagret i datasettet, i tillegg til å sjekke om en modell er konsistent. Vi vil vise hvordan OWL-modeller kan utvikles basert på dokumentasjon fra SKD og NAV, og hvordan disse kan brukes som spørregrensesnitt over dataene. I tillegg vil vi vise hvordan modellene kan brukes til å automatisk stadfeste om to personer er like basert på DSF- og SED-dataene, og hvordan dette vil være med på å løse en av utfordringene i identifiseringsproblemet.

2.7 Oppsummering

Det er flere utfordringer knyttet til dagens distribusjon av folkeregisterdata, og dette kommer blant annet konkret til uttrykk ved problemer knyttet til identifiseringsproblemet. Problemer knyttet til mangel på oppdaterte og innholdsrike data og tid som brukes til journalføring av «skyggeregister», er eksempler på konsekvenser av dagens distribusjonsmodell. I tillegg kommer mer alvorlige konsekvenser som kan føre til at identifiseringen av personer feiler og personer kan registreres dobbelt opp i Folkeregisteret. Det har vært flere eksempler på personer som har utnyttet denne situasjonen ved for eksempel å skaffe seg dobbel stønad fra NAV.

En ny distribusjonsløsning må kunne tilby en fleksibel distribusjonsmodell som gjør det mulig å stille vilkårlige spørringer mot dataene, og bidrar til enklere integrasjon av heterogene datakilder. En løsning basert på

semantiske teknologier vil kunne gi en fleksibel løsning som muliggjør enklere integrering av heterogene datakilder, et fleksibelt spørregrensesnitt og mulighet for å utvikle en modell som eksplisitt og matematisk definerer begreper og relasjonene mellom disse.

Kapittel 3

Semantiske teknologier

Dette kapittelet gir en generell introduksjon til semantiske teknologier, og går gjennom de viktigste begrepene innenfor dette feltet. Kapittelet har et spesielt fokus på de delene av semantiske teknologier som benyttes av demonstratoren som presenteres i kapittel 4 og 5.

Kapittelet er delt opp i tre hoveddeler. Seksjon 3.1 forklarer kort historien bak begrepet semantiske teknologier, og videre presenteres det sentrale begreper som viser hvordan semantiske teknologier skiller seg fra tradisjonell databaseteknologi. Deretter presenteres de viktigste språkene innenfor semantiske teknologier som dataformat, spørrespråk og modelleringsspråk i seksjon 3.2. Seksjon 3.3 presenterer ulike teknologier som gjør at språkene kan anvendes til å lage applikasjoner og tjenester.

3.1 Introduksjon og grunnprinsipper

Kravene til en ny distribusjonsmodell for Folkeregisteret er knyttet til tre hovedpunkter: distribusjon, spørringer og modell for dataintegrasjon. Semantiske teknologier gjør det mulig å lage en løsning som oppfyller alle disse tre kravene, ved å tilby et dataformat, spørrespråk, modelleringsspråk og teknologier som kombinert vil gi en fleksibel og kraftig distribusjonsmodell.

Semantiske teknologier er et sett med standarder og teknologier som egentlig er blitt utviklet for å utvikle en såkalt semantisk web. Dette begrepet ble presentert av Tim Berners-Lee, oppfinneren av World Wide Web (heretter kalt web), i 2000 i boken *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web* [16]. Essensen i denne visjonen var at i fremtidens web skulle maskinene være i stand til å tolke og forstå meningen (semantikken) med dataene, og basert på dette kunne ta egne beslutninger. Dette i motsetning til den tradisjonelle weben som har som mål å formatere og presentere dataene, i hovedsak tekstdokumenter, for mennesker. Den semantiske weben har vært utsatt for mye «hype», og

har ikke fått den responsen eller suksessen som mange spådde. Men ut i fra begrepet semantisk web har det blitt utviklet en rekke standarder, verktøy og teknologier som er kjent som semantiske teknologier, og disse teknologiene er også godt egnet til å brukes i lukkede applikasjoner.

Vi argumenterer for at identifiseringsproblemet, beskrevet i kapittel 2, er godt egnet til å løses ved hjelp av semantiske teknologier, og viser hvordan folkeregisterdataene kan distribueres bedre ved hjelp av disse teknologiene. En av hovedutfordringene med identifiseringsproblemet er at data fra flere kilder kombineres, og at data fra disse kildene kan være overlappende. Det vil si at samme person kan eksistere i både Folkeregisteret og i et SED-dokument, men at personen er identifisert med ulike identifikatorer i de to datasettene. I tradisjonelle teknologier, som relasjonelle databaser, er det ofte utfordringer knyttet til det å koble data fra flere kilder sammen hvor samme entitet er identifisert med ulike identifikatorer. Med semantisk teknologi er dette derimot enklere fordi det er mulig å eksplisitt si om to entiteter er like og ulike, og at dette ikke er gitt ut i fra identifikatoren til entiteten.

3.1.1 Åpen/Lukket verden semantikk

Åpen-verden semantikk eller *Open World Assumption* (OWA) er et meget viktig prinsipp innenfor semantiske teknologier, og et av de viktigste punktene der semantiske teknologier skiller seg fra tradisjonelle teknologier som relasjonelle databaser. OWA sier at hvis vi vet at noe ikke er sant i henhold til en gitt modell, er det ikke nødvendigvis slik at utsagnet er usant i henhold til samme modell. Dette skiller seg fra en *Lukket-verden semantikk* eller *Closed World Assumption* (CWA), som er antagelsen om at hvis et utsagn ikke er sant i henhold til en gitt modell, er det usant.

Som et eksempel kan vi se på en relasjonell database som kun inneholder utsagnet *Kari bor i Norge*. Hvis vi nå ønsker å sjekke om utsagnet *Ola bor i Norge* er sant i henhold til databasen, vil vi få *nei* som svar. Dette skyldes at relasjonelle databaser tar utgangspunkt i CWA som svarer til en antagelse om at vi har en fullstendig og komplett kunnskap om det som modelleres: Vet vi ikke at et utsagn er sant, kan vi si at det er usant. Siden vi ikke har noen informasjon om at Ola bor i Norge, kan vi si at det motsatte er sant, nemlig at Ola ikke bor i Norge.

Hvis en database som tok utgangspunkt i OWA inneholdt det samme utsagnet (*Kari bor i Norge*), hadde vi fått svaret *vet ikke* istedenfor *nei* hvis vi sjekket om utsagnet *Ola bor i Norge* er sant i henhold til databasen. Dette skyldes at OWA svarer til en antagelse om at vi ikke nødvendigvis har komplett kunnskap om det som modelleres, og hvis vi ikke vet at et utsagn er sant, er det nødvendigvis ikke usant, det er rett og slett udefinert. Vi har ingen informasjon om at Ola bor i Norge i vår modell, men det kan være at vårt datasett ikke er komplett, og vi har dermed ikke grunnlag for å kunne konkludere med om Ola bor i Norge eller ikke.

En av grunnene til at semantiske teknologier antar OWA, er at teknologiene ble utviklet med tanke på å utvide den eksisterende weben, og bygge videre på dens prinsipper. Et viktig prinsipp med weben er at *«alle kan si noe om alt»*, det vil si at det ikke er en autorativ enhet som kontrollerer det som legges ut på weben, men alle kan legge ut informasjon om alle mulige temaer. Dette fører til at det er mye informasjon på weben som kan inneholde feil, eller være mangelfull. På grunn av dette ville det vært feil å anta at datasettet (weben) som det jobbes med er komplett. Hvis spørsmålet om Ola bor i Norge stilles, og vi ikke får et svar tilbake som sier at Ola bor i Norge, kan vi likevel ikke si at Ola ikke bor i Norge. Dette er fordi det kan være datakilder som vi ikke har tilgang til som inneholder denne informasjonen, eller det kan være noen som enda ikke har lagt ut informasjon om at Ola bor i Norge (men gjør det kort tid etterpå).

Som nevnt antar relasjonelle databaser CWA, og i mange systemer er dette en riktig semantikk å bruke. For eksempel i et system som inneholder alle ansatte i et firma, er det rimelig å anta at datasettet er komplett, og hvis en person ikke ligger i dette systemet, er personen ikke ansatt. Men det finnes også systemer hvor det kan diskuteres om CWA er den riktige antagelsen. I et system som inneholder pasientinformasjon om pasientens allergier, og systemet ikke har informasjon om at en pasient har en gitt allergi, er det ikke nødvendigvis mulig å konkludere med at pasienten ikke har denne allergien. Et slikt datasett vil rett og slett aldri bli helt komplett fordi det kan være allergier hos pasienten som ikke er dokumentert og lagret i systemet.

Å anta en åpen-verden semantikk vil være relevant for identifiseringsproblemet, fordi det der er ønskelig å kunne identifisere en person, det vil si dra slutninger basert på potensiell ufullstendig informasjon. Det er ikke sikkert at en person ikke eksisterer selv om personen ikke eksisterer i ett registre, men det kan rett og slett være at personen ligger i et register som en saksbehandler ikke har tilgang til, eller at personen enda ikke er blitt registrert.

3.1.2 Antagelsen om unike/ikke-unike navn

Antagelsen om unike navn eller *Unique Name Assumption* (UNA) er antagelsen om at to entiteter med forskjellig navn er forskjellige. Dette i motsetning til antagelsen om ikke-unike navn eller *Non-Unique Name Assumption* (NUNA), hvor to entiteter med forskjellig navn ikke nødvendigvis er forskjellig entiteter. Relasjonelle databaser antar UNA, hvor to rader med forskjellig id alltid vil representere ulike entiteter. Innenfor semantiske teknologier derimot, som antar NUNA, vil to entiteter med ulike navn ikke nødvendigvis være forskjellige. Som nevnt ble semantiske teknologier utviklet med tanke på å utvide den eksisterende weben, og på weben er det ikke mulig med UNA. Det eksisterer ikke et id-regimé på weben, og siden *«alle kan si noe om alt»*, vil en antagelse om unike identifikatorer ikke kunne fungere.

Det at relasjonelle databaser antar UNA, er en av grunnene til at det oppstår problemer når data fra to relasjonelle databaser skal integreres. Vi tenker oss to databaser, A og B, hvor A inneholder informasjon om personer og deres adresser, mens B inneholder personer og deres familierelasjoner som ektefeller og barn. Noen personer vil være representert i begge databasene, mens noen personer kun vil være representert i én. Hver person vil være unikt identifisert av en nøkkel i begge databasene, men den vil kun være unik innenfor den aktuelle databasen. Det vil si at en person, Ola Nordmann, som opptrer både i database A og B, (høyst sannsynlig) vil ha forskjellig nøkkel i de to databasene, mens to forskjellige personer kan ha samme nøkkel i de to databasene. Hvis vi vil slå sammen database A med database B med standard teknologi, tvinger disse teknologienes krav om UNA oss til å lage en ny database C, siden vi ellers kunne risikert at Ola Nordmann ville blitt representert som to personer med to ulike nøkler. Men dette krever i sin tur at vi må foreta en transformasjon av data som ofte er omfattende og også blir en feilkilde i seg selv. Spesielt må vi sørge for at identiteter i A og B mappes over til unike identiteter i C, slik at to id-er i A og B får samme id i C når de betegner samme entitet, og ellers får forskjellig id i C.

Semantiske teknologier tilbyr derimot separate identitetsutsagn som gjør det mulig å definere at to entiteter som er identifisert med ulike identifikatorer omtaler samme entitet. Et viktig poeng er at innenfor semantiske teknologier er to individer med ulikt navn hverken like eller ulike før dette er eksplisitt definert. I begge antagelser gjelder (selvfølgelig) at to individer med samme navn er samme individ.

For identifiseringsproblemet er dette relevant fordi vi der har flere kilder hvor den samme personen potensielt kan være identifisert med ulike identifikatorer, og det vil være en stor fordel om det er mulig å si at to entiteter med forskjellige identifikator faktisk representerer samme entitet. Semantiske teknologier har videre et «system» som gjør at id-er får et globalt omfang, slik at det ikke vil være slik at to ulike personer er identifisert med lik identifikator.

3.2 Språk

3.2.1 RDF

Det første kravet til den nye løsningen er distribusjon, og tett knyttet til distribusjon er valg av representasjonsspråk. World Wide Web Consortium (W3C) har utviklet og standardisert en datamodell som brukes som dataformat innenfor semantiske teknologier som heter Resource Description Framework (RDF). RDF [50] er et formelt språk for å beskrive strukturert informasjon [41, side 19], og har flere egenskaper som gjør det til en attraktiv datamodell for å koble sammen data fra flere kilder, også hvor dataenes skjema er forskjellige [61].

Ved å representere og distribuere data som RDF blir det mulig for mange ulike applikasjoner å konsumere og prosessere de samme dataene, og RDF egner seg også godt som dataformat hvor flere kilder skal integreres. Et problem med dataintegrasjon av relasjonelle databaser, er at databaseskjemaer er statiske og vanskelig å utvide når de først er utviklet, og en endring i databaseskjemaet fører til store forandringer i både datasettet og applikasjoner som bruker dataene [9]. I motsetning til relasjonelle databaser som representerer alt som tabeller med relasjoner mellom disse tabellene, representerer RDF alt som såkalte tripler. En trippel består av et subjekt, et predikat og et objekt, og brukes til å si at subjektet og objektet er relatert, og måten de er relatert på beskrives av predikatet. Fellesbetegnelsen på subjekter, predikater og objekter er ressurser.

RDF-dokumenter består av et sett med tripler, og en stor fordel med RDF er at det å legge til eller fjerne en slik trippel ikke vil påvirke de eksisterende dataene eller ikke minst applikasjonene som bruker dataene. En annen fordel er at alt fra tabulære data og relasjonelle databaser til trestrukturer kan oversettes til slike tripler, og gjør det derfor mulig å eksponere en rekke forskjellige datakilder som RDF. Eksempler på noen tripler er:

```
Ola  erEn Person
Kari erEn Person
Ola harEktefelle Kari
Ola harNavn "Ola Nordmann"
```

RDF startet som et språk for å beskrive metadata om ressurser på weben, som tittelen på et HTML-dokument, forfatteren til dokumentet og når det ble opprettet [50]. RDF har videre utviklet seg til et språk for å beskrive konsepter og «ting» som kan bli identifisert på weben, selv om disse «tingene» ikke nødvendigvis eksisterer på weben. For å beskrive data i RDF utvides lenkestrukturen vi har på den tradisjonelle weben, ved å bruke Uniform Resource Identifiers (URI-er) til å navngi og identifisere relasjonen mellom to ting, samt de to tingene på enden av relasjonen. URI-er er en generalisering av Uniform Resource Locators (URL-er) som er en URI som i tillegg til å identifisere en ressurs, også angir en måte å lokalisere ressursen på ved å angi en aksessmekanisme [11]. URL-er blir ofte brukt som et begrep for en webadresse som peker til et dokument på nettet hvor HTTP-protokollen brukes til å lokalisere ressursen.

En av hovedgrunnene til å bruke URI-er er at det gir en identifikator som har et globalt omfang, i motsetning til relasjonelle databaser som har lokale identifikatorer som kun er unike innenfor databasen. Dette hjelper i sammenkoblingen av forskjellig datakilder, fordi det er mindre sannsynlig at to forskjellige entiteter har lik identifikator. Siden URI-er er en generalisering av URL-er er det også mulig å legge informasjon om en ressurs på URL-en som entiteten har som identifikator. Dette gjør det mulig å utforske RDF-data ved hjelp av vanlige mekanismer som brukes for å navigere på weben.

Et viktig poeng er at URI-er kun er en tekststreng på et spesielt format, og

det er ikke mulig å sikre at en URI vil være unik. Det er ingenting i veien for at flere personer bruker samme URI for å identifisere to helt ulike entiteter. En konvensjon innenfor semantiske teknologier er derfor å bruke domener som man selv eier når ressurser skal beskrives. Dette gjør at det er mindre sannsynlig at én URI brukes til å identifisere to ulike entiteter.

I tillegg til å bruke URI-er til datarepresentasjon kan objekter representeres som såkalte literaler. En literal er en konstant verdi, representert av tekststrenger, og kan kun bli brukt som objekt (ikke subjekt eller predikat) [50]. For å kunne vite hvordan en literal skal tolkes, er det mulig å angi en datatype som beskriver om literalen skal tolkes som et tall, en streng eller en dato. En slik literal kalles for en typet literal. Hvis en literal ikke er typet, kan literalen angis en språkkode som sier hvilket språk literalen er på. Merk at en literal enten kan ha en datatype, eller en språkkode, ikke begge deler.

Hvis vi tar utgangspunkt i triplene ovenfor, kan vi representere hver enkelt ressurs som en URI[1] eller en literal. Vi får da følgende tripler:

```
http://sws.ifi.uio.no/dsf/henriwi/Ola
  http://www.w3.org/1999/02/22-rdf-syntax-ns#type
    http://sws.ifi.uio.no/dsf/henriwi/Person

http://sws.ifi.uio.no/dsf/henriwi/Kari
  http://www.w3.org/1999/02/22-rdf-syntax-ns#type
    http://sws.ifi.uio.no/dsf/henriwi/Person

http://sws.ifi.uio.no/dsf/henriwi/Ola
  http://sws.ifi.uio.no/dsf/harEktefelle
    http://sws.ifi.uio.no/dsf/henriwi/Kari

http://sws.ifi.uio.no/dsf/henriwi/Ola
  http://sws.ifi.uio.no/dsf/harNavn
    "Ola Nordmann"
```

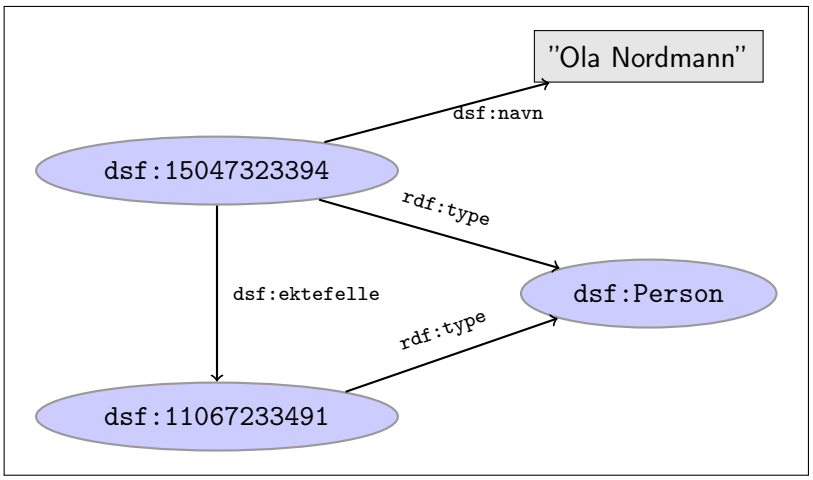
Vi ser at URI-er fort kan bli lange og vanskelige å lese. Det er derfor vanlig å benytte seg av prefikser som gjør at lengden på triplene reduseres og lesbarheten øker. Vi sier at prefikset `dsf:` skal representere URI-en `http://sws.ifi.uio.no/dsf/henriwi/`, og prefikset `rdf:` skal representere URI-en `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. Ved å anvende disse prefiksene blir triplene seende slik ut:

```
dsf:Ola rdf:type dsf:Person
dsf:Kari rdf:type dsf:Person
dsf:Ola dsf:ektefelle dsf:Kari
dsf:Ola dsf:navn "Ola Nordmann"
```

RDF kan visualiseres som en rettet graf, det vil si et sett av noder som er koblet sammen ved hjelp av rettede kanter [41, side 20]. I en

[1] Grunnen til at URI-en `http://sws.ifi.uio.no/dsf/henriwi/` er valgt, er fordi URI-en har et domene som vi selv har kontroll over.

slik graf vil nodene beskrive subjektene og objektene, mens de rettede kantene beskriver predikatene. Figur 3.1 viser hvordan triplene ovenfor kan visualiseres som en rettet graf. Her representerer de ovale nodene subjekter og objekter som er identifisert av en URI, mens den firkantede noden representerer en literal. I tillegg har vi erstattet identifikatorene til Ola og Kari med et fødselsnummer. Dette er fordi det bør tilstrebes å bruke unike identifikatorer, og det er stor sannsynlighet for at flere personer har samme navn, og derfor egner navnet seg dårlig som identifikator.



Figur 3.1: En RDF-graf.

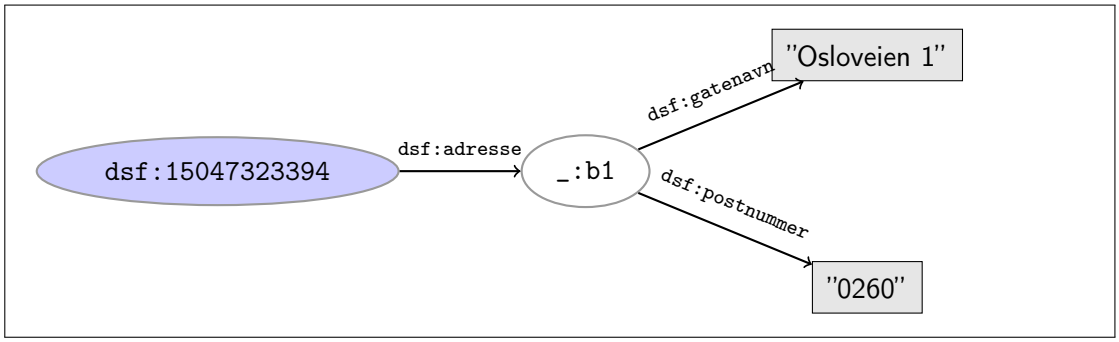
En RDF-graf er en mengde med tripler, og dette gjør at vanlige mengdeoperasjoner kan utføres. Hvis vi for eksempel ønsker å slå sammen to RDF-grafer, er det bare å ta unionen av disse to grafene, så vil vi få en ny RDF-graf. Dette skiller seg fra for eksempel tabulære data hvor tabelldimensjonene må matche, eller trestrukturer hvor det stilles mye strengere krav til strukturen. Blant annet kan en node kun ha én forelder, samt at treet må ha én og bare én rotnode. Dette er også grunnen til at det er enkelt å utvide en RDF-graf, fordi det bare er å legge til flere tripler til den eksisterende mengden.

Blanke noder

I tillegg til å kunne representere ressurser som URI-er og literaler, kan subjekter og objekter også bli representert som såkalte blanke noder. Slike noder har ingen URI tilknyttet seg, og brukes når vi ønsker å hevde at det finnes en ressurs, men vi vet ikke identiteten til ressursen eller det er ikke hensiktsmessig eller mulig å identifisere ressursen.

En blank node kan enten være navngitt med et lokalt navn som er unikt innenfor det aktuelle RDF-dokumentet, eller den kan være anonym. En blank node kan gis et lokalt navn hvis den skal opptre i flere tripler, men hvis det ikke er nødvendig å kunne referere til den blanke noden i andre tripler, kan den blanke noden være anonym.

Figur 3.2 viser hvordan en blank node kan brukes til å gruppere flere tripler sammen for å representere en adresse. Det er lite hensiktsmessig å bruke en egen identifikator for noden som representerer en adresse, fordi det er vanskelig å finne en god identifikator for en adresse (annet enn `adresse1` og lignende), og det er derfor en god idé å bruke en blank node til dette. Figuren viser også at den blanke noden har fått navnet `_:b1`.



Figur 3.2: En RDF-graf med bruk av blanke noder.

RDF-serialiseringer

RDF er en datamodell og kan visualiseres og representeres på flere måter. RDF-grafer egner seg godt for visualisering av små grafer og er lette å lese og tolke for mennesker. Men for at maskiner skal kunne lagre, prosessere og tolke RDF, må datamodellen til RDF serialiseres til en konkret syntaks. Den offisielle anbefalingen fra W3C er serialiseringen RDF/XML [6], og figur 3.3 viser hvordan grafen i figur 3.1 på forrige side kan bli serialisert til RDF/XML-format.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dsf="http://sws.ifi.uio.no/dsf/henriwi/">
  <dsf:Person
    rdf:about="http://sws.ifi.uio.no/dsf/henriwi/15047323394">
    <dsf:ektefelle>
      <dsf:Person
        rdf:about="http://sws.ifi.uio.no/dsf/henriwi/11067233491" />
    </dsf:ektefelle>
    <dsf:navn>Ola Nordmann</dsf:navn>
  </dsf:Person>
</rdf:RDF>
```

Figur 3.3: RDF/XML-serialisering.

Fordelen med RDF/XML-serialisering er at den har god verktøystøtte siden verktøy som støtter XML kan brukes. Ulempen er at siden XML-dokumenter krever en hierarkisk fremstilling av innholdet, må triplene være på en hierarkisk struktur, og det er nødvendig med en omskriving

fra grafstrukturen som ligger til grunn for datamodellen til RDF. I tillegg er serialiseringen ganske vanskelig å lese for mennesker.

En serialisering som er mye brukt, blant annet fordi den er lett å lese for mennesker, er Turtle [7]. Turtle har ingen hierarkisk struktur som RDF/XML-serialiseringen, og en RDF-graf kan mer eller mindre oversettes direkte til et Turtle-dokument. Figur 3.4 viser hvordan grafen i figur 3.1 på side 23 kan bli serialisert til Turtle. Det er vanlig å bruke navnerom eller prefikser i Turtle, og figuren viser hvordan prefikser defineres ved å bruke nøkkelordet `@prefix`.

Det er ofte tilfellet at flere tripler dreier seg om samme subjekt, og det er derfor mulig å bruke et semikolon slik at det ikke er nødvendig å skrive subjektet på nytt. I figur 3.4 vises dette ved at det er tre tripler som omhandler subjektet `dsf:15047323394`, men subjektet skrives kun én gang. Videre vises det at et punktum brukes for å angi at utsagnene knyttet til det aktuelle subjektet er ferdig. Literaler omgis av fnutter, og hvis det er en datatype knyttet til en literal, angis dette på følgende måte: `"42"^^xsd:integer`. Hvis det er en språkkode knyttet til en literal, angis dette på følgende måte: `"Oslo"@no`, hvor «no» betyr Norsk.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dsf: <http://sws.ifi.uio.no/dsf/henriwi/> .

dsf:11067233491 rdf:type dsf:Person .

dsf:15047323394 rdf:type dsf:Person ;
                dsf:ektefelle dsf:11067233491 ;
                dsf:navn "Ola Nordmann" .
```

Figur 3.4: Turtle-serialisering.

Blanke noder uttrykkes i Turtle ved å bruke en «_» istedenfor et prefiks:

```
dsf:15047323394 dsf:adresse _:b1 .
_:b1  dsf:gatenavn "Osloveien 1" ;
      dsf:postnummer "0260" .
```

For å lage en anonym blank node, brukes `[ ... ]`:

```
dsf:15047323394 dsf:adresse [
  dsf:gatenavn "Osloveien 1" ;
  dsf:postnummer "0260" .
] .
```

Alle eksemplene i resten av oppgaven bruker Turtle.

Det finnes en rekke andre serialiseringer som denne oppgaven ikke kommer til å gå i detalj på, men noen av disse er N3 [12], N-Triples [32] og TriG [18].

Navngitte grafer

Som nevnt er det ønskelig at en ny distribusjonsmodell gjør det mulig for flere aktører å benytte seg av den samme løsningen, samtidig som den kan tilpasses de forskjellige aktørers behov. En måte å bidra til dette på er å kunne dele opp RDF-dataene i flere bolker, for eksempel for å øke ytelsen, for å sette tilgangskontroll og for å separere de forskjellige aktørers data. Det er mulig å utvide hver enkelt RDF-trippel med et fjerde element, slik at det dannes en såkalt 4-tuppel. Det fjerde elementet er en URI som tilordner kontekst til hver enkelt trippel, og alle 4-tupplene som har det samme fjerde elementet er knyttet til samme graf. Et RDF-datasett kan dermed bestå av en mengde navngitte grafer, hvor hver graf er identifisert av en URI.

Dette gjør det mulig å dele opp dataene i forskjellige bolker, for eksempel en bolk som tilhører NAV, og en annen bolk som tilhører SKD. Figur 3.5 viser hvordan et RDF-datasett bestående av to navngitte grafer kan serialiseres til TriG. TriG er en serialisering som baserer seg på Turtle, men utvidet med mulighet for å definere navngitte grafer. Figuren viser hvordan vi kan ha én graf for NAV og én for SKD.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dsf: <http://sws.ifi.uio.no/dsf/henriwi/> .

dsf:NAV = { dsf:11067233491 rdf:type dsf:Person ;
                            dsf:navn "Kari Nordmann" . } .

dsf:SKD = { dsf:15047323394 rdf:type dsf:Person ;
                            dsf:navn "Ola Nordmann" . } .
```

Figur 3.5: Navngitte grafer serialisert til TriG.

3.2.2 SPARQL

Det andre kravet til den nye løsningen skissert i kapittel 2 er å ha et spørregrensesnitt som fungerer på tvers av applikasjoner og som gjør det mulig å stille vilkårlige spørringer mot dataene. SPARQL Protocol And RDF Query Language (SPARQL) [60] er en standard for å skrive spørringer mot RDF-data og for å representere resultatet [41, side 262]. Språket kan minne om SQL, men skiller seg allikevel på en del punkter siden de opererer på veldig forskjellige datastrukturer. SPARQL vil bidra med et fleksibelt og kraftig spørrespråk, og dermed gjøre det mulig å stille vilkårlige spørringer på tvers av applikasjoner og datasett.

Hovedideen til SPARQL er å skrive spørringer for å matche såkalte grafmønstre (eng: graph pattern), og spørringene skrives med en syntaks som ligner på RDF-serialiseringen Turtle. SPARQL introduserer variabler for å kunne spesifisere hvilke deler av grafmønsteret som skal returneres som en del av resultatet [41, side 262]. Figur 3.6 på neste side viser et

eksempel på en SPARQL-spørring, og spørringen returnerer alle personer og navn på disse. Her brukes `?person` og `?navn` som variabler, hvor disse variablene representerer mulige konkrete verdier som variablene kan få etter at spørringen er blitt evaluert på datasettet. Legg også merke til at prefikser kan defineres på en lignende måte som i Turtle ved hjelp av nøkkelordet `PREFIX`. I resten av SPARQL-eksemplene antar vi at prefiksene er korrekt satt opp.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dsf: <http://sws.ifi.uio.no/dsf/henriwi/>

SELECT ?person ?navn
WHERE {
  ?person rdf:type dsf:Person;
          dsf:navn ?navn .
}
```

Figur 3.6: Eksempel på en SPARQL-spørring.

Hvis spørringen i figur 3.6 utføres over RDF-grafen i figur 3.1 på side 23, vil resultatet bli som vist i tabell 3.1.

person	navn
dsf:15047323394	"Ola Nordmann"

Tabell 3.1: Resultatet av SPARQL-spørringen i figur 3.6.

Blanke noder

Som nevnt i avsnitt 3.2.1 på side 23, er det vanlig at RDF-dokumenter består av blanke noder, og det er derfor viktig at dette støttes i SPARQL. SPARQL bruker samme syntaks som Turtle for å angi en blank node, nemlig `_:b1` for en blank navngitt node, og `[ ... ]` for å definere en blank, ikke-navngitt node.

Spørringen i figur 3.7 på neste side viser hvordan blanke noder kan brukes i en SPARQL-spørring, og spørringen returnerer gatenavnet og postnummeret fra en persons adresse.

Komplekse grafmønstre

I tillegg til å matche på enkle grafmønstre, er det mulig å kombinere enkle grafmønstre og en rekke forskjellige operatorer til å skrive mer komplekse spørringer.

OPTIONAL I et datasett bestående av personer, er det ikke sikkert alle personene har lagret et navn i datasettet. Som tabell 3.1 viser, får vi

```
SELECT ?gatenavn ?postnummer
WHERE {
  dsf:15047323394 dsf:adresse [
    dsf:gatenavn ?gatenavn ;
    dsf:postnummer ?postnummer ;
  ] .
}
```

Figur 3.7: Bruken av blanke noder i en SPARQL-spørring.

kun ut de personene som er registrert med et navn hvis vi utfører spørringen i figur 3.6 på forrige side. Dette er fordi SPARQL kun henter de personene hvor det er treff på *hele* grafmønsteret. Hvis vi ønsker å få ut alle personene, og personens navn hvis de har et, kan vi bruke `OPTIONAL`. Vi deler da spørringene opp i to grafmønstre, og lager det mønsteret hvor vi spør etter navnet til et valgfritt mønster. Figur 3.8 viser hvordan `OPTIONAL` kan brukes til å få ut alle personene, og hvis de har et navn, returnere dette i tillegg.

```
SELECT ?person ?navn
WHERE {
  ?person rdf:type dsf:Person .
  OPTIONAL {
    ?person dsf:navn ?navn .
  }
}
```

Figur 3.8: Eksempel på en SPARQL-spørring med bruk av `OPTIONAL`.

UNION `UNION` brukes til å spesifisere flere alternative grafmønstre, hvor resultatet blir returnert hvis det er treff i ett eller flere av disse. Figur 3.9 viser et eksempel på en spørring med bruk av `UNION`. Eksempelet viser hvordan `UNION` kan brukes for å ta hensyn til at flere predikater kan anvendes til å referere til etternavnet til en person.

```
SELECT ?person ?navn
WHERE {
  ?person rdf:type dsf:Person .
  { ?person dsf:etternavn ?navn .}
  UNION
  { ?person dsf:slektsnavn ?navn . }
}
```

Figur 3.9: Eksempel på en SPARQL-spørring med bruk av `UNION`.

FILTER Kombinasjon av enkle grafmønstre, og `OPTIONAL` og `UNION` vil gi et spørrespråk som ikke vil være særlig kraftig for praktisk bruk i applikasjoner. Det vil blant annet ofte være nødvendig å kunne hente ut data hvor verdiene er innenfor et bestemt område. For å få til dette

med SPARQL kan filteret benyttes til å redusere antall resultater, ved kun å returnere de resultatene som oppfyller filterkriteriene. Hvis et filterkriterium returnerer sannhetsverdien `true`, blir variablene i grafmønsteret tilordnet verdier, men hvis filterkriteriet returnerer `false` blir variablene ikke tilordnet verdier, og neste treff i datasettet blir evaluert. Det er verdt å merke seg at filtre, i motsetning til grafmønstre, ikke er basert på RDF modellen, men kan inneholde alle krav som kan bli sjekket beregningsmessig [41, side 272].

Figur 3.10 viser et eksempel på bruken av filter. Spørringen returnerer alle personer som har en alder større enn 20. Det er mulig å bruke operatorer som «=» (lik), «<» og «>» (større/mindre enn), «>=» og «<=» (større/mindre eller lik) og «!=» (forskjellig fra) i filteret, og alle operatorene er definert for datatypene som støttes av SPARQL.

```
SELECT ?person ?alder
WHERE {
  ?person rdf:type  dsf:Person ;
          dsf:alder ?alder .
  FILTER(?alder > 20)
}
```

Figur 3.10: Eksempel på en SPARQL-spørring med bruk av `FILTER` og sammenligningsoperatorer.

I tillegg til å kunne bruke sammenligningsoperatorer, er det definert en rekke funksjoner. Noen av disse er:

`BOUND(A)` Returnerer `true` hvis variabelen A har fått angitt en verdi.

`isURI(A)` Returnerer `true` hvis variabelen A er en URI.

`isBLANK(A)` Returnerer `true` hvis variabelen A er en blank node.

`REGEX(A,B)` Her er A en variabel, mens B er et regulært uttrykk som matches mot A. Hvis det er en match mellom innholdet i A og det regulære uttrykket, returneres `true`.

Det er også mulig å kombinere flere filteruttrykk ved hjelp av de boolske operatorene «||» (`ELLER`), «&&» (`OG`) og «!» (`IKKE`)

Modifikatorer

Selv om det er mulig å redusere antall resultater ved hjelp av blant annet filteret, er det ofte ønskelig å kunne modifisere resultatet av en spørring ytterligere. SPARQL har derfor en rekke modifikatorer som kan brukes til nettopp dette.

ORDER BY Denne modifikatoren brukes for å sortere resultatet basert på en eller flere variabler. Standardoppførselen er at resultatet blir sortert stigende, men det er også mulig å spesifisere hvordan sorteringen

skal skje ved å bruke operatorene `ASC` for stigende og `DESC` for synkende.

DISTINCT Denne modifikatoren brukes til å fjerne alle duplikater fra resultatet av en spørring. Resultater hvor alle variablene er tilordnet samme verdier blir slått sammen til én rad [41, side 281].

LIMIT og OFFSET Disse modifikatorene brukes for å returnere kun en del av et resultat. `LIMIT` brukes for å spesifisere hvor mange tripler som skal returneres, og `LIMIT 5` vil returnere 5 tripler. `OFFSET` brukes for å definere hvor i resultatsettet man skal starte, og `OFFSET 25` vil si at man skal starte med den 25. trippelen. Operatorene kan videre kombineres slik: `LIMIT 5 OFFSET 25`, noe som vil føre til at 5 tripler blir returnert, startende med trippel nummer 25.

Resultatformater

I alle eksemplene hittil er nøkkelordet `SELECT` benyttet, som returnerer resultatet av spørringen på tabellform. Selv om en slik representasjon av resultatet er nyttig i mange sammenhenger for å prosessere resultatet, egner ikke alltid RDF seg så godt på tabellformat, fordi det ofte er nødvendig å duplisere deler av tabellen [41, side 276]. I tillegg vil det ofte være ønskelig å kunne få presentert resultatet som en RDF-graf og ikke på en tabellstruktur. Blant annet derfor finnes det flere nøkkelord som gjør det mulig å returnere resultatet på andre formater.

CONSTRUCT `CONSTRUCT` returnerer en ny RDF-graf som er spesifisert av en graf-mal, hvor malen beskriver hvordan grafen som skal returneres skal se ut [41, side 277]. Dette resultatformatet minner om `VIEW` i relasjonelle databaser, som brukes til å kombinere og presentere data for sluttbruker uten å alterere databasestrukturen eller persistere dataene. På samme måte kan `CONSTRUCT`-spørringer brukes til å modifisere presentasjonen av dataene og returnere disse til sluttbruker, uten at dataene endres i databasen.

Et eksempel på en `CONSTRUCT`-spørring er vist i figur 3.11 på neste side. Denne spørringen viser hvordan en ny trippel basert på informasjonen i det eksisterende datasettet kan genereres, og spørringen lager en ny trippel som går fra adressen til en person til personen. Hvis disse triplene så blir lagt inn i datasettet, vil det eksistere en relasjon fra alle personer til adressen, og også en relasjon fra adressen til personen. I seksjon 4.2.2 på side 53 viser vi hvordan `CONSTRUCT`-spørringer fungerer som en sentral del i konvertering av data til RDF, og hvordan denne type spørring fungerer som et viktig ledd i utviklingen av en ny distribusjonsløsning.

DESCRIBE Det kan ofte være ønskelig å få en oversikt over hvilke data som er lagret i datasettet det spørres mot, og `DESCRIBE` kan bidra til å skaffe en slik oversikt. Figur 3.12 på neste side viser hvor-

```
CONSTRUCT { ?adresse dsf:erAdresseTil ?person }
WHERE {
    ?person rdf:type dsf:Person ;
    dsf:adresse ?adresse .
}
```

Figur 3.11: Eksempel på en `CONSTRUCT`-spørring.

dan en `DESCRIBE`-spørring kan brukes til å hente ut all informasjon om en persons adresse, og resultatet vil være en RDF-graf som beskriver adresseressursen til personen identifisert av URI-en: `dsf:01010112345`.

```
DESCRIBE ?adresse
WHERE {
  dsf:01010112345 dsf:adresse ?adresse .
}
```

Figur 3.12: Eksempel på en `DESCRIBE`-spørring.

Et problem med å bruke `DESCRIBE`-spørringer er at det ikke er definert nøyaktig hvor mye informasjon som skal returneres om en ressurs. Det vil si at det ikke er definert i SPARQL-standarden hvor mange kanter man skal bevege seg gjennom RDF-grafen før resultatet blir returnert. Det kan dermed være at forskjellig svar blir returnert avhengig av hvilken SPARQL-implementasjon som blir brukt.

ASK `ASK` brukes til å teste om et grafmønster har en løsning eller ikke [35], og returnerer `true` hvis det eksisterer en løsning, `false` hvis ikke. Figur 3.13 viser hvordan `ASK` kan brukes til å spørre om det eksisterer en person med fornavn «Ola» i datasettet.

```
ASK { ?x dsf:fornavn "Ola" }
```

Figur 3.13: Eksempel på en `ASK`-spørring.

Navngitte grafer

Et fjerde element kan legges til i hver trippel, som gjør det mulig å tilordne kontekst til hver enkelt trippel og videre definere en navngitt graf for det aktuelle datasettet. Ved å bruke nøkkelordene `FROM`, `FROM NAMED` og `GRAPH` kan spørringer som jobber mot datasett med navngitte grafer skrives. Hvis ingen av nøkkelordene brukes, vil det spørres mot dataene i den såkalte standard-grafen, det vil si alle de triplene som ikke har et fjerde element oppgitt.

Figur 3.14 på neste side viser hvordan SPARQL kan brukes til å jobbe mot navngitte grafer. `FROM` brukes for å legge til den navngitte grafen

til standard-grafen, mens `FROM NAMED` og `GRAPH` brukes for å adressere en aktiv graf. Figuren viser hvordan datasettet har en navngitt graf som er identifisert av URI-en `http://sws.ifi.uio.no/dsf/henriwi/felles`, og en annen navngitt graf som er identifisert av URI-en `http://sws.ifi.uio.no/dsf/henriwi/nav`, og hvordan personers navn hentes fra én graf, mens personers adresser hentes fra en annen. Prefikset `nav:` er tilordnet URI-en: `http://sws.ifi.uio.no/dsf/henriwi/nav/`.

```
SELECT ?person ?navn ?adresse
FROM <http://sws.ifi.uio.no/dsf/henriwi/felles>
FROM NAMED <http://sws.ifi.uio.no/dsf/henriwi/nav>
WHERE {
  ?person rdf:type dsf:Person;
      dsf:navn ?navn .
  OPTIONAL {
    GRAPH <http://sws.ifi.uio.no/dsf/henriwi/nav> {
      ?person nav:adresse ?adresse .
    }
  }
}
```

Figur 3.14: Eksempel på en SPARQL-spørring med bruk av navngitte grafer.

SPARQL 1.1

SPARQL 1.1 [35] er en ny versjon av SPARQL-standarden som er under standardisering, et såkalt *working draft*, og har flere interessante muligheter som vil bidra til et enda kraftigere spørregrensesnitt i den nye distribusjonsmodellen, enn det som er mulig med det vi hittil har vist.

Aggregering Ved hjelp av operatorer som `COUNT`, `SUM`, `MIN` og `MAX` med flere er det mulig å anvende aggregering over grupper av resultater [35].

Subspørringer Subspørringer gjør det mulig å bygge en eller flere SPARQL-spørringer inn i en annen SPARQL-spørring. Dette gjøres normalt hvis det ikke er mulig å oppnå resultatet kun ved bruk av én spørring [35].

Property path En *property path* er en mulig vei gjennom en graf mellom to grafnoder [35], og muliggjør mer konsise utrykk for noen SPARQL-spørringer som ofte blir mer komplekse enn de trenger å være. *Property paths* gjør det for eksempel mulig å skrive grafmønstre på følgende måte istedenfor å bruke `UNION`:

```
    { ?p rdf:type dsf:Far|dsf:Mor .}
```

Negasjon Den første versjonen av SPARQL har dårlig støtte for negasjon[2].

[2]Det er mulig å oppnå negasjon ved å bruke OPTIONAL og BOUND(), men dette kan fort føre til kompliserte spørringer, og anses ikke å være i henhold til intensjonen bak SPARQL.

Grunnen er at semantiske teknologier antar en åpen-verden semantikk, hvor det ikke vil være hensiktsmessig å ha negasjon siden vi antar at vi ikke jobber med et komplett datasett, og dermed ikke kan si at noe er usant i henhold til datasettet, bare fordi vi vet at det ikke er sant.

I lukkede systemer og andre applikasjoner hvor datasettet antas å være komplett, vil det ofte være ønskelig å ha støtte for negasjon. SPARQL 1.1 har derfor støtte for negasjon ved operatoren `FILTER NOT EXISTS`. Denne operatoren brukes til å filtrere bort resultater hvor det *ikke* er treff på grafmønsteret. Figur 3.15 viser hvordan dette kan brukes til å hente alle personer som ikke har et navn.

```
SELECT DISTINCT * WHERE {
       ?person a dsf:Person .
       FILTER NOT EXISTS { ?person dsf:navn ?navn }
}
```

Figur 3.15: Eksempel på bruk av negasjon i SPARQL 1.1.

SPARQL 1.1 Update En annen viktig del av SPARQL 1.1-spesifikasjonen er SPARQL 1.1 Update, som er et språk for å oppdatere RDF grafer [31]. SPARQL 1.1 Update er ment å være et standardisert språk for å oppdatere RDF-grafer i en såkalt *graph store*, mer kjent som triple store (se avsnitt 3.3.1 på side 40). Ved hjelp av nøkkelord som `INSERT` og `DELETE` kan data henholdsvis legges til eller fjernes.

3.2.3 Vokabularer, RDFS og OWL

Ved hjelp av språkene som hittil er presentert, RDF og SPARQL, har vi et dataformat og spørrespråk som vil bidra til å lage en forbedret distribusjonsløsning for Folkeregisteret. Men som nevnt i kapittel 2 vil det være nødvendig med en modell som kan brukes til eksplisitt å definere begreper og relasjoner mellom disse, og dermed kunne brukes til en enklere dataintegrasjon av heterogene kilder. Hvis vi ser nærmere på grafen i figur 3.1 på side 23, vil de fleste intuitivt forstå hva de forskjellige nodene og kantene betyr. Nemlig at det handler om to individer som er personer, og at disse er ektefeller. Videre vil vi intuitivt forstå meningen til de forskjellige begrepene, og hvordan begrepene kan anvendes. For eksempel vil vi forstå at ektefelle-predikatet skal koble to personer sammen og ikke for eksempel to adresser, og at en person er en generalisering av mann og kvinne (selv om dette ikke blir vist i figur 3.1 på side 23). For datamaskiner derimot vil et RDF-dokument kun være et sett med tekststrenger uten noen form for mening eller sammenheng. Serialiseringer gjør dataene maskinleselige og dermed mulige å prosessere for maskiner, men det er ønskelig å gi maskinen mer informasjon slik at maskinen i større grad forstår meningen (semantikken) til dataene, informasjon som eksplisitt sier

det et menneske intuitivt forstår. En måte å beskrive denne nødvendige bakgrunnsinformasjonen på er å benytte seg av vokabularer.

Et vokabular er en samling med identifikatorer (URI-er) med en klart definert mening [41, side 33]. En slik samling definerer konsepter og relasjoner innenfor et bestemt domene, og sammenhengen mellom disse. Disse definisjonene og relasjonene er videre beskrevet på en maskinprosesserbar måte, det vil si at bakgrunnskunnskapen i et vokabular er lagret på en måte som gjør at maskiner kan lese og tolke denne informasjonen. I tillegg vil det, ved å bruke veldefinerte eksisterende vokabularer, gjøre det enklere for mennesker å forstå meningen med et RDF-dokument.

Det eksisterer en rekke veldefinerte vokabularer som kan brukes når RDF-dokumenter skal lages, og et eksempel på et vokabular er RDF-vokabularet som er definert på URI-en `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. Dette vokabularet får som regel prefikset `rdf:` og en ressurs fra dette vokabularet er:

```
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
```

Denne ressursen brukes for å angi typen til en ressurs, som for eksempel trippelen:

```
dsf:ola rdf:type dsf:Person
```

som sier at «Ola er en person». Denne ressursen er så vanlig å bruke at den har fått et eget tegn i Turtle-serialiseringen, nemlig bokstaven «a».

Et eksempel på et annet vokabular, er *Friend Of A Friend* (FOAF) [29]. Dette vokabularet inneholder et sett med URI-er for å definere mennesker og ulike forhold mellom disse. Selv om dette vokabularet ikke er definert av en offisiell standardiseringsorganisasjon, er vokabularet likevel såpass kjent, at det unngås forvirring rundt meningen av de forskjellige begrepene og relasjonene [41, side 35].

Ved å gjenbruke etablerte vokabularer unngår man å opprette URI-er for entiteter, konsepter og relasjoner som allerede har en veletablert definisjon. I tillegg gjør det det enklere å forstå innholdet av RDF-dokumentet hvis velkjente forhåndsdefinerte begreper blir benyttet. For eksempel vil det være lite hensiktsmessig å opprette et nytt «type»-predikat istedenfor å bruke det velkjente og etablerte `rdf:type`. Når RDF-dokumenter skal lages, lønner det seg derfor å prøve å gjenbruke identifikatorer fra eksisterende vokabularer der dette er mulig.

RDFS

RDF Schema (RDFS) er et språk for å beskrive vokabularer og kan dermed brukes til å spesifisere nødvendig bakgrunnsinformasjon, såkalt terminologisk kunnskap. Tanken med RDFS er å tilby et generisk språk slik at semantiske egendefinerte vokabularer kan bli utviklet [41, side 46]

og RDFS kan sees på som et enkelt modelleringsspråk. Ved hjelp av RDFS er det dermed mulig å lage vokabularer innenfor et domene hvor slike ikke eksisterer, eller hvor eksisterende vokabularer ikke strekker til.

RDFS lar en definere klasser og egenskaper hvor klasser kan sees på som en mengde med ressurser, mens egenskaper er det samme som predikater i avsnitt 3.2.1 på side 20 om RDF. Eksempelet i figur 3.16 viser hvordan RDFS kan brukes til å beskrive et vokabular.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix dsf: <http://sws.ifi.uio.no/dsf/henriwi/> .

dsf:Person    a rdfs:Class .
dsf:Mann      a rdfs:Class ;
              rdfs:subClassOf dsf:Person .

dsf:Kvinne    a rdfs:Class ;
              rdfs:subClassOf dsf:Person .

dsf:ektefelle a rdf:Property ;
              rdfs:subPropertyOf foaf:knows ;
              rdfs:domain dsf:Person ;
              rdfs:range  dsf:Person .
```

Figur 3.16: Et lite RDFS vokabular.

Dette vokabularet definerer at `dsf:Person`, `dsf:Mann` og `dsf:Kvinne` er klasser, og at `dsf:Mann` og `dsf:Kvinne` er subklasser av `dsf:Person`. Videre defineres det et predikat `dsf:ektefelle`, som er en subegenskap av `foaf:knows`. Dette viser hvordan nye vokabularer kan benytte seg av og utvide eksisterende vokabularer. Til slutt defineres domenet og verdiområdet til `dsf:ektefelle` til å være `dsf:Person`, det vil si at denne relasjonen relaterer to entiteter av typen `dsf:Person`. Som figur 3.16 viser benyttes RDF til å beskrive RDFS, og dette fører til at ethvert RDFS-dokument også er et gyldig RDF-dokument [41, side 46]. Dette gjør det igjen enkelt å prosessere RDFS-dokumenter, siden verktøy som støtter RDF kan benyttes.

For at maskiner skal kunne utnytte bakgrunnsinformasjonen som et vokabular gir, har RDFS en formell semantikk [36] som gir en klar matematisk definisjon av hvilke konsekvenser som kan utledes fra en RDF-graf. En slik semantikk sier, på en utvetydig måte, når et sett med påstander (i denne konteksten tripler) er en logisk konsekvens av et sett med andre påstander. Det vil si at semantikken inneholder en presis matematisk metode å si når en RDF-graf er en logisk konsekvens av en annen RDF-graf. En slik semantikk gjør det mulig å konkludere med hvilke utsagn som medfører andre utsagn, men den inneholder ingen algoritmiske midler for å utlede disse konsekvensene [41, side 90]. Derfor er semantikken videre

reflektert i et sett med regler, som gir en syntaktisk metode for å avgjøre om en RDF-graf medfører en annen RDF-graf. Disse reglene kan videre anvendes til å utlede ny informasjon basert på en RDF-graf og et vokabular, såkalt resonnering.

Hver regel består av et sett med premisser som, hvis de er tilstede i RDF-grafen, gjør at en ny trippel (konklusjonen) blir lagt til. Innenfor konteksten til semantiske teknologier vil alle reglene føre til at nye tripler blir lagt til. Det vil si at, basert på de eksisterende triplene i en RDF-graf, vil nye tripler bli lagt til i RDF-grafen. Et eksempel på en slik regel er *rdfs9* vist i figur 3.17. Regelen sier at hvis `uuu` er subklasse av `xxx`, og `vvv` er av typen `uuu`, så skal det legges til en trippel som sier at `vvv` også er av typen `xxx`.

$$\frac{\texttt{uuu rdfs:subClassOf xxx . vvv rdf:type uuu .}}{\texttt{vvv rdf:type xxx .}}$$

Figur 3.17: RDFS-regel *rdfs9*.

Hvis vi har en RDF-graf som inneholder trippelen `dsf:Ola rdf:type dsf:Mann` og vokabularet i figur 3.16 på forrige side, kan *rdfs9* anvendes til å legge til trippelen: `dsf:Ola rdf:type dsf:Person`. Nedenfor vises et strukturert oppsett av beviset hvor linje 1 og 2 angir premissene, mens linje 3 angir konklusjonen og hvilke linjer og regel som ligger til grunn for konklusjonen.

1. `dsf:Mann rdfs:subClassOf dsf:Person - P`
2. `dsf:Ola rdf:type dsf:Mann - P`
3. `dsf:Ola rdf:type dsf:Person - 1, 2, rdfs9`

Selv om et sett med slike regler gjør det mulig å syntaktisk avgjøre om en RDF-graf medfører en annen RDF-graf, er det ikke enkelt å lage en algoritme eller en automatisk prosess som kan sjekke om en RDF-graf medfører en annen. Grunnen er at det er en stor utfordring å finne ut når prosessen med å anvende nye regler skal slutte uten at man går glipp av relevante konsekvenser. Det å lage en slik automatisk prosess, eller en resonnerer, er derfor en veldig utfordrende oppgave som både krever innsikt i teoretiske sider og gode kunnskaper innenfor programvareutvikling [41, side 101].

RDFS kan brukes til å utvikle enkle modeller og vokabularer, men kan ikke bli brukt til å modellere mer kompleks kunnskap. Det er blant annet ingen mulighet for å modellere mer komplekse begrepsdefinisjoner enn subklasser, men en av de mest vesentlige manglene er det at ikke er noen mulighet for å modellere negasjon i RDFS. Det vil si at ikke er mulig å modellere det faktum at et utsagn ikke er sant [41, side 106]. En annen ulempe er at det ikke er et klart skille mellom modell og data i semantikken til RDFS. Det vil si at det er ingenting i veien for at en ressurs både kan være en klasse og et individ. I neste seksjon om OWL viser vi hvordan OWL kan

brukes til å modellere mer kompleks kunnskap og negasjon, samt viser fordeler med at det er et klart skille mellom modell og data.

OWL

Web Ontology Language (OWL) er en W3C standard for å modellere ontologier [41, side 111]. En ontologi er en beskrivelse av kunnskap om et spesifikt domene [41, side 47], og inneholder som regel mer kompleks informasjon enn det et vokabular inneholder. OWL eksisterer i to versjoner, OWL [42] og den nyere OWL 2 [76] . Når denne oppgaven snakker om OWL, menes OWL 2.

OWL utvider RDFS og gjør det mulig å modellere mer kompleks kunnskap enn RDFS, og ved hjelp av OWL kan komplekse modeller innenfor et bestemt domene utvikles. OWL er basert på beskrivelseslogikk [3], en form for logikk som har blitt brukt mye innenfor kunnskapsrepresentasjon. Beskrivelseslogikk, og dermed OWL, har, i motsetning til RDFS, et klart skille mellom klasser og individer. Dette gjør at det ikke er mulig med formell metamodellering i OWL, men det er flere fordeler ved å ha et slikt skille. For det første fører det til at det er enkelt å separere utsagnene som inneholder informasjon om klasser og forholdet mellom disse (skjemainformasjon) og utsagnene om dataene. Dette gjør det for eksempel enkelt å gjenbruke dataene sammen med andre skjemautsagn, uten å måtte endre på dataene [10]. En annen fordel er at ulike resonnerere kan anvendes på ulike typer informasjon. For eksempel kan en resonnerer som er godt egnet til å utlede skjemainformasjon brukes på denne delen av ontologien, mens en resonnerer som er godt egnet til å utlede informasjon knyttet til instansdata, kan anvendes til dette.

OWL-ontologier tilbyr klasser, roller, individer og dataverdier for å modellere. For en fullstendig oversikt over mulighetene til OWL, henvises det til [4] og [43].

Klasser blir brukt til å gruppere individer som har noe til felles, for eksempel alle personer, eller alle personer som har innvandret til Norge. En klasse kan sees på som et sett med individer, og det er ingenting i veien for at et individ kan være medlem av flere klasser. For klasser støtter OWL blant annet mengdeteoretiske operasjoner som snitt, union, komplement, opplisting, subklasse, ekvivalens og disjunksjon.

Roller beskriver forhold mellom klasser og individer, og det eksisterer tre typer roller: objektroller, dataroller og annotasjonsroller. Objektroller er roller som relaterer individer til individer, for eksempel en far til sin sønn, mens dataroller er roller som angir dataverdier (literaler) til et individ, for eksempel alderen til en person. Annotasjonsroller brukes for å angi tilleggsinformasjon om selve ontologien, klasser og individer, for eksempel hvem som har utviklet ontologien og dokumentasjon av klasser og roller. Denne informasjonen

kan sees på som ren metainformasjon og tekstlig dokumentasjon av ontologien, og vil ikke bli tatt hensyn til under resonnering. Rollene kan relateres til andre roller ved for eksempel å være ekvivalente-, disjunkte- og subroller. I tillegg kan rollene ha ulike karakteristikker som funksjonell, invers, inversfunksjonell, refleksiv, irrefleksiv, symmetrisk, asymmetrisk og transitiv. Det er en del færre karakteristikker som kan sies om dataroller og annotasjonsroller enn om objektroller. I tillegg kan domene og verdiområde defineres.

Individer representerer objekter i domenet som modelleres, og kan sees på som instanser av en klasse. Det er viktig å bemerke at siden OWL har en antagelse om ikke-unike navn, så vil individer med ulike navn verken være like eller forskjellige før dette blir eksplisitt definert. De to viktigste utsagnene som kan sies om individer er derfor likhet og ulikhet mellom individer.

Figur 3.18 på neste side viser et eksempel på en ontologi serialisert til Turtle. Ontologien utvider eksempelet modellert i RDFS i figur 3.16 på side 35 og viser hvordan OWL kan brukes til å modellere at `dsf:Mann` og `dsf:Kvinne` er disjunkte, det vil si at ingen individer både kan være `dsf:Mann` og `dsf:Kvinne`. Videre har det blitt modellert at `dsf:ektefelle` er en objektrolle og at denne er symmetrisk. Det vil si at hvis person `a` er relatert til `b` med rollen `dsf:ektefelle`, så er også `b` relatert til `a` med samme rolle. Til slutt er det lagt til en datarolle `dsf:alder` som er funksjonell. Det vil si at en person kun kan ha én alder.

OWL har som RDFS en formell semantikk som er reflektert i en rekke regler som kan anvendes til å utlede ny informasjon. I tillegg til å gjøre implisitt informasjon eksplisitt ved å legge til tripler basert på en eksisterende RDF-graf og ontologien, kan resonnering brukes til å sjekke om en modell er konsistent. Det vil si at en resonnerer kan anvendes for å sjekke at en OWL-modell ikke inneholder noen selvmotsigelser.

Ontologier har et vidt spekter av bruksområder, og det stilles derfor ulike krav til kompleksiteten til en ontologi avhengig av hva ontologien skal benyttes til. Noen ontologier skal brukes i applikasjoner hvor resonnering må kunne utføres raskt, og det er derfor viktig at ontologien ikke er for kompleks, mens andre ontologier ikke har noen krav til at det skal kunne utføres resonnering, og det trengs derfor ikke stilles noen krav til kompleksiteten. Generelt sett er OWL et veldig uttrykksfullt språk, og kan derfor være vanskelig å implementere og jobbe med både for mennesker og maskiner [43]. Det eksisterer derfor flere versjoner av OWL som har forskjellige bruksområder.

OWL 2 DL kan sees på som «vanlig» OWL. Ifølge W3C er OWL 2 DL brukt uformelt til å beskrive ontologier som tolkes ved hjelp av den direkte semantikken [53] til OWL [76], som er en semantikk som er veldig tett koblet til semantikken til beskrivelseslogikk. Dette i motsetning til OWL 2 Full, som er brukt uformelt til å beskrive ontologier som tolkes ved hjelp av den RDF-baserte semantikken [21], hvor det blant annet ikke er

```
@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 foaf: <http://xmlns.com/foaf/0.1/> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix dsf:  <http://sws.ifi.uio.no/dsf/henriwi/> .

dsf:Person    a owl:Class .
dsf:Mann      a owl:Class ;
              rdfs:subClassOf dsf:Person ;
              owl:disjointWith dsf:Kvinne .

dsf:Kvinne    a owl:Class ;
              rdfs:subClassOf dsf:Person ;
              owl:disjointWith dsf:Mann .

dsf:ektefelle a owl:ObjectProperty ;
              a owl:SymmetricProperty ;
              rdfs:subPropertyOf foaf:knows ;
              rdfs:domain dsf:Person ;
              rdfs:range  dsf:Person .

dsf:alder     a owl:DatatypeProperty ;
              a owl:FunctionalProperty ;
              rdfs:domain dsf:Person ;
              rdfs:range xsd:int .
```

Figur 3.18: Ontologi i OWL serialisert til Turtle.

et skille mellom klasser og individer, og hvor kort fortalt «alt» er lov. Dette fører til at OWL 2 Full ikke er avgjørbart, det vil si at det ikke er noen garanti for at et svar vil bli returnert når resonnering utføres, og OWL 2 Full er derfor ikke attraktivt hvis resonnering er ønskelig. Et viktig poeng er at selv om OWL 2 DL er avgjørbart, kan modellen bli så kompleks at resonneringsproblemer, som å sjekke konsistens, har veldig høy kompleksitet.

Det er også utviklet flere profiler, eller subsett av OWL. Det er flere utsagn fra OWL som har høyere kompleksitet enn andre utsagn, og de forskjellige profilene er en syntaktisk restriksjon av OWL. Det vil si at profilene inneholder noen utsagn som er lovlige, men også mange utsagn som ikke er lov innenfor den aktuelle profilen. Profilene som eksisterer for OWL er:

OWL 2 EL Denne profilen er spesielt nyttig i applikasjoner som benytter ontologier som inneholder svært store mengder egenskaper og/eller klasser [52]. Et eksempel på en ontologi som anvender denne profilen er den biomedisinske ontologien SNOMED CT[3], som er en ontologi som definerer mer enn hundre tusen klasser.

OWL 2 QL Denne profilen er rettet mot applikasjoner som bruker svært store mengder instansdata, og hvor svar fra spørringer er den viktigste oppgaven [52]. Profilen er også designet på en måte som gjør det mulig å skrive om SPARQL-spørringer til en tradisjonell SQL-spørring uten å måtte forandre på dataene.

OWL 2 RL Denne profilen er ment for applikasjoner som krever skalerbar resonneringen uten å måtte ofre for mye uttrykkskraft. Profilen er designet for å imøtekomme OWL 2-applikasjoner som kan ofre noe uttrykkskraft for effektivitet, samt RDF(S)-applikasjoner som trenger litt ekstra uttrykkskraft [52].

3.3 Teknologi og arkitektur

Hittil har kapittelet dreid seg om prinsipper og språk innenfor semantiske teknologier. Men for å kunne anvende disse språkene i en ny distribusjonsløsning, er det nødvendig med teknologier som muliggjør lagring, forvalting og distribusjon av dataene som er beskrevet av språkene.

3.3.1 Triple store

For å kunne bruke RDF-data i applikasjoner er det nødvendig med en komponent som gjør at dataene kan persisteres. En triple store er en slik komponent, og er designet for lagring og henting av RDF-data [63]. Det finnes flere måter å implementere en triple store på og det eksisterer en rekke forskjellige implementasjoner, men boken *Semantic Web*

[3] `http://www.ihtsdo.org/snomed-ct/`

Programming [38, side 144-145] deler de forskjellige implementasjonene inn i to hovedgrupper.

Den første gruppen er relasjonsbaserte triple stores, hvor vanlige tabeller benyttes til å lagre RDF-dataene. Det er flere måter å implementere dette på, men en måte er å ha en tabell som inneholder alle tripler og som igjen har referanser til de forskjellige verdiene av disse triplene. Det å hente ut data fra en slik triple store kan ofte være ineffektivt, da det kan kreve mange koblinger av tabeller og at store deler av tabellen må traverseres før det som søkes etter blir funnet. Den andre gruppen er såkalte graf-baserte triple stores. Disse er blitt utviklet som datastrukturer som prøver å modellere strukturen til en RDF-graf mer direkte, og på denne måten unngå noen av ytelsesproblemene som kan oppstå med en relasjonsbasert triple store.

3.3.2 SPARQL-endepunkt

En triple store i seg selv er ikke i stand til å distribuere data til en bruker, det er kun et system for lagring og forvalting av RDF-data. Et SPARQL-endepunkt er et grensesnitt som gjør det mulig for brukere å utføre SPARQL-spørringer mot en triple store for å hente ut data [64]. Resultatet fra spørringen blir returnert på ett eller flere maskin-prosesserbare formater.

Et SPARQL-endepunkt fungerer som et grensesnitt mot en triple store, hvor brukeren kan skrive SPARQL-spørringer for å hente ut dataene. Brukeren vil kommunisere med endepunktet, som igjen vil kommunisere med en triple store. For å hente ut data ved hjelp av endepunktet sendes en HTTP-forespørsel til endepunkt med en SPARQL-spørring som parameter. For å utveksle SPARQL over HTTP, har W3C definert standarden *SPARQL Protocol for RDF* [33] som bruker WSDL 2.0 [22] til å beskrive hvordan en SPARQL-spørring skal formidles fra en klient til en spørringsprosessor som kan behandle forespørselen.

Siden HTTP brukes til kommunikasjon og dataoverførsel, gjør dette det mulig for mange applikasjoner å bruke et SPARQL-endepunkt uten å måtte benytte et språkavhengig API, som er tilfellet med tradisjonelle teknologier som relasjonelle databaser. Et SPARQL-endepunkt gjør det også mulig å utføre nøyaktig de spørringene en bruker har behov for å utføre, så lenge personen kjenner vokabularet eller ontologien til dataene som er lagret. Dette i motsetning til mer tradisjonelle webtjenester, hvor alle spørringene som regel må være satt opp på forhånd.

Ved hjelp av forskjellig SPARQL-endepunkt som distribuerer data fra flere forskjellige kilder, er det mulig å skrive federerte SPARQL-spørringer [59]. Dette er spørringer som spenner over flere datasett og datakilder. En datakilde kan refereres til ved hjelp av den offentlige URL-en hvor endepunktet er publisert, og med en slik spørring er det mulig å kombinere data distribuert på tvers av systemer og til og med etater. SPARQL-endepunkter gjør det dermed mulig å stille SPARQL-spørringer over en

mengde forskjellige datasett «on the fly».

3.3.3 REST

Ved å la demonstratoren ha et SPARQL-endepunkt blir det mulig for sluttbrukere å skrive kraftige SPARQL-spørringer for å hente ut RDF-data. SPARQL er et forholdsvis enkelt spørrespråk, men kan for mange oppleves komplekst, og det er sikkert at ikke alle brukerne ønsker å lære seg dette. I tillegg er det strenge krav til hvilke data som skal distribueres fra DSF, og hvilke brukere som skal ha tilgang til de ulike dataene. Det er derfor ikke sikkert det er ønskelig at alle brukerne skal kunne skrive vilkårlige spørringer.

Et alternativ kan derfor være å sette opp demonstratoren som en såkalt RESTfull webtjeneste. REST (Representational State Transfer) er en type softwarearkitektur som egner seg for distribuerte hypermediasystemer [28]. En RESTfull arkitektur består av klienter og servere, hvor klientene sender forespørsler til serverne, og serverne prosesserer og returnerer passende resultater. Et prinsipp innenfor RESTfulle systemer, er at alt er eksponert som ressurser via et enhetlig grensesnitt, og det er ikke et mål å skjule distribusjon av data [79]. Dette i motsetning til mer tradisjonelle distribuerte systemer, som ofte bruker RPC [74] (Remote Procedure Calls) eller SOAP [34] (Simple Object Access Protocol), som prøver å skjule distribueringen og feilsituasjoner, og som gjør at programmerere kan jobbe med distribuerte systemer som om det var et lokalt system. REST er på mange måter et lettvektsalternativ til tradisjonelle webtjenester som SOAP, WSDL og RPC [25].

RPC-baserte systemer blir oftere brukt i enterprise-systemer på grunn av ønsket om å utvikle et homogent system hvor distribuerte systemer virker lokale, og fordi det gir utviklere et enklere miljø å utvikle i på grunn av måten distribusjonen skjules. RESTfulle systemer blir derimot mer og mer populære på weben, hvor det ikke er et mål å bygge et stort homogent system, men hvor det er behov for enkelt å kunne ta i bruk heterogene kilder. Store aktører som Google og Flickr tilbyr både SOAP-systemer og RESTfulle systemer, og de RESTfulle systemene blir mer brukt enn de mer tradisjonelle SOAP-systemene [79].

En RESTfull webtjeneste er en tjeneste som eksponeres gjennom HTTP-protokollen og som er basert på prinsippene til REST. Tjenesten er en samling av ressurser hvor ressursene er representert som URL-er på følgende format [5]:

```
protokoll://tjener/applikasjonssti/ressurstype/ressursid
```

Her fungerer `tjener` og `applikasjonssti` som navnerom for alle ressursene innenfor en kontekst, mens `ressurstype` og `ressursid` unikt identifiserer en samling med ressurser av en bestemt type eller én unik ressurs.

Videre brukes fire ulike HTTP-metoder for å modifisere ressursene: `GET`, `PUT`, `POST` og `DELETE`, og tabell 3.2 viser hva de forskjellige HTTP-metodene brukes til. Hvis informasjon om en ressurs av type `person` med id 1 skal hentes fra den RESTfulle tjenesten `http://sws.ifi.uio.no/dsf/henriwi/rest/`, sendes en HTTP-GET forespørsel mot følgende URL: `http://sws.ifi.uio.no/dsf/henriwi/rest/person/1`.

GET	Henter informasjon om ressursen(e).
PUT	Erstatter ressursen(e), eller oppretter den/de hvis den/de ikke eksisterer.
POST	Oppretter en/en samling ressurs(er)
DELETE	Sletter ressursen(e)

Tabell 3.2: Oversikt over hvordan de forskjellige HTTP-metodene brukes til å modifisere ressurser i en RESTfull tjeneste.

Responsen som en RESTfull tjeneste gir, avhenger av hva slags type innhold klienten ønsker å motta. Denne innholdstypen, eller `MIME`-typen, settes som verdi i *Accept*-feltet i HTTP-headeren i forespørselen fra klienten. Noen vanlige innholdstyper er `application/json` som betyr at klienten ønsker å få respons som JSON[4], og `application/xml` som betyr XML.

Ved å skjule den underliggende arkitekturen til SPARQL-endepunktet på denne måten, trenger ikke brukeren å forholde seg til kompleksiteten i RDF og SPARQL. Hovedforskjellen mellom et RESTfullt SPARQL-endepunkt[5] og en tradisjonell RESTfull webtjeneste, vil være at endepunktet representerer og returnerer data som RDF. Forskjellen vil også være at i en tradisjonell webtjeneste er det ofte en relasjonell database som brukes som underliggende lagringsstruktur, mens et RESTfullt SPARQL-endepunkt har en triple store som underliggende lagringstruktur.

Ulempen er at det, som i en standard webtjeneste, er nødvendig å sette opp alle spørringene på forhånd, mens det med et standard SPARQL-endepunkt kan skrives vilkårlige spørringer.

3.3.4 Lenkede data

Lenkede data refererer til data som er publisert på weben på en slik måte at de er maskin-leselige, betydningen er eksplisitt definert, de er lenket opp mot andre eksterne datasett og de kan bli lenket til av andre datasett [19]. Tim Berners-Lee sier selv at lenkede data er «semantisk web utført på riktig måte» [14], og grunnideen til lenkede data er å ta utgangspunkt i den eksisterende weben og dens arkitektur, og utvide denne med strukturerte,

[4] JavaScript Object Notation — `http://www.json.org/`

[5] Med dette mener vi en RESTfull webtjeneste som kommuniserer med et SPARQL-endepunkt.

delte data. For vår del er lenkede data interessant fordi det blant annet har mange felles prinsipper med REST, og er en måte å prøve å skjule en del av kompleksiteten til RDF og SPARQL.

En av grunnene til at en semantisk web ikke har blitt realisert, kan være at det krever en enighet om begreper på tvers av personer og organisasjoner. Disse begrepene kan så brukes i felles ontologier for deretter å koble disse ontologiene til data. Dette kan sees på som en toppen-og-ned tilnærming, mens lenkede data har en nedenfra-og-opp tilnærming, det vil si at det tas utgangspunkt i dataene, og så kobles dataene opp mot en eller flere ontologier eller vokabularer som kan passe til dataene. Først konsentrerer en seg om å få distribuert dataene som lenkede data, og deretter forsøkes det å koble dataene opp mot eksisterende ontologier eller utvikle egne. Som vist er det viktig at ontologier benyttes for å bidra med nødvendig bakgrunnsinformasjon, men det er likevel dataene som er det viktigste, og ikke det å bli enige om begrepsdefinisjoner.

Tim Berners-Lee presenterte i 2006 fire grunnprinsipper for hva lenkede data er [13].

1. Bruk URI-er til å navngi ting.
2. Bruk HTTP URI-er, så det kan gjøres oppslag på URI-ene ved hjelp av HTTP.
3. Når noen slår opp en URI, tilby nyttig informasjon gjennom standardene (RDF, SPARQL).
4. Inkluder lenker til andre URI-er, slik at det kan oppdages flere ting.

Hovedpoengene som kan trekkes frem er at URI-er brukes til å identifisere entiteter som i RDF. Videre bør HTTP URI-er (URL-er) benyttes, noe som sikrer at en representasjon av ressursen blir returnert når en HTTP-forespørsel sendes til URI-en som beskriver den aktuelle ressursen. Videre bør entitetene i datasettet være lagret og representert som RDF, og dermed gjøre det mulig å lage lenker fra eget datasett til andre datasett. Lenkede data er for mange synonymt med lenkede *åpne* data, som omhandler data som er fritt tilgjengelig på weben uten noen form for tilgangskontroll. Det er flere initiativer for å publisere offentlige data som lenkede åpne data, både i USA[6], Storbritannia[7] og Norge[8]. Berners-Lee har også laget et notat som beskriver hvordan det bør gåes frem for å publisere offentlige data på weben som lenkede data [15].

Selv om disse initiativene har et fokus på *åpne* data er det samtidig ingen ting i veien for at prinsippene som ligger bak lenkede data og semantiske teknologier kan brukes i lukkede applikasjoner og på data som ikke skal være åpne på weben [37]. Dataene som ligger i Folkeregisteret og SED-filene er ikke åpne på grunn av dataenes innhold og all den sensitive

[6] `http://www.data.gov/`
[7] `http://data.gov.uk/`
[8] `http://data.norge.no/`

informasjonen som ligger der. Men det er mulig å se for seg at det etterhvert vil eksistere en rekke datasett som følger prinsippene til lenkede data, og hvor disse datasettene er eksponert som RDF, innenfor brannmuren til for eksempel NAV eller SKD. Det vil da være enklere å kombinere data på tvers av systemer, hvis det allerede er lagt opp til at dette skal kunne gjøres med folkeregisterdataene.

3.3.5 Lenkede data, REST og SPARQL-endepunkt

Det er flere felles prinsipper mellom REST, semantisk web og lenkede data, og flere har forsket på dette tidligere. Battle og Benson [5] ser på hvordan man kan minske gapet mellom den såkalte Web 2.0 [57] og den semantiske weben ved hjelp av REST. Wilde og Hausenblas [79] mener at dagens SPARQL har en RPC-stil over seg, og viser hvordan prinsippene til REST kan anvedes på SPARQL, mens Page et. al [58] utforsker likhetene og forskjellene mellom REST og lenkede data, og foreslår hvordan de i kombinasjon kan hjelpe utviklere å fokusere på domenedrevne applikasjoner.

Både REST og lenkede data representerer ressurser som HTTP URI-er, og tanken er å enkelt kunne gi en respons i form av en representasjon av en gitt ressurs, ved å foreta en HTTP-forespørsel mot den aktuelle URI-en. Sammenlignet med SPARQL har REST og lenkede data en slakere læringskurve for sluttbrukere, og det er enklere å konsumere data som følger prinsippene til REST enn å være nødt til å lære seg SPARQL. Motivasjonen bak både REST og lenkede data er å tilby et enkelt, generisk grensesnitt som gjør terskelen lav for å konsumere og bruke data. Det å enkelt kunne foreta et oppslag etter en bestemt ressurs eller en ressurstype, vil være veldig nyttig for vårt aktuelle problem, for enkelt å se om en person med et bestemt fødselsnummer eksisterer i DSF. Selv om det først og fremst er på weben at RESTfulle systemer blir mer og mer populære, kan enterprise-systemer lære fra suksessen til weben fra et arkitektur- og designperspektiv [8]. Prinsippene som å bruke URI-er til å identifisere ressurser, og et enkelt grensesnitt basert på HTTP, vil også kunne tilføre verdi i et enterprise-miljø og i en type miljø som denne oppgaven jobber med.

Men det vil ikke være tilstrekkelig å kun tilby et RESTfullt grensesnitt eller lenkede data. Som vi har sett, og skal se senere, kan SPARQL benyttes til å lage kraftige spørringer som både kombinerer data fra flere kilder, og som også filtrerer bort uønskede resultater. Derfor vil et godt valg for den nye distribusjonsløsningen være en kombinasjon av et SPARQL-endepunkt som gjør det mulig for applikasjoner og brukere å stille komplekse SPARQL-spørringer, samt et grensesnitt som følger prinsippene til REST og lenkede data, noe som gjør det enkelt å slå opp på enkeltressurser eller samlinger av en bestemt ressurstype.

Kapittel 4

Transformering av data og utvikling av ontologier

De to foregående kapitlene har presentert problemet, overordnete krav til en ny løsning, samt språk og teknologier som bør anvendes i den nye distribusjonsløsningen for folkeregisterdata. Dette kapittelet beskriver konkret hvordan vi har gått frem for å designe en ny distribusjonsløsning for Folkeregisteret, og fokuserer på konverteringen fra rådata til RDF og utvikling av ontologier. Kapittel 5 beskriver implementasjonsdetaljene til demonstratoren.

Demonstratoren er basert på et anonymisert uttrekk av Folkeregisteret som er mottatt fra SKD, lik den dumpen som NAV mottar i figur 2.2 på side 8. Siden dette er et uttrekk av Folkeregisteret vil demonstratoren ikke være like rik på dataelementer som DSF er, men det er ingenting i veien for at de samme prinsippene og metodene senere kan bli anvendt på hele DSF. Videre har vi mottatt flere beskrivelser av SED-filene beskrevet i seksjon 2.4 på side 8 av NAV, og disse filene ligger til grunn for å distribuere SED-dataene på samme måte som DSF-dataene og dermed gjøre det mulig å vise hvordan disse to datakildene kan integreres.

Det første som må gjøres når semantiske teknologier skal anvendes for å lage en distribusjonsløsning, er å konvertere dataene til et format som gjør at semantiske teknologier kan brukes til å jobbe med dataene. Siden RDF fungerer som grunnstenen innenfor semantiske teknologier og ligger til grunn for de andre språkene og teknologiene, er det naturlig at dataene blir konvertert til RDF. Seksjon 4.1 og 4.2 beskriver dumpen med folkeregisterdata som er mottatt, og prosessen med å konvertere denne til RDF. Videre beskriver seksjon 4.3 hvordan en ontologi bør utvikles før seksjon 4.4 og 4.5 beskriver ontologiene for henholdsvis DSF og SED. Til slutt beskriver seksjon 4.6 hvordan ontologiene har blitt integrert.

4.1 Fra rådata til tabulære data

Dumpen med folkeregisterdata som vi har mottatt består av to filer. En inneholder informasjon om personer og deres attributter, mens den andre inneholder informasjon om personer og deres barn. Dataene er strukturert ved at alle dataene kommer rett etter hverandre uten skilletegn, én linje per person. Eneste måten å vite hvordan dataene er strukturert, er å ha tilgang til en beskrivelse som vi også mottok. Denne beskriver hvilke data som er på hvilket tegnnummer for hver linje i filen. For eksempel er personnummeret til en person fra tegn nummer 1 til tegn nummer 11.

Nedenfor vises et uttrekk av en linje fra dumpen som inneholder informasjon om personer og deres attributter, mens figur 4.1 viser et utdrag fra beskrivelsen av de samme dataene. Som figuren viser er tegn nummer 1 til 11 fødselsnummeret, tegn nummer 12 er en statuskode, mens tegn nummer 13 til 20 angir en eventuell dødsdato. I eksempelet nedenfor er det her bare nullverdier og det betyr at personen ikke er død. Til slutt er det personens slektsnavn fra tegn nummer 21 til 70, mens personens fornavn er fra tegn nummer 71 til 120. Merk at mellomrom som eksisterer i dumpen, og som skal telles med, er fjernet fra linjen nedenfor for å øke lesbarheten.

```
01010750160100000000TOPSTADTOMAS
```

NAVN		**TEKST**			**FORMAT**		**LENGDE**	
KFXNDIST		**Load-register til distributører**			**F**		**1300**	
Felt nr	Nivå	Datanavn	PIC	Data type	Ant	Fra	Til	Kommentar
1	01	LOAD-RECORD						
2	05	L-FODSELSNR	9(11)		11	1	11	
3	05	L-STATUSKODE	X		1	12	12	
4	05	L-DATO-DOED	9(8)		8	13	20	=ÅÅÅÅMMDD
5	05	L-SLEKTSNAVN	X(50)		50	21	70	
6	05	L-FORNAVN	X(50)		50	71	120	

Figur 4.1: Utdrag fra beskrivelsen til dumpen med folkeregisterdata.

Det finnes en rekke verktøy som konverterer data fra ulike formater til RDF. Det finnes derimot ikke noe verktøy som kan konvertere data som er på et slikt format som dumpen med folkeregisterdataene er på, til RDF. Det kreves derfor et mellomsteg i konverteringen hvor rådataene konverteres til et tabulært format, hvor dataene er tydelig skilt, og hvor det ikke er nødvendig med den eksterne beskrivelsen for at en maskin skal forstå hvor de forskjellige dataelementene starter og slutter.

Et populært format som kan lagre tabulære data er CSV[1] [66]. Dette formatet lagrer dataene i ren tekst, hvor hver post er plassert på en egen linje, og hver verdi er separert av et komma, semikolon eller et annet skilletegn. Konverteringen fra rådataene er gjort ved å lage et program som leser inn de to dump-filene sammen med beskrivelsen, hvor beskrivelsen er oversatt til et format som gjør det enkelt for programmet å behandle denne. Videre konverterer programmet dump-filene til CSV og skriver disse filene

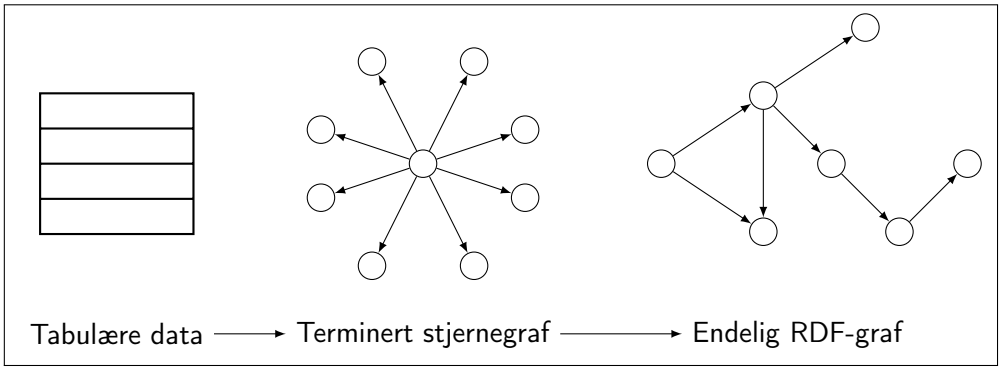
[1] Kommaseparerte verdier (eng: comma separated values)

til disk slik at disse kan brukes videre i konverteringsprosessen. Dette programmet er nærmere beskrevet i seksjon 5.5 på side 84.

Nedenfor vises et uttrekk av samme linje som vist ovenfor, på CSV-format etter konverteringen.

```
"01010750160";"1";"00000000";"TOPSTAD";"TOMAS";
```

4.2 Fra tabulære data til RDF



Figur 4.2: Prosessen med å konvertere tabulære data til RDF.

Når rådataene er blitt konvertert til et tabulært format, kan disse dataene videre konvertes til RDF. Figur 4.2 viser hvordan konverteringen av tabulære data til RDF kan gjøres i to hovedsteg. I det første steget lages en RDF-graf som er en én-til-én-mapping fra de tabulære dataene som vi konverterer fra. I den andre delen raffineres RDF-dataene ytterligere, ved å lage en bedre struktur, samt koble inn eksisterende vokabularer og ontologier.

RDF kan sees på som en abstraksjon av tabulære data, og det eksisterer en enkel algoritme for å konvertere tabulære data til RDF.

- Angi en unik URI til hver rad u_r.
- Angi en unik URI til hver kolonne u_k.
- Trekk ut verdien v fra hver celle på rad r og kolonne k og legg til trippelen (u_r, u_k, v)

Dette gir en såkalt terminert stjernegraf og kalles ofte for den *naive* algoritmen. Denne algoritmen fungerer som et godt utgangspunkt i konverteringen av dataene. Ved å bruke denne algoritmen er det garantert at alle dataelementene som eksisterer i de originale tabulære dataene vil eksistere i RDF-grafen, siden det kun skjer en én-til-én mapping fra de tabulære dataene til RDF. Hvis konverteringen utføres ved å gå direkte fra de originale dataene til en mer kompleks RDF-graf, er det derimot vanskeligere å garantere at dataene har blitt bevart gjennom

konverteringen. Dette begrunnes i at ikke er trivielt å se at alle dataene har blitt med i transformasjonen fra den tabulære strukturen til RDF-grafen.

For å konvertere dataene fra et tabulært format til RDF, trengs det et verktøy som gjør dette, og her har verktøyet XLWrap blitt brukt. XLWrap tar i mot en CSV-fil og en mal som angir hvordan dataene skal konverteres fra CSV til RDF. Dette verktøyet presenteres nærmere i seksjon 5.5.1 på side 87.

4.2.1 Lage gode URI-er

Det å lage gode URI-er for å representere entiteter er essensielt for å oppnå god datakvalitet og mulighet for gjenbruk. For det første er det viktig å ha tenkt gjennom hvordan URI-ene skal se ut, fordi URI-ene blir bedre og mer konsistente hvis det eksisterer en plan for hvordan URI-ene konstrueres. I tillegg er det slik at når data skal distribueres som RDF, bør det alltid være mulig for andre datasett å koble seg opp mot det datasettet som distribueres. Derfor bør oppbyggingen av URI-ene være gjennomtenkt slik at det er liten sannsynlighet for at disse vil forandre seg, og at andre må oppdatere sine lenker til våre data.

Heath og Bizer [37] foreslår følgende tre retningslinjer når URI-er skal konstrueres:

1. Hold deg til navnerom (domener) som du selv har kontroll over.
2. Abstraher URI-ene bort fra implementasjonsdetaljer.
3. Bruk naturlige nøkler i URI-ene

Det første punktet er allerede nevnt i seksjon 3.2.1 på side 20, og betyr at URI-er ikke bør konstrueres med et domene man selv ikke har kontroll over. For det første blir det mindre sannsynlig at én URI beskriver to forskjellige ting, men det gir en også bedre mulighet for å følge et prinsipp fra lenkede data, som sier at det bør brukes HTTP URI-er til å beskrive ressurser, og dermed kunne gi en representasjon av denne ressursen hvis en HTTP-forespørsel blir sendt til den aktuelle URI-en. I tillegg gir det mulighet for å følge prinsippene til *innholdsforhandling*, som betyr at når en klient ønsker å dereferere en URI, kan den angi hva slags type innhold den ønsker å få tilbake fra serveren [17]. Dette gjør at en applikasjon kan sende en forespørsel til demonstratoren å be om en RDF/XML-representasjon av ressursen, mens en bruker kan be om å få en HTML-representasjon som beskriver den samme ressursen.

Punkt nummer to betyr at URI-er ikke bør vise implementasjonsdetaljer og underliggende infrastruktur som servernavn og portnummer. Dette er for å bedre lesbarheten til URI-ene, men også fordi slike URI-er ofte må endres hvis infrastrukturen endres.

Punkt nummer tre sier at for å lage URI-er som er unike, bør det benyttes nøkler i URI-ene som gjør at URI-ene er unike innenfor sin

kontekst. Der det er mulig bør nøkler som er meningsfulle innenfor det aktuelle domene anvendes. En naturlig nøkkel i Folkeregisteret vil være en persons fødselsnummer. Dette nummeret blir brukt til unikt å identifisere en person, og det skal ikke være to personer som har det samme fødselsnummeret.

Basert på de tre prinsippene er det blitt laget URI-er på følgende format for å identifisere en person med fødselsnummer `01010750160`:

```
http://sws.ifi.uio.no/data/dsf/henriwi/person/01010750160
```

Denne URI-en gjør det også mulig å utnytte prinsippene til REST, siden den legger opp til at en liste over alle personer i datasettet returneres ved å «kutte» bort fødselsnummeret. URI-en `http://sws.ifi.uio.no/data/dsf/henriwi/` er benyttet som basis-URI for *data* i datasettet, mens URI-en `http://sws.ifi.uio.no/vocab/dsf/henriwi/` er benyttet som basis-URI til *vokabular*- eller skjemainformasjon.[2].

Når det gjelder predikater sier den naive algoritmen at det skal lages et predikat for hver kolonne i de tabulære dataene. Nedenfor vises et uttrekk av de første kolonnenavnene. Disse er hentet ut fra beskrivelsen til dataene og lagt til som kolonnenavn for hvert dataelement i CSV-filen.

```
"L-FODSELSNR";"L-STATUSKODE"; ... ;"L-SLEKTSNAVN";"L-FORNAVN";
```

Som uttrekket viser er det ofte brukt forkortelser, og en del av navnene er ikke selvforklarende. RDF bør konstrueres på en slik måte at dokumentet er så selvforklarende som mulig, slik at det er enkelt å forstå også for mennesker hva dataene betyr. Som vi skal se senere er det mulig å legge til en bedre beskrivelse i en ontologi, men predikatene bør likevel lages så selvforklarende som mulig. På grunn av dette ble kolonnenavnene oversatt til mere selvforklarende predikatnavn, og forkortelser ble unngått så mye som mulig.

Den naive algoritmen vil lage en terminert stjernegraf hvor alle objektene er literaler, men dette utnytter dårlig datamodellen til RDF. Siden literaler kun kan være objekter og ikke subjekter, vil det ikke være mulig å koble literaler opp mot andre ressurser, og dermed ikke utnytte mulighetene som datamodellen til RDF gir. Ved å gjøre om noen av literalene til ressurser, vil det være mulig for disse å opptre også som subjekter og dermed relatere disse videre til andre ressurser. Dette vil for det første gi mulighet for applikasjoner og brukere å navigere seg gjennom datasettet ved å følge lenkene mellom de ulike entitetene. For det andre blir det mulig for andre datasett å koble seg opp mot vårt datasett ved å angi en lenke til en URI som vi har definert. Selv om folkeregisterdataene ikke skal være åpne data, er det likevel viktig å bruke prinsippene som ligger bak dette paradigmet,

[2]Videre i oppgaven vil `dsf:` brukes som prefiks til URI-en `http://sws.ifi.uio.no/vocab/dsf/henriwi/`, `dsfp:` som prefiks til URI-en `http://sws.ifi.uio.no/data/dsf/henriwi/person/` og `dsfv:` som prefiks til URI-en `http://sws.ifi.uio.no/vocab/dsf/henriwi/dsf#`

siden det på den måten lages et rammeverk som det senere er enkelt å utvide til å omhandle andre datasett.

På bakgrunn av dette ble flere literaler oversatt til URI-er. I rådataene brukes fødselsnummeret til å angi en relasjon mellom to personer. For eksempel vil det i kolonnen `barn` kun ligge fødselsnummeret som identifiserer barnet. Det ble her laget URI-er for å beskrive familierelasjoner (barn, far, mor og ektefelle), ved at en URI med fødselsnummeret som nøkkel ble laget, og en trippel som beskriver at en person har et barn vil se slik ut:

```
dsfp:02088600110 dsfv:harBarn dsfp:01010750160
```

Dette gjør det enklere å navigere seg i RDF-grafen og få opp all informasjon om en persons barn, enn hvis objektet i trippelen over hadde vært en literal. Dette hadde derimot ført til at en ny spørring basert på fødselsnummeret til barnet hadde måttet utføres.

Et annet sted hvor literaler kan konverteres til URI-er er der hvor literalen er en kode. Det er mange koder i rådataene hvor det kun er koden som er lagret i DSF, og hvor verdien er definert i separat dokumentasjon. Det at verdien og den korresponderende koden er relatert i dataene vil være en fordel, i motsetning til i dag hvor det må letes i dokumentasjon og brukerhåndbøker for å finne verdien. Et eksempel på en kode er statuskode, som sier hva slags registreringsstatus personen har. Denne koden er et tall, og verdien til koden er en streng. Det ble her laget en URI på formatet `dsf:registreringsstatus#kode-1`[3] hvor `registreringsstatus` angir kodetypen, `kode` sier at dette er kode (og ikke selve verdien) og `1` er den faktiske koden.

Figur 4.3 på neste side viser et uttrekk av resultatet etter den første konverteringen. Figuren viser informasjon knyttet til en person med fødselsnummer `01010750160`. Legg merke til at det er flere kodetyper som har blitt konvertert til en URI, og at datoene har blitt angitt en type, `xsd:date`. Denne representasjonen er en bedre utnyttelse av datamodellen til RDF enn resultatet av å bruke den rene naive algoritmen, og som vi ser er objektene representert som både URI-er og literaler. RDF-grafen har imidlertid fortsatt ingen struktur, og benytter seg i liten grad av eksisterende vokabularer.

I seksjon 3.2.1 på side 23 om blanke noder, ble det vist at disse kan brukes til blant annet å gruppere tripler som hører sammen, som for eksempel adresseinformasjon. Dette gjør det også mulig å angi en type til denne blanke noden, og dermed gjøre det enklere å hente ut all adresseinformasjon til en person. Siden DSF-dataene blant annet inneholder adresseinformasjon, vil denne teknikken passe bra å anvende på disse dataene. I tillegg er det fire andre steder hvor den samme teknikken kan utføres. Dette er informasjon knyttet til en persons innvandring, innflytting, utvandring og postadresse. Alle disse fem stedene er også gruppert på beskrivelsen vi mottok

[3] Skrivemåten `dsf:registreringsstatus#kode-1` er ikke gyldig Turtle, men er brukt for å gjøre det mer lesbart.

```
dsfp:01010750160
    a       dsfv:Person ;
    dsfv:harFar dsfp:02088600110 ;
    dsfv:flyttedatoForAdresse "2007-01-01"^^xsd:date ;
    dsfv:gateGaard "00019" ;
    dsfv:harAdressenavn "BOKS 6300, ETTERSTAD" ;
    dsfv:harAdressetype dsf:adressetype#kode-M ;
    dsfv:harForeldreansvar dsf:foreldreansvar#kode-D ;
    dsfv:harFornavn "TOMAS" ;
    dsfv:harPostnummer "0603" ;
    dsfv:harSlektsnavn "TOPSTAD" ;
    dsfv:harStatuskode dsf:registreringsstatus#kode-1 ;
    dsfv:husBruk "0019" ;
    dsfv:harMor dsfp:03088600230 ;
    dsfv:regdatoForAdresse "2007-01-01"^^xsd:date ;
```

Figur 4.3: Resultatet av første konvertering.

fra SKD, og det er enda et argument for hvorfor det er hensiktsmessig å videreføre denne grupperingen.

4.2.2 Konservativ transformasjon ved hjelp av CONSTRUCT-spørringer

Det finnes flere muligheter for å raffinere dataene videre for blant annet å få en bedre struktur på RDF-grafen. Uavhengig av hvilken metode som blir valgt, bør det være mulig å gå tilbake til de forskjellige versjonene av RDF-grafen for å se nøyaktig hvilke steg som er utført underveis i konverteringen. I tillegg er det viktig å kunne garantere at dataene blir bevart gjennom hele konverteringsprosessen. Det vil si at all informasjon som eksisterer i rådataene skal eksistere i den ferdige RDF-grafen, og at det ikke er noe informasjon som har blitt borte underveis i konverteringen. På den andre siden er det også viktig at det ikke har sneket seg inn feilaktig tilleggsinformasjon.

Et verktøy som XLWrap anvender en mal som brukes til å konvertere de tabulære dataene til RDF. Hvis et slikt verktøy er benyttet, er et alternativ å oppdatere malen som den naive algoritmen baserer seg på, og deretter utføre en ny konvertering. Problemet med dette er at det vil overskrive den originale RDF-grafen, historikken går tapt, og dermed muligheten for å gå tilbake til de forskjellige versjonene. Dette kan løses ved å lage flere separate maler, men det er ikke enkelt å se de forskjellige stegene, fordi hver enkelt mal må sammenlignes manuelt med den forrige for å se hva forskjellene er. Det gir heller ingen garanti for at dataene har blitt bevart gjennom alle stegene i konverteringen. Det er selvfølgelig mulig å manuelt sjekke dette, men på store datasett vil dette i praksis ikke være mulig, og det vil være stor sannsynlighet for feil.

Bizer og Schultz har laget rammeverket *R2R* [20], som kan brukes for å

mappe data fra et vokabular til et annet. Det fungerer ved at mappinger mellom termer fra et kildevokabular til termer i et målvokabular angis, og at disse mappingene automatisk blir kombinert for å konvertere kildedataene til måldataene. Fordelen med R2R er at vi får en mappingfil som viser hvordan termer i kildevokabularet mappes til termer i målvokabularet, og dette gjør det mulig å gå tilbake og enkelt se hvordan transformeringen har foregått, samt at mappingfilen vil fungere som dokumentasjon. Ulempen er at det ikke gir noen garanti for at dataene blir bevart gjennom transformasjonen, og det er ikke mulig å vite om noe av informasjonen har blitt forvrengt i konverteringen fra kildevokabularet til målvokabularet.

En annen mulighet er å benytte seg av SPARQL `CONSTRUCT`-spørringer. Som vist brukes `CONSTRUCT`-spørringer til å returnere en ny RDF-graf basert på eksisterende data. Siden `CONSTRUCT`-spørringer kan brukes til å jobbe med kun en liten av del av RDF-grafen av gangen, kan denne spørringstypen brukes til å ta små steg, og samtidig ha full oversikt over hvilke endringer som er gjort. Det kan lages en rekke `CONSTRUCT`-spørringer, og deretter utføre disse over RDF-datene, en etter en. Det er da enkelt å gå tilbake i ettertid og se på de forskjellige `CONSTRUCT`-spørringene, og dermed få en oversikt og dokumentasjon over hva som er gjort. I tillegg er `CONSTRUCT`-spørringer en åpen og tilgjengelig W3C-standard, som fører til at man ikke binder seg til en spesiell teknologi eller et rammeverk for å utføre konverteringen.

Selv om det gir bedre kontroll over de forskjellige stegene når `CONSTRUCT`-spørringer anvendes på denne måten, er det likevel ikke problemfritt. Hvordan vet vi for eksempel at vi beholder alle dataene som originalt er til stede i rådataene? Det vil si, hvordan kan vi garantere at RDF-grafen inneholder all informasjon som eksisterer i det originale datasettet? Problemet er når vi går fra en måte å representere dataene på til en annen, hvordan kan vi garantere at all informasjon blir holdt intakt?

Stolpe og Skjæveland foreslår en metode i artikkelen *Preserving Information Content in RDF using Bounded Homomorphisms* [70], hvor de viser at man ved å bruke en strukturbevarende funksjon for å oversette en RDF-graf inn en annen, kan garantere at dataene som skrives om fra en graf til en annen ikke blir forvrengt. En slik strukturbevarende funksjon kalles for en *p-map*, og er en homomorfi h på ressurser i en kildegraf til ressurser i en målgraf som bevarer strukturen til triplene i kilden. Det vil si hvis trippelen `(a p b)` er i kilden, så må trippelen $(h$`(a)` h`(p)` h`(b))` eksistere i målet. Dette betyr hvis en trippel eksisterer i kilden, må trippelen i sin oversatte form eksistere i målet. Videre er homomorfien identitet på subjekter og objekter, fordi det antas at dataelementene i en RDF-graf er i subjektene og objektene, og at disse skal bevares like gjennom hele transformasjonen, og at det derfor er egenskapene til dataelementene (predikatene) som det er ønskelig å forandre på.

Videre definerer Stolpe og Skjæveland tre *tilbakebetingelser* som relaterer enkelte tripler i målet tilbake til tripler i kilden. Disse betingelsene sikrer at data i målet som er et resultat av transformasjonen av kildedataene ikke

forstyrres av annen data i målet, og de forskjellige tilbakebetingelsene sier noe om hvor mye mer data (i denne sammenheng subjekter og objekter) en kan ha for predikater som kildepredikater mappes til. Et eksempel som viser hvorfor dette er nødvendig er å anta at vi har en kildegraf som inneholder trippelen (`dsfp:Ola dsfv:harMor dsfp:Kari`) og en målgraf som inneholder trippelen (`dsfp:Kari dsfv:erBarnTil dsfp:Ola`). Anta videre at homomorfien er definert til at predikatet `dsfv:harMor` skrives om til predikatet `dsfv:erBarnTil`. Hvis vi nå skriver om kildegrafen inn i målgrafen vil målgrafen bestå av følgende to tripler, noe som ikke gir mening, siden Kari er barn til Ola, samtidig som Ola er barn til Kari.

```
dsfp:Kari dsfv:erBarnTil dsfp:Ola .
dsfp:Ola  dsfv:erBarnTil dsfp:Kari .
```

For å vise hvordan de forskjellige tilbakebetingelsene stiller ulike krav til transformasjonen, antar vi at vi har følgende kilde:

```
dsfp:Ola dsfv:barns_fodselsnr dsfp:Lisa .
```

La videre homomorfien være definert til at predikatet `dsfv:barns_fodselsnr` skrives om til predikatet `dsfv:harBarn`. Til slutt lar vi følgende tripler uttrykke mulige målgrafer, hvor vi vil referere til linjeintervaller for å vise hvilke tripler vi anser som målgrafen.

```
1: dsfp:Ola   dsfv:harBarn dsfp:Lisa .
2: dsfp:Lars  dsfv:harBarn dsfp:Mari .
3: dsfp:Ola   dsfv:harBarn dsfp:Trude .
4: dsfp:Lisa  dsfv:harBarn dsfp:Ola .
5: dsfp:Ola   rdf:type     dsfv:Person .
```

De tre tilbakebetingelsene er definert som følger:

p1) Hvis trippelen (`a` h`(p) b`) er i målet, så må trippelen (`a p b`) være i kilden. Dette betyr at hvis det eksisterer en trippel i målet som bruker et predikat som et kildepredikat blir mappet til, må en korresponderende trippel eksistere i kilden som bruker kildepredikatet. Det fører til at det ikke er lov å legge til noen nye dataelementer (subjekter eller objekter) til predikater som kildepredikater mappes til for at en målgraf skal tilfredsstille p1. I dette tilfellet vil det si at det ikke er lov å relatere elementer med `dsfv:harBarn` i målet, hvis disse elementene ikke er relatert av `dsfv:barns_fodselsnr` i kilden.

    Målgrafen bestående kun av linje `1` vil tilfredstille p1, fordi det eneste som er skjedd i transformasjonen er at predikatet har blitt omskrevet. En målgraf bestående av linje `1` og `2`, vil derimot ikke tilfredsstille p1, fordi subjektet og objektet i trippelen (`dsfp:Lars dsfv:harBarn dsfp:Mari`) ikke er relatert av predikatet `dsfv:barns_fodselsnr` i kilden.

p2) Hvis trippelen (h`(a)` h`(p) b`) eller (`a` h`(p)` h`(b)`) er i målet, så må trippelen (`a p b`) være i kilden. For at en målgraf skal tilfredsstille

tilbakebetingelsen p2, kan nye dataelementer legges til predikater som kildepredikater blir mappet til, så lenge disse elementene relateres til kun nye dataelementer. Dette betyr at nye tripler som bruker predikater som kildepredikater blir mappet til er tillatt, så lenge subjektene og objektene i disse triplene også er nye, det vil si at de ikke finnes i kilden. Målgrafen bestående av linje `1` og `2` vil derfor tilfredsstille p2 fordi hverken `dsfp:Lars` eller `dsfp:Mari` eksisterer i kilden, mens målgrafen bestående av linjene `1–3` ikke vil tilfredsstille p2, fordi subjektet `dsfp:Ola` eksisterer i kilden, mens objektet `dsfp:Trude` ikke gjør det. Vi vil dermed ha en trippel i målet som bruker et predikat som et kildepredikat blir mappet til, men hvor subjektet ikke er et nytt dataelement, men eksisterer i kilden fra før.

p3) Hvis trippelen (h`(a)` h`(p)` h`(b)`) er i målet, så må trippelen (`a p b`) være i kilden. For at en målgraf skal tilfredsstille tilbakebetingelsen p3, er det mulig å relatere gamle og nye elementer med predikater som kildepredikater blir mappet til, så lenge de gamle elementene er i samme posisjon (subjekt forblir subjekt og objekt forblir objekt) i målet som i kilden. Målgrafen bestående av linje `1–3` vil tilfredsstille tilbakebetingelsen p3, fordi ressursen `dsfp:Ola` er subjekt i både kilden og målet, mens `dsfp:Trude` er en ny ressurs som ikke eksisterer i kilden. En målgraf bestående av linje `1–4` vil derimot ikke tilfredsstille p3, fordi ressursen `dsfp:Ola` opptrer som subjekt i kilden, men både som subjekt og objekt i målet.

Det er også viktig å nevne at det kun er predikater som er i verdiområdet til homomorfien som gir begrensninger. Målgrafen bestående av linje `5` vil derfor tilfredsstille både p1, p2 og p3 fordi `rdf:type` ikke er et predikat som et kildepredikat blir mappet til. I tillegg vil tilbakebetingelsen p1 medføre tilbakebetingelsen p2, som igjen medfører tilbakebetingelsen p3. Det vil si at ethvert p-map som tilfredsstiller p1 også vil tilfredsstille p2 og p3, og ethvert p-map som tilfredsstiller p2, vil tilfredsstille p3.

I tillegg til å kunne mappe predikater til andre predikater, er teorien utvidet til å gjelde mappinger fra kjeder av predikater til andre kjeder av predikater. Det vil si at kjeden `dsfv:barns_fodselsnr, dsfv:navn` for eksempel kan mappes til kjeden `dsfv:harBarn, dsfv:navn`. Teorien er også utvidet til å gjelde SPARQL `CONSTRUCT`-spørringer, og det vises at dersom det finnes et p-map mellom grafmønsteret i `WHERE`-blokken og grafmønsteret i `CONSTRUCT`-blokken, hvor variabler kun er tillatt på subjekt- og objektposisjon, og variabler er identitet, så vil samme p-map eksistere mellom grafen som `WHERE`-blokken matcher og resultatet av `CONSTRUCT`-spørringen. Det vil si at hvis vi finner et p-map mellom grafmønsteret i `WHERE`-blokken og grafmønsteret i `CONSTRUCT`-blokken er vi garantert at `CONSTRUCT`-spørringene som utføres over kildegrafen ikke forvrenger informasjonen under omskrivingen til målgrafen.

Ved hjelp av denne teorien er vi garantert at data som omskrives ved hjelp av `CONSTRUCT`-spørringer ikke blir forvrengt under transformasjonen. Men

hvordan kan vi garantere at vi får med oss alle dataene fra RDF-grafen produsert av den naive algoritmen over i den endelige RDF-grafen? Siden det kun er en liten del av RDF-grafen som blir påvirket av `CONSTRUCT`-spørringene, kan vi ikke anvende denne teorien på resten av grafen. Måten vi har løst dette på, er at istedenfor å ha en kildegraf A som vi henter ut data fra og stadig utvider en målgraf B med resultatene av `CONSTRUCT`-spørringene, jobber vi kun på kildegraf A. Det vil si at vi utfører en `CONSTRUCT`-spørring hvor resultatet er en RDF-graf, og så skrives denne `CONSTRUCT`-spørringen om til en `DELETE`-spørring hvor alle dataene som er blitt matchet av `WHERE`-blokken i `CONSTRUCT`-spørringen fjernes. Deretter legger vi til triplene i RDF-grafen vi fikk fra `CONSTRUCT`-spørringen til kildegraf A. Et problem som da dukker opp er at de konverterte bitene fra disse spørringene kan forkludres av de eksisterende dataene i A, og omvendt. En måte å sjekke dette på er å sjekke hele grafen som er et resultat av den naive algoritmen opp mot den ferdige grafen, som er et resultat etter å ha utført `CONSTRUCT`-spørringene. Verktøyet MapperDan[4] kan sjekke dette, men dette verktøyet har på det nåværende tidspunktet ikke støtte for å sjekke den type spørringer som vi utfører, selv om de støttes i teorien. Men det er enkelt å se at `CONSTRUCT`-spørringene vi anvender (presenteres nedenfor) ikke forkludrer eksisterende data fordi enhver transformasjon av et predikat gjøres for alle tripler med dette predikatet, og alle predikater transformeres til en kjede som er unik for denne transformasjonen og som er ny for målgrafen.

De neste avsnittene viser de forskjellige type `CONSTRUCT`-spørringer som har blitt brukt i transformeringen av DSF-dataene, hva slags p-map som er blitt benyttet, og hvilken tilbakebetingelse som er oppfylt. Her vises kun `CONSTRUCT`-spørringen for adressedelen av RDF-grafen, mens vedlegg B viser spørringene som er utført for innvandrings-, innflyttings-, utvandrings- og postadressedelen av RDF-grafen. I tillegg viser vedlegget dataene både før og etter at den aktuelle spørringen har blitt utført.

Adresse

`CONSTRUCT`-spørringen som brukes til å transformere adressedelen av RDF-grafen, vises i figur 4.4 på neste side. Her ser vi hvordan det blir opprettet en blank node for å angi en persons adresse, og at denne blir angitt en type, `dsfv:AktuellAdresse`. Den blanke noden brukes til å gruppere alle adressepåstander, og gjør det enklere å hente ut data om en persons adresse. Legg også merke til at det brukes `OPTIONAL` i `WHERE`-delen av spørringen. Dette er fordi det viser seg at det er mange personer som kun har noen av feltene fylt ut, og for å sørge for at alle adresser blir raffinert, må `OPTIONAL` være med.

Vi angir følgende p-map:

`dsfv:kommunenummer` $\mapsto$ `dsfv:harAktuellAdresse, dsfv:kommunenummer,`

[4] `http://sws.ifi.uio.no/MapperDan/`

```
CONSTRUCT { ?person dsfv:harAktuellAdresse [
              rdf:type dsfv:AktuellAdresse ;
              dsfv:kommunenummer ?kommunenummer ;
              dsfv:regdatoForAdresse ?regdato ;
              dsfv:flyttedatoForAdresse ?flyttedato ;
              dsfv:gateGaard ?gateGaard ;
              dsfv:husBruk ?husBruk ;
              dsfv:bokstavFestenr ?bokstavFestenr ;
              dsfv:undernr ?undernr ;
              dsfv:adressenavn ?adressenavn ;
              dsfv:harAdressetype ?adressetype ;
              dsfv:tilleggsadresse ?tilleggsadresse ;
              dsfv:postnummer ?postnummer ;
              dsfv:valgkrets ?valgkrets ;
            ]
}
WHERE {
  OPTIONAL { ?person dsfv:kommunenummer ?kommunenummer }
  OPTIONAL { ?person dsfv:regdatoForAdresse ?regdato }
  OPTIONAL { ?person dsfv:flyttedatoForAdresse ?flyttedato }
  OPTIONAL { ?person dsfv:gateGaard ?gateGaard }
  OPTIONAL { ?person dsfv:husBruk ?husBruk }
  OPTIONAL { ?person dsfv:bokstavFestenr ?bokstavFestenr }
  OPTIONAL { ?person dsfv:undernr ?undernr }
  OPTIONAL { ?person dsfv:adressenavn ?adressenavn }
  OPTIONAL { ?person dsfv:harAdressetype ?adressetype }
  OPTIONAL { ?person dsfv:tilleggsadresse ?tilleggsadresse }
  OPTIONAL { ?person dsfv:postnummer ?postnummer }
  OPTIONAL { ?person dsfv:valgkrets ?valgkrets }
}
```

Figur 4.4: `CONSTRUCT`-spørringen som brukes til å raffinere adressedelen.

```
dsfv:regdatoForAdresse ↦ dsfv:harAktuellAdresse, dsfv:regdatoForAdresse,
dsfv:flyttedatoForAdresse ↦ dsfv:harAktuellAdresse, dsfv:flyttedatoForAdresse,
dsfv:gateGaard ↦ dsfv:harAktuellAdresse, dsfv:gateGaard,
dsfv:husBruk ↦ dsfv:harAktuellAdresse, dsfv:husBruk,
dsfv:bokstavFestenr ↦ dsfv:harAktuellAdresse, dsfv:bokstavFestenr,
dsfv:undernr ↦ dsfv:harAktuellAdresse, dsfv:undernr,
dsfv:adressenavn ↦ dsfv:harAktuellAdresse, dsfv:adressenavn,
dsfv:adressetype ↦ dsfv:harAktuellAdresse, dsfv:adressetype,
dsfv:tilleggsadresse ↦ dsfv:harAktuellAdresse, dsfv:tilleggsadresse,
dsfv:postnummer ↦ dsfv:harAktuellAdresse, dsfv:postnummer,
dsfv:valgkrets ↦ dsfv:harAktuellAdresse, dsfv:valgkrets
```

Mappingen tilfredsstiller tilbakebetingelsen p1. Dette er fordi alle elementer som er i målgrafen, også er relatert i kildegrafen, og det eneste som gjøres er å omskrive predikatene. I tillegg har den blanke noden fått en type, og dette er også tillatt, siden `rdf:type` ikke er i verdiområdet til mappingen.

Far, Mor og Ektefelle

Rådataene relaterer en person til sin far med et felt som inneholder fødselsnummeret til personens far. I tillegg eksisterer det et felt som inneholder navnet til faren og et felt som inneholder statsborgerskapet. Grunnen til dette er at det skal være mulig å hente ut informasjon om farens navn og statsborgerskap direkte fra en person uten å måtte stille en ny spørring som henter ut informasjon om faren. En person er relatert til sin mor og ektefelle på samme måte. Etter at den naive algoritmen er utført vil RDF-grafen som viser en persons relasjon til sin far se slik ut:

```
dsfp:01014600139
  dsfv:harFar dsf:20056507125 ;
  dsfv:farsNavn "HANSEN ARNE" ;
  dsfv:harFarsStatsborgerskap dsf:land#kode-103 .
```

Et problem med dette er at det vil ligge duplikatinformasjon i datasettet. Det vil si at farens navn og statsborgerskap vil ligge lagret både knyttet til faren og til personen som har vedkommende som far. Datamodellen til RDF gjør det mulig å hente ut informasjon om farens navn, statsborgerskap og all annen informasjon knyttet til faren, ved å navigere seg fra en person til en persons far og derfra videre til den ønskede egenskapen. Følgende SPARQL-spørring viser hvordan dette kan gjøres:

```
SELECT ?navn WHERE {
  dsfp:01014600139 dsfv:harHar ?far .
  ?far dsfv:navn ?navn .
}
```

Det er derfor ønskelig å fjerne relasjonene mellom en person og farens navn og statsborgerskap, og kun beholde relasjonen til ressursen som beskriver faren. Dette gjelder selvfølgelig kun hvis vi vet farens fødselsnummer. Hvis

den eneste informasjonen som finnes er farens navn eller statsborgerskap, vil vi beholde dette. Følgende CONSTRUCT-spørring vil utføre dette:

```
CONSTRUCT { ?person dsfv:harFar ?far }
WHERE {
    ?person dsfv:harFar ?far .
    OPTIONAL { ?person dsfv:farsNavn ?navn }
    OPTIONAL { ?person dsfv:harFarsStatsborgerskap ?statsb }
}
```

Problemet med denne spørringen er at det ikke eksisterer et gyldig p-map fra WHERE-blokken til CONSTRUCT-blokken. Dette er fordi teorien ikke sier noe om hvordan informasjon kan *fjernes* på en konservativ måte. Et forbedret forsøk er å endre CONSTRUCT-blokken til følgende:

```
CONSTRUCT {
  ?person dsfv:harFar ?far .
  ?far    dsfv:navn ?navn ;
          dsfv:harStatsborgerskap ?statsb .
}
```

Med spørringen over kan vi prøve å angi følgende p-map:

```
dsfv:harFar ↦ dsfv:harFar,
dsfv:farsNavn ↦ dsfv:harFar, dsfv:navn,
dsfv:harFarsStatsborgerskap ↦ dsfv:harFar, dsfv:harFarsStatsborgerskap
```

Et krav til teorien er at variabler kun kan opptre i subjekt- og objektposisjon og at variabler er identitet, det vil si at variabler i CONSTRUCT-blokken må opptre på samme plass i WHERE-blokken. Som vi ser opptrer variablene ?far som objekt i WHERE-blokken, mens den opptrer som både objekt og subjekt i CONSTRUCT-blokken, noe som ikke er lovlig. Dette fører til at vi har et ugyldig p-map, og at det dermed ikke er mulig å garantere ved hjelp av denne teorien at spørringen ikke forvrenger informasjonen.

Selv om teorien ikke støtter denne formen for transformasjon, har vi likevel valgt å utføre CONSTRUCT-spørringene vist i figur 4.5, 4.6 på neste side og 4.7 på neste side. Dette er fordi det er enkelt å se at transformeringen ikke forvrenger informasjonen, og at målgrafen vil utnytte datamodellen til RDF bedre, og det vil føre til at det ikke ligger duplikatinformasjon i datasettet.

```
CONSTRUCT { ?person dsfv:harFar ?far }
WHERE {
    ?person dsfv:harFar ?far .
    OPTIONAL { ?person dsfv:farsNavn ?navn }
    OPTIONAL { ?person dsfv:harFarsStatsborgerskap ?statsb }
}
```

Figur 4.5: CONSTRUCT-spørringen som brukes til å raffinere en persons far.

```
CONSTRUCT { ?person dsfv:harMor ?far }
WHERE {
    ?person dsfv:harMor ?far .
    OPTIONAL { ?person dsfv:morsNavn ?navn }
    OPTIONAL { ?person dsfv:harMorsStatsborgerskap ?statsb }
}
```

Figur 4.6: CONSTRUCT-spørringen som brukes til å raffinere en persons mor.

```
CONSTRUCT { ?person dsfv:harEktefelle ?ektefelle }
WHERE {
    ?person dsfv:harEktefelle ?ektefelle ;
    OPTIONAL { ?person dsfv:ektefellePartnerNavn ?navn }
    OPTIONAL { ?person
                  dsfv:harEktefellePartnerStatsborgerskap ?statsb }
}
```

Figur 4.7: CONSTRUCT-spørringen som brukes til å raffinere en persons ektefelle.

4.3 Utviklingen av ontologiene

Etter å ha konvertert rådataene til RDF gjennom flere iterasjoner, har dataene fått en god struktur. Men RDF-dataene vil ikke kunne bli utnyttet til det fulle før disse knyttes opp mot et vokabular eller en ontologi. Selv om dataene nå kan distribueres som RDF, vil det fortsatt være vanskelig å forstå meningen med dataene, både for et menneske og en maskin, fordi bakgrunnsinformasjonen som er nødvendig ikke er definert. Det vil også være vanskelig å skrive spørringer mot dataene fordi strukturen til dataene ikke er enkelt tilgjengeliggjort, og resonnering vil ikke gi den store verdien fordi det ikke eksisterer bakgrunnsinformasjon til dette aktuelle domenet.

Det å utvikle en ontologi, såkalt *ontology engineering* er et område som det ikke er forsket veldig mye på, men boken *Foundations of Semantic Web Technologies* [41] foreslår i kapittel 8 en del metodikker og teknikker for hvordan en ontologi bør utvikles. Det å utvikle en ontologi er, som annen programvareutvikling, ingen eksakt vitenskap og krever kompetanse og erfaring. Det er derfor viktig å ha noen retningslinjer for hvordan ontologien skal utvikles og ha en plan for arbeidet før utviklingen starter. Boken foreslår tre hovedsteg: analyse av kravene, selve utviklingen og til slutt kvalitetssikring av ontologien.

Under analysen bør det stilles spørsmål som gjør at det skapes en forståelse for hvilket domene som skal modelleres, og hva slags oppgaver ontologien skal være med på å utføre. Det å modellere et domene med en komplett beskrivelse av alle egenskapene til dette domenet vil for det første være en stor utfordring å modellere, og for det andre vil det sannsynligvis føre til en modell med høy kompleksitet. Det er derfor viktig å definere hva modellen skal benyttes til, om den for eksempel skal brukes til å navigere

seg gjennom for å utforske domene og ha lite data knyttet til seg, eller om det er essensielt at resonnering kan utføres raskt og at ontologien skal benyttes sammen med et stort datasett.

Når kravene til ontologien er etablert kan selve utviklingen starte. Det å utvikle en ontologi kan sees på som å overføre kunnskap til et maskinleselig format [41, side 309], og denne kunnskapen kan ligge lagret i mange forskjellige kilder. Dette kan være i dokumenter som systemdokumentasjon, databaseskjemaer og brukerhåndbøker, men også hos mennesker som domeneeksperter og brukere av eksisterende systemer. Disse kildene kan overlappe hverandre og være motstridende, og en utfordring under utviklingen er derfor å klare å kombinere informasjon fra flere forskjellige kilder til én ontologi.

For å vurdere en ontologi, kan spørsmål som «Oppfyller ontologien det tiltenkte bruksområdet?» stilles [41, side 317]. Det vil si at det er mulig å utlede den informasjonen som det er ønskelig å utlede, og ikke minst at det å anvende ontologien sammen med resten av en applikasjon gjør at brukeren får utført den tiltenkte oppgaven. I tillegg til denne «åpenbare» vurderingen, bør ontologien gåes gjennom for å sjekke at det ikke har oppstått logiske selvmotsigelser, og at ontologien ikke fører til at informasjon som er feil kan bli utledet.

Resten av dette kapittelet vil vise hvordan ontologiene som brukes i demonstratoren er utviklet. Siden identifiseringsproblemet består av to kilder, DSF og SED, og disse skal integreres, har vi valgt å utvikle flere ontologier. Først vises utviklingen og resultatet av DSF-ontologien før det samme gjøres med SED-ontologien. Deretter vises det hvordan SED-ontologien har blitt integrert opp mot DSF-ontologien.

4.4 Ontologi for Det sentrale folkeregister

4.4.1 Analyse av kravene til ontologien

Ontologien for DSF kan sees på som hovedontologien av de to ontologiene som er utviklet, og det er to punkter som ontologien skal brukes til. Det ene er å fungere som et spørregrensesnitt mot dataene, det vil si at ontologien brukes til å formulere SPARQL-spørringer over DSF-dataene. Det andre er å bruke resonnering generelt til å utvide datasettet og sjekke om modellen er konsistent, og spesielt til å fastslå om personer fra SED og DSF er like. I henhold til et sett med kriterier som er definert i ontologien, skal en resonnerer kunne legge til en trippel som sier at en person fra SED-datasettet er lik en annen person i DSF-datasettet.

Basert på disse to bruksområdene er det derfor viktig at ontologien balanserer uttrykkskraft og kompleksitet, det vil si at ontologien må være komplett nok til å kunne spørre over alle deler av dataene, samtidig som den ikke er unødig kompleks. Som vi skal se senere er ontologien som er

utviklet ganske mager og langt fra komplett. I tillegg er mengden med data som vi har fått tilgang til, og som modellen derfor skal brukes sammen med, liten. Dette fører til at kompleksitet ikke er et viktig og relevant krav for ontologien som skal utvikles til bruk i vår demonstrator. Dette er også grunnen til at ontologien ikke følger en bestemt profil, fordi det ikke er nødvendig å begrense seg til et subsett med utsagn, men det er bedre å ha mulighet til å bruke alle utsagn som eksisterer for å vise mulighetene som OWL gir.

4.4.2 Kilder

For å utvikle ontologien har vi anvendt flere forskjellige kilder. Kildene er viktige for å klare å få en forståelse for domenet som skal modelleres, og det er en utfordrende og krevende jobb å omskrive tekstlige beskrivelser og dokumenter til formell logikk. I utviklingen av DSF-ontologien er følgende kilder benyttet.

Beskrivelse av dumpen med folkeregisterdata

Beskrivelsen av dumpen med folkeregisterdata ble brukt til konverteringen fra rådataene til RDF og har også blitt brukt mye i utviklingen av ontologien. Som nevnt er dette et word-dokument, og et lite utsnitt fra denne ble vist i figur 4.1 på side 48.

Beskrivelsen er en komplett syntaktisk beskrivelse av dataene på et lavt nivå. Med lavt nivå menes data som er atomiske og som ikke kan deles opp i flere mindre deler. Ved hjelp av denne beskrivelsen har vi kunnet lage alle rollene til ontologien for den første delen av konverteringen hvor en én-til-én-mapping utføres. I tillegg har alle datofelt en kommentar som sier hva slags formatering datoene har i rådataene, som igjen gjør det mulig å få riktig format på datoene i de endelige RDF-dataene.

En stor ulempe med beskrivelsen er den ikke inneholder noen form for semantikk eller metainformasjon knyttet til de forskjellige feltene, som for eksempel datatype eller hva feltet faktisk betyr. Det er dermed ikke mulig å vite hvordan verdien til et felt ser ut uten å studere dataene nærmere. For eksempel ser vi at feltet `L-STATUSKODE` består av ett tegn, men dette sier ingenting om semantikken til feltet og hva definisjonen og betydningen av en statuskode er. I tillegg er det vanskelig å vite hva de forskjellige feltene betyr, for eksempel er det ikke enkelt å forstå hva feltet `L-TK-NR` betyr hvis man ikke vet det, eller har tilgang til en annen beskrivelse. Som nevnt tidligere består rådataene av en rekke koder, men beskrivelsen sier ingenting om hva verdien til disse kodene er.

Beskrivelsen gir muligheter for å lage en komplett modell med tanke på alle rollene, men det er ingen beskrivelser av klasser og konsepter som kan modelleres. Basert på dette punktet og de nevnt i forrige avsnitt, fører dette

til at beskrivelsen i seg selv ikke er nok til å lage en god ontologi, men at det er nødvendig å kombinere denne informasjonen med informasjon fra andre kilder.

Databaseskjema

Databaseskjemaet er beskrevet i et stort word-dokument som inneholder informasjon om alle tabeller som har med Folkeregisteret å gjøre. Dette dokumentet har vi mottatt fra SKD, og mye av innholdet er ikke relevant for vårt arbeid fordi vi ikke jobber med hele DSF, men kun en anonymisert eksempelversjon av det uttrekket som blir sendt til distributører. Dumpen med data som distribueres genereres ut i fra en tabell som inneholder siste døgns endringer i DSF, og denne tabellen er beskrevet i dette dokumentet.

Til forskjell fra beskrivelsen av dumpen med folkeregisterdata så er ikke databaseskjemaet helt korrekt for de dataene vi har mottatt, det vil si at det er noen felt som eksisterer i dumpen som vi ikke har klart å identifisere i databaseskjemaet og motsatt. Til gjengjeld er det en kommentar til de fleste felter, hvor denne kommentaren inneholder metainformasjon om de forskjellige feltene, og har gitt god kunnskap om hva en del felter inneholder som ikke har vært mulig å forstå kun ut i fra navnet. For eksempel står det i beskrivelsen til feltet `TK_NR` at dette er `Trygdekontornummer`. Som nevnt inneholder ikke beskrivelsen noen kodeverdier, men det gjør databaseskjemaet, og det har derfor vært mulig å modellere forholdet mellom kode og verdi direkte i modellen (mer om dette i seksjon 4.4.3 på neste side)

Brukerhåndbok

Brukerhåndboken er en manual for sluttbruker som skal oppdatere Folkeregisteret, og har dermed først og fremst et fokus på systembruk og ikke struktur på dataene. Det er derfor mye informasjon som ikke er relevant, og det har vært en stor utfordring å filtrere ut den informasjonen som er interessant for vårt bruk. Boken inneholder også lister med koder og deres verdier, og har vært nyttig sammen med databaseskjemaet for å finne ut hvilke verdier de forskjellige kodene har.

Brukerhåndboken har også til en viss grad blitt benyttet til å finne metainformasjon og forklaring til hva de forskjellige feltene betyr. Videre har boken blitt brukt noe til å identifisere klasser og konsepter som kan modelleres — som person, adresse, innvandring og utvandring.

Oppsummering

Både beskrivelsen av rådataene og databaseskjemaene er mer eller mindre korrekte om syntaksen, men inneholder lite eller ingen metainformasjon

knyttet til feltene. Den største mangelen er imidlertid at det ikke er noen beskrivelse av konsepter og klasser som kan modelleres. En naturlig klasse i dette domenet er selvfølgelig `Person`, men det er vanskelig å vite hva en `Person` er. Har alle personer et fornavn og et etternavn og en statuskode, eller er det flere eller færre felter alle personer skal ha, og hvilke andre klasser er naturlige å ha med i ontologien? Dette har vært en stor utfordring, da det ikke finnes noen klar definisjon på hva slike konsepter er i Folkeregisteret i de kildene vi har brukt.

Kildene som er presentert og benyttet under arbeidet viser at de ikke er tilstrekkelig for å klare å lage en komplett modell over DSF. For at dette skulle vært mulig, hadde det vært nødvendig med en modell som et E/R-diagram eller UML-modell som kan overføres mer eller mindre direkte til OWL. En annen mulighet er å ha tilgang til domeneeksperter, noe vi dessverre ikke har hatt. Domeneeksperter er gjerne personer som til daglig jobber med domenet som skal modelleres og har stor oversikt på området, og kan fungere som et supplement til tekstlige kilder, og oppklare uklarheter eller mangler i disse dokumentene. For eksempel hvilke krav må være oppfylt for alle personer i DSF, det vil si hvilke felter er obligatoriske for alle personer, hvilke er valgfrie også videre. Med de kildene som vi har hatt tilgjengelig har arbeidet med ontologien vært vanskelig, og mye er basert på sterke antagelser om hva DSF inneholder. Dette er grunnen til at ontologien er forholdsvis mager og langt fra komplett.

Videre er dette også grunnen til at det ikke har vært mulig å kvalitetssikre modellen på samme måte som kravene til ontologiutvikling sier at vi burde gjort. Vi har selv kunnet til en viss grad sjekke at modellen ikke inneholder noen logiske feil eller selvmotsigelser, samt anvendt resonnering for å sjekke at modellen ikke inneholder inkonsistens. Men når det kommer til spørsmål om ontologien oppfyller det tiltenkte bruksområdet, er dette en kvalitetssikring som må gjøres av personer som har stor kunnskap om domenet, samt sluttbrukere som skal anvende ontologien og systemet som ontologien er en del av. Dette er også vanlig i generell programvareutvikling hvor det er sluttbrukere som er ansvarlig for å verifisere korrektheten til en akseptansetest av et system [77].

4.4.3 Ontologien

Denne seksjonen presenterer deler av selve ontologien. I de neste avsnittene vil henholdsvis klassene, objektrollene og datarollene bli presentert. Vi vil her trekke frem det vi mener er de viktigste klassene og rollene, og det henvises til vedlegg C hvor både DSF- og SED-ontologiene er vist i sin helhet.

Klasser

Siden ontologien skal brukes til å representere og distribuere persondata, er det naturlig at en klasse er `:Person`[5]. Videre er det definert fem subklasser av `:Person`, som representerer spesialiseringer av personer. Dette er:

```
:Barn
:Ektefelle
:Forelder
  :Far
  :Mor
```

Klassene `:Mor` og `:Far` er satt disjunkte fordi et individ ikke kan være både far til noen og samtidig mor. De andre klassene er derimot ikke satt disjunkte, fordi det ikke er noe i veien for at et individ både kan være `:Barn` av noen og samtidig `:Far` til noen andre.

Klassen `:AktuellAdresse` er en annen klasse som er på det øverste nivået av klassehierarkiet. Denne klassen representerer en persons aktuelle adresse og har to subklasser: `:MatrikkelAdresse` og `:OffisiellAdresse`. Disse to klassene er videre definert lik de adressene med adressetype henholdsvis lik `:matrikkelAdresse` eller `:offisiellAdresse`. Dette fører til at en resonnerer kan utlede hvilken klasse en bestemt adresse er medlem av, basert på adressetypen. Klassen som representerer postadressen til en person er klassen `:Postadresse`.

Klassen `Kode` fungerer som en superklasse for alle kodetyper innenfor DSF. Klassen har elleve subklasser som hver representerer en kodetype. Disse er:

```
:Aarsakskode
:Adressetype
:Arbeidstillatelse
:Foreldreansvar
:Kirkemedlem
:Personkode
:Registreringsstatus
:Samemanntall
:Sivilstand
:SpesifisertRegtype
:Umyndiggjort
```

Det er også laget klasser for en persons innvandring, innflytting og utvandring. En innvandring beskriver en persons innvandring fra et annet land til Norge, mens en innflytting beskriver en persons innflytting fra en kommune i Norge. Utvandring beskriver en persons utvandring fra Norge til et annet land.

[5]URI-en `http://sws.ifi.uio.no/vocab/dsf/henriwi/dsf#` er forkortet til «:».

Objektroller

Det er definert 35 objektroller i ontologien. De fleste er hentet fra beskrivelsen og databaseskjemaet, men i tillegg har det blitt lagt til noen inverse roller samt noen superroller for å gruppere rollene i hierarkier. Når det gjelder navngiving av roller er det ingen offisiell anbefaling fra W3C, men det eksisterer en navnekonvensjon som mange følger som sier at hver objektrolle bør ha prefikset *har* (eng: has) eller *er* (eng: is) [45, side 27]. Fordelen med dette kommer til uttrykk hvis vi ser på følgene tre tripler:

```
dsfp:02088600110 dsfv:harBarn dsfp:01010751388 .
dsfp:01010751388 dsfv:erBarnTil dsfp:02088600110 .
dsfp:02088600110 dsfv:barn dsfp:01010751388 .
```

Det er ingen tvil om bruken og betydning til rollene `dsfv:harBarn` og `dsfv:erBarnTil`, nemlig at den første relaterer en forelder til sitt barn, mens den andre relaterer barnet til sin forelder. Rollen `dsfv:barn` er det derimot ikke utvetydig hvilken vei det er tenkt at relasjonen skal brukes, det vil si om den skal brukes til å relatere en forelder til sitt barn, eller barnet til sin forelder. For å unngå denne problematikken har vi derfor lagt til prefiksene *har* og *er* på alle objektroller, mens vi på dataroller ikke har lagt på dette prefikset. Dette begrunnes i at siden objektet i en trippel som bruker en datarolle som predikat alltid vil være en literal, kan rollen kun brukes én vei (siden en literal ikke kan opptre som subjekt).

Følgende liste viser et utdrag av objektrollene som har med en persons familierelasjoner å gjøre, med rollens eventuelle karakteristikk i parentes. Hvis rollen har en invers rolle, vises dette ved at inversen av den inverse rollen er satt ekvivalent med selve rollen.

```
:harBarn ≡ :harForelder⁻
  :erFarTil ≡ :harFar⁻
  :erMorTil ≡ :harMor⁻
:harEktefelle (symmetrisk)
:harForelder ≡ :harBarn⁻
  :harFar ≡ :erFarTil⁻
  :harMor ≡ :erMorTil⁻
```

Videre listes alle roller som relaterer en person til en kode. Etter at konverteringen er blitt gjennomført, og før eventuell resonnering har blitt anvendt, relaterer disse rollene en person til én kode. Etter at resonnering er blitt utført vil disse rollene bli brukt til å relatere en person til koden, men også til selve verdien av denne koden som er definert i ontologien.

```
:harKode
  :erMedlemIKirken
  :erUmyndiggjort
  :harAdressetype
  :harArbeidstillatelse
  :harForeldreansvar
  :harPersonkode
```

```
:harSamemanntall
:harSivilstand
:harSpesifisertRegtype
:harStatsborgerskap
:harStatuskode
```

Dataroller

Datarollene brukes som forventet til å angi verdier som navn til en person, gatenavn til en adresse også videre. Et av de mest interessant datarollehierarkiene er hierarkiet som har `:identifikator` som øverste rolle. De tre rollene `:fodselsnummer`, `:tidligereFnrDnr` og `:nyttFodselsnummer` er alle hentet fra beskrivelsen av dumpen med folkeregisterdataene, mens de to superrollene er konstruert for å vise hvordan rollehierarkier kan utvikles. Vi skal senere vise hvordan akkurat dette hierarkiet blir viktig under integreringen av DSF- og SED-ontologien, og hvordan dette brukes under prosessen med å identifisere like personer fra SED og DSF.

Videre vises det hvordan en rolle kan brukes til å gruppere andre roller ved å lage et hierarki. Dette er vist ved at rollen `:navn` er brukt som superrolle til alle rollene som har med en persons navn å gjøre. Denne måten å gruppere relaterte roller på er brukt flere steder både når det gjelder data- og objektroller.

```
:identifikator
  :personidentifikator
    :fodselsnummer
    :nyttFodselsnummer
    :tidligereFnrDnr
:navn
  :forkortetNavn
  :fornavn
  :mellomnavn
  :slektsnavn
  :slektsnavnUgift
```

Koder

Som nevnt er det mange verdier i de originale dataene som er koder, og hvor verdiene er beskrevet i databaseskjemaene og brukerhåndboken. OWL gjør det mulig å definere forholdet mellom kode og verdi eksplisitt i modellen. Dette gjør det mulig for en bruker eller applikasjon å kunne velge mellom å få ut koden eller verdien til en gitt kodetype. I tillegg er det ikke nødvendig for en utvikler å vite hva de forskjellige kodene betyr når en applikasjon utvikles, det vil si at ontologien fungerer som modell for dataene og samtidig som dokumentasjon.

Det kan være lurt å modularisere ontologien for blant annet enklere vedlikehold og muligheter for gjenbruk [41, side 327]. For kode/verdi-parene er det blitt laget egne ontologier for hver kodetype, og disse har videre blitt importert i DSF-ontologien. For å definere forholdet mellom kode og verdi blir predikatet `owl:sameAs` benyttet. Dette predikatet brukes for å angi at to URI-er er samme entitet, og eksempelet nedenfor viser hvordan en kode og denne kodens verdi er koblet sammen ved hjelp av dette predikatet. I tillegg er det knyttet en tekstlig beskrivelse av kodeverdien til individet ved hjelp av predikatet `rdfs:comment`, som gir en beskrivelse av koden.

```
dsf:statuskode#kode-1 owl:sameAs dsf:statuskode#bosatt
```

Kodene og verdiene er definert som individer i ontologien. Dette argumenteres med at det eksisterer en rekke forskjellige koder, som igjen er av en bestemt kodetype, og som vist er det definert flere klasser som representerer ulike kodetyper. For å holde de ontologiene som inneholder kode/verdi-parene så små og enkle som mulig, og dermed enklere å vedlikeholde og utvide, er det ikke definert noen klasser i disse ontologiene. Derimot er det som vist definert klasser for hver kodetype i DSF-ontologien, og for hvert individ angitt en kodetype etter at kode/verdi-parene har blitt importert inn i DSF-ontologien. Dette gjør det mulig å legge til nye kodetyper ved å først legge til to nye individer, ett for koden og ett for verdien, og deretter angi en kodetype for disse individene.

Eksisterende vokabularer og datasett

Som nevnt i avsnitt 3.2.3 på side 33 bør det tilstrebes å gjenbruke eksisterende vokabularer der dette er mulig. Det å gjenbruke vokabularer øker forståelsen av dataene, og bidrar til lettere å integrere data fra flere kilder, hvis alle kildene bruker deler av det samme vokabularet. På den andre siden bør man være kritisk til gjenbruk av et vokabular og i såfall hvilke deler av det, fordi gjenbruk ofte kan føre til mindre presisjon med tanke på den informasjonen som skal formidles. Dette gjelder særlig i mer lukkede applikasjoner og offentlige systemer hvor datamodellen er definert av en offentlig organisasjon eller etat, og det er lite sannsynlig at det finnes et eksisterende vokabular eller en ontologi som beskriver dataene nøyaktig som de er representert. I vårt tilfelle gjelder dette da det ikke finnes noen eksisterende ontologier som beskriver DSF eller SED, men det er allikevel deler av ontologien som, om ikke erstattes, så ihvertfall kan utvides ved hjelp av eksisterende vokabularer.

Et interessant vokabular er `?hvor`-vokabularet [69], som er et vokabular for å beskrive norske adresser. Siden DSF-dataene inneholder adresseinformasjon er det naturlig å prøve å anvende dette vokabularet i ontologien til DSF. Problemet er at det er flere steder hvor det er uoverensstemmelser mellom `?hvor`-vokabularet og DSF-dataene. Mange av adressefeltene i dumpen fra DSF er slått sammen til to felter, og for eksempel er

feltene `bruksnummer` og `husnummer`, som henholdsvis angir bruksnummeret i en matrikkeladresse og husnummer relativt til gatenavn, slått sammen til ett felt. Dette gjør at `?hvor`-vokabularet ikke er perfekt til å beskrive en adresse, men noe fra vokabularet kan allikevel brukes. Vokabularet er publisert og tilbyr en tekstlig beskrivelse av alle ressursene i vokabularet, og følger prinsippene til lenkede data og innholdsforhandling, og kan tilby både en maskinleselig RDF-versjon eller en HTML-beskrivelse for mennesker. Dette er veldig nyttig med tanke på at det kan være brukere av DSF-dataene som ønsker en forklaring på hva de forskjellige feltene i dataene betyr, og `?hvor`-vokabularet kan benyttes til å innhente definisjoner knyttet til adressebegreper. Relasjoner mellom DSF-ontologien og `?hvor`-vokabularet kan angis ved å sette roller og klasser i DSF-ontologien som subklasser eller ekvivalente klasser til roller og klasser i `?hvor`-vokabularet. Rollen `dsfv:kommunenummer` i DSF-ontologien er satt ekvivalent med predikatet `hvor:kommunenr` og `dsfv:postnummer` er satt ekvivalent med `hvor:postnummer`. Videre er klassen `dsfv:AktuellAdresse` satt til subklasse av `hvor:Adresse`.

En konsekvens av at de eksisterende vokabularene er koblet til ontologien, er at informasjonen vil «sildre» ned på dataene under resonnering, og utvide datasettet uten å måtte endre noe på dataene. Dette viser en fordel med ontologier og hvordan informasjon fra modellen kan utvide datasettet uten at dataene trenger å endres av den grunn.

I tillegg til å anvende eksisterende vokabularer kan DSF-dataene utvides ved å ha lenker til andre datasett. DSF-dataene inneholder flere landkoder som blant annet brukes til å angi hvilket land en person har innvandret fra eller utvandret til. Til det vi kjenner til så følger disse kodene ISO-standarden *ISO 3166-1 numeric* hvor landene er representert med en 3-sifret landkode. *Geonames*[6] er en database som blant inneholder alle land i verden. Disse tilbyr også en RDF-representasjon av datasettet, og gjør det mulig å lage lenker fra DSF-datasettet til dette datasettet. For de kodene hvor vi har funnet korresponderende land[7] er det blitt lagt til en lenke mellom koden og det faktiske landet representert i geonames. For eksempel tilsvarer koden `104` landet Kamerun som er identifisert av URL-en `http://sws.geonames.org/2233387`. Ved å lage lenker til dette eksisterende datasettet fra DSF-datasettet, vil DSF-dataene bli utvidet og det er også mulig å navigere seg videre fra geonames-datasettet til andre datasett som dette datasettet har lenker til.

[6] `http://www.geonames.org/`

[7] Vi har ikke klart å finne korresponderende land til alle kodene i datasettet, men dette kan ha med å gjøre at vi jobber med testdata, og at noen av landkodene har blitt anonymisert til ikke-eksisterende land.

4.5 Ontologi for SED

SED-ontologien er utviklet for å representere deler av en SED. Det eksisterer flere SED-er som alle er beskrevet av to forskjellige type dokumenter. Det første er et Word-dokument, mens det andre er et XML-skjema. Word-dokumentet blir fylt ut av saksbehandlerne hos NAV, men for å sende SED-filene til andre etater konverteres Word-dokumentene til XML og det er XML som blir brukt som utvekslingsformat. Videre blir det derfor fokusert på XML-skjemaet til SED-filene og ikke Word-dokumentet.

Som nevnt i seksjon 2.4 på side 8, vil vi fokusere på flyt FA002, som benyttes ved lovvalg. Innenfor denne flyten vil vi videre fokusere på A003 som inneholder informasjon som NAV mottar og bruker til å identifisere personer.

4.5.1 Analyse av kravene til ontologien

SED-ontologien er utviklet for å vise hvordan datakilder kan integreres ved hjelp av semantiske teknologier og RDF og ontologier spesielt. Kravene til SED-ontologien er veldig lik kravene til DSF-ontologien og har de samme to bruksområdene: å kunne bruke ontologien som et spørregrensesnitt samt kunne utføre resonnering generelt og spesielt ved at to personer kan bli angitt som like. SED-ontologien er ment for å vise hvordan to ontologier kan integreres og videre hvordan to datakilder beskrevet av disse ontologiene kan integreres. På grunn av mangel på kilder, samt at hovedfokuset har vært å utvikle DSF-ontologien, vil også SED-ontologien være mager og ikke komplett. Kompleksiteten vil derfor heller ikke her være et viktig krav.

4.5.2 Kilder

Figur 4.8 på neste side viser et utdrag fra SED-dokumentet A003. Dokumentet inneholder mye informasjon som ikke er så relevant for å illustrere hvordan SED kan integreres mot DSF, og det er derfor mye informasjon i dokumentet som vi ikke har prøvd å modellere. Vi har valgt å fokusere på den delen av dokumentet som inneholder informasjon om en person, og figuren viser et uttrekk av dette.

SED skiller mellom to forskjellige type personer. Den første typen er personer som har et personlig identifiseringsnummer, mens den andre typen er personer som ikke er identifisert av et slikt nummer. For begge typene lagres følgende informasjon:

- Familienavn
- Fornavn
- Fødselsdato

```
<neutralMainStructure>
  <field xmi.id="Im7b63b671m12944991835mmcfb"
    attributeName="familyName"
    maxOccurs="1"
    minOccurs="1"
    dataType="string"
    longLabel=""
    shortLabel="Family name(s)"
    documentation="" equivalent="">
    <constraints>
      <constraint type="xsd" value="maxLength 155"/>
    </constraints>
  </field>
  <field xmi.id="Im7b63b671m12944991835mmce7"
    attributeName="forenames"
    maxOccurs="1"
    minOccurs="1"
    dataType="string"
    longLabel=""
    shortLabel="Forename(s)"
    documentation=""
    equivalent="">
    <constraints>
      <constraint type="xsd" value="maxLength 155"/>
    </constraints>
  </field>
  <field xmi.id="Im7b63b671m12944991835mmcd2"
    attributeName="birthDate"
    maxOccurs="1"
    minOccurs="1"
    dataType="date"
    longLabel=""
    shortLabel="Birth date"
    documentation=""
    equivalent=""/>
  <enum xmi.id="I76c91f3fm120608fe812m78da"
    className="SexIdentifier"
    packageName="GlobalEESSI"
    attributeName="gender"
    shortLabel="Sex"
    longLabel=""
    documentation=""
    equivalent=""
    minOccurs="0"
    maxOccurs="1"
    dataType="string">
    <option code="Female" value="Female"/>
    <option code="Male" value="Male"/>
    <option code="unknown" value="Unknown"/>
  </enum>
  .
  .
  .
</neutralMainStructure>
```

Figur 4.8: Et utdrag fra SED-dokumentet A003.

- Kjønn (mann, kvinne eller ukjent)
- Familienavn ved fødselen
- Fornavn ved fødselen

For personer med et personlig identifiseringsnummer lagres i tillegg følgende informasjon:

- Personlig identifiseringsnummer hos avsender
- Personlig identifiseringsnummer hos mottaker

Mens det for personer uten et personlig identifiseringsnummer i tillegg lagres følgende:

- Fødested
- Fars familienavn ved fødselen
- Mors familienavn ved fødselen
- Fars fornavn
- Mors fornavn

4.5.3 Ontologien

Klasser

Ontologien inneholder klassen `sedv:SED`[8] som representerer én SED. Denne inneholder såkalt proveniens informasjon, i denne sammenheng informasjon om opphavet til SED-en, når ble den sendt, hvem har sendt den også videre. Videre er klassen `sedv:Person` definert, som representerer en person i SED og inneholder de egenskapene som er felles for begge persontypene. Det skilles så mellom de to persontypene ved å ha to subklasser av `sedv:Person`, henholdsvis `sedv:PersonWithPin` og `sedv:PersonWithoutPin`. Klassen `sedv:PersonWithPin` er klassen for de personene hvor det foreligger et personlig identifikasjonsnummer, mens `sedv:PersonWithoutPin` er klassen for de personene hvor et slikt nummer ikke foreligger. Det er så lagt til et såkalt dekningsaksiom (eng: covering axiom) som er et utsagn som sier at alle individer av typen `sedv:Person`, også må være av typen `sedv:PersonWithPin` eller `sedv:PersonWithoutPin`. Den siste klassen er `sedv:Gender` som representerer kjønnet til en person. En ting å legge merke til er at SED-ontologien er modellert med engelske navn, mens DSF-ontologien er modellert med norske navn. Dette er fordi dokumentasjonen som ontologiene er basert på henholdsvis er på engelsk og norsk.

[8] Prefikset `sedv:` angir URI-en `http://sws.ifi.uio.no/vocab/dsf/henriwi/sed#`

Objekt- og dataroller

Vi har to objektroller, og det er rollen `sedv:hasGender` som relaterer en person til sitt kjønn, og rollen `sedv:hasPerson` som relaterer et individ av typen `sedv:SED` til et individ av typen `sedv:Person`. De andre rollene er dataroller, og ontologien inneholder følgende dataroller:

```
sedv:birthDate
sedv:caseNumberSending
sedv:dateSent
sedv:familyName
sedv:familyNameAtBirth
sedv:fatherFamilyNameAtBirth
sedv:forenameOfFather
sedv:forenameOfMother
sedv:forenames
sedv:forenamesAtBirth
sedv:institutionCode
sedv:institutionName
sedv:motherFamilyNameAtBirth
sedv:pinReceivingInstitution
sedv:pinSendingInstitution
sedv:placeOfBirth
```

Hvis vi ser nærmere på figur 4.8 på side 72, ser vi at det er flere steder hvor attributtene *minOccurs* og *maxOccurs* benyttes etterfulgt av en verdi. For eksempel har attributtnavnet *birthDate* verdien 1 hos både *minOccurs* og *maxOccurs*. Dette betyr at fødselsdatoen skal opptre én og bare én gang i SED-dokumentet. Dette er videreført til modellen ved å si at et individ av typen `sedv:Person`, kun kan ha én fødselsdato. Dette viser hvordan OWL kan modellere kardinalitetsutsagn, og hvordan et XML-skjema kan oversettes til OWL hvor semantikken til skjemaet ivaretas.

SED-data

Basert på ontologien er det blitt laget flere eksempel SED-er i RDF, men det er ikke brukt tid på å konvertere SED-filer fra de originale dataene til RDF. For det første har vi vist detaljert hvordan folkeregisterdataene kan konverteres til RDF, og de samme prinsippene kan brukes for å konvertere XML til RDF (bortsett fra at det kreves andre verktøy som kan behandle XML). I tillegg har vi ikke fått tilgang til eksempeldata av SED-filene.

Figur 4.9 på neste side viser et eksempel på hvordan en SED med en person som ikke har et personlig identifiseringsnummer blir representert som RDF. Figuren viser hvordan en SED blir representert som et individ av typen `sedv:SED`, og at dette individet har egenskaper som sier hvilken institusjon som har sendt SED-en, når den er sendt, nummer på sendingen og relasjonen til personen som SED-en omhandler.

```
@prefix sedv: <http://sws.ifi.uio.no/vocab/dsf/henriwi/sed#> .
@prefix sed: <http://sws.ifi.uio.no/data/dsf/henriwi/sed/> .
@prefix sedp: <http://sws.ifi.uio.no/data/dsf/henriwi/sed/person/> .
@prefix gender: <http://sws.ifi.uio.no/data/dsf/henriwi/sed/gender/> .

sedp:1 a sedv:PersonWithoutPIN ;
    sedv:birthDate "2007-01-01"^^xsd:date ;
    sedv:familyName "TOPSTAD" ;
    sedv:familyNameAtBirth "TOPSTAD" ;
    sedv:forenames "TOMAS" ;
    sedv:hasGender gender:male ;
    sedv:forenameofFather "JOSTEIN" ;
    sedv:forenameofMother "TÜVA" .

sed:100 a sedv:SED ;
      sedv:caseNumberSending "100" ;
      sedv:insititutionCode "103" ;
      sedv:dateSent "2012-03-25"^^xsd:date ;
      sedv:hasPerson sedp:1 .
```

Figur 4.9: En SED og en person fra SED uten et personlig identifiseringsnummer representert som RDF.

4.6 Integrering av ontologiene

Det å integrere flere ontologier på tvers av ulike domener kalles ontologimatching. Dette sees på som en løsning på problemet med semantisk interoperabilitet, ved at korrespondanser mellom semantisk-relaterte entiteter i ontologiene etableres, og dette muliggjør samhandling av kunnskapen og dataene som uttrykkes gjennom ontologiene [27].

En mulighet når ontologimatching skal benyttes til å integrere flere modeller, er å lage en egen modell som kun har som oppgave å relatere begreper mellom de ulike ontologiene. Dette er særlig hensiktsmessig når mange modeller skal integreres, og det er vanskelig å finne én modell som de andre skal matches mot. I vårt tilfelle derimot er det kun to ontologier som skal integreres, og vi anser DSF-ontologien som en hovedontologi og SED-ontologien som en ontologi som bør matches direkte opp mot denne. Vi har derfor ikke laget en tredje ontologi, men integrert ontologiene ved å koble konsepter og egenskaper fra SED-ontologien opp mot konsepter og egenskaper i DSF-ontologien. Konseptene og egenskapene er blitt koblet sammen ved hjelp av predikatene `owl:equivalentProperty`, `rdfs:subPropertyOf`, og `rdfs:subClassOf`, som henholdsvis brukes til angi ekvivalente roller, subroller og subklasser.

For å integrere klassen `sedv:Person` mot DSF-ontologien er én mulighet å sette denne klassen som subklasse av `dsfv:Person`, eller sette `dsfv:Person` som subklasse av `sedv:Person`. Men det er ikke nødvendigvis slik at alle SED-personer også er DSF-personer eller motsatt. Vi har derimot benyttet oss av det eksisterende vokabularet FOAF, som har definert

klassen `foaf:Person`, og både `dsfv:Person` og `sedv:Person` er satt som subklasser av `foaf:Person`.

Når det gjelder egenskapene så viser listen nedenfor de aktuelle rollene fra SED-ontologien og forholdet de har fått til roller i DSF-ontologien.

- `sedv:familyName` $\equiv$ `dsfv:slektsnavn`
- `sedv:forenames` $\equiv$ `dsfv:fornavn`
- `sedv:placeOfBirth` $\equiv$ `dsfv:foedested`
- `sedv:pinSendingInstitution` $\sqsubseteq$ `dsfv:personidentifikator`
- `sedv:pinReceivingInstitution` $\sqsubseteq$ `dsfv:personidentifikator`

Det hadde vært ønskelig å kunne modellere sammenhengen mellom navnet til en persons foreldre i SED og navnet til en persons foreldre i DSF. Dette kunne for eksempel bli modellert ved å angi at `sedv:forenameOfFather` $\equiv$ `dsfv:harFar`, `dsfv:fornavn`, som sier at fornavnet til faren i SED er ekvivalent med rollekjeden som går fra en person til personens far og videre til farens fornavn i DSF. Det er nødvendig å bruke en slike rollekjede fordi det ikke eksisterer en relasjon direkte fra en person til farens fornavn i DSF-ontologien. Problemet er at rollekjeder kun kan brukes for objektroller og det er derfor ikke mulig å modellere forholdet mellom en person og farens navn på denne måten.

Etter at konsepter og relasjoner mellom de to ontologiene er blitt koblet sammen, er predikatet `owl:hasKey` benyttet til å angi hvilke egenskaper som unikt identifiserer en person. Dette predikatet kan sammenlignes med en primærnøkkel i relasjonelle databaser, og ved å bruke dette predikatet kan en resonnerer utlede at to entiteter er like fordi de har samme verdier knyttet til egenskapen(e) som er angitt som nøkkel. I vårt tilfelle er dette blitt brukt for å angi at en person er unikt identifisert av rollen `dsfv:personidentifikator`. Siden både fødselsnummer og det personlige identifiseringsnummeret som brukes i SED er satt som subrolle av denne rollen, vil en person som er lagret i SED med samme identifikator som en person i DSF, bli satt til være samme entitet. Helt konkret vil det legges til en relasjon mellom de to individene hvor relasjonen er beskrevet av predikatet `owl:sameAs`.

Det å si at to personer er like hvis de har samme identifiseringsnummer er brukt for å vise hvordan OWL, sammen med resonnering, automatisk kan fastslå at to entiteter fra forskjellige datasett beskriver samme person, og for å vise mulighetene med teknologiene. Det er ingenting i veien for at helt andre og mer komplekse nøkler kan brukes til dette i en eventuell produksjonssatt løsning. En interessant nøkkel kunne for eksempel vært kombinasjonen av en persons navn, fødested, fødselsdato og foreldrenes navn. I dag må alle disse feltene sjekkes manuelt opp mot DSF, men hvis det var mulig å sette dette som en nøkkel ville det vært mulig å automatisk fastslå at to personer var like basert på disse kriteriene. Som vi derimot har sett er det ikke mulig å relatere navnet til en persons foreldre i DSF og SED

på denne måten ved hjelp av OWL, og dermed bruke dette i en nøkkel. En annen ulempe er at to personer kun vil bli identifisert som like hvis det er en eksakt match på verdiene, det vil si at to personer ikke blir satt som samme person hvis det for eksempel er forskjell i stavemåter.

Figur 4.10 viser alle klassene fra begge ontologiene, og sammenhengen mellom dem etter at ontologiene er integrert.

Figur 4.10: Klassene og sammenhengen mellom dem etter at ontologiene er integrert.

Kapittel 5

Implementasjon

Det forrige kapittelet beskrev hvordan løsningen er blitt designet med hensyn til konvertering av data og utvikling og integrering av ontologiene. Dette kapittelet beskriver hvordan de konvertere dataene og ontologiene er benyttet til å implementere en demonstrator som distribuerer dataene og som dermed gjør det mulig å konsumere og bruke dataene.

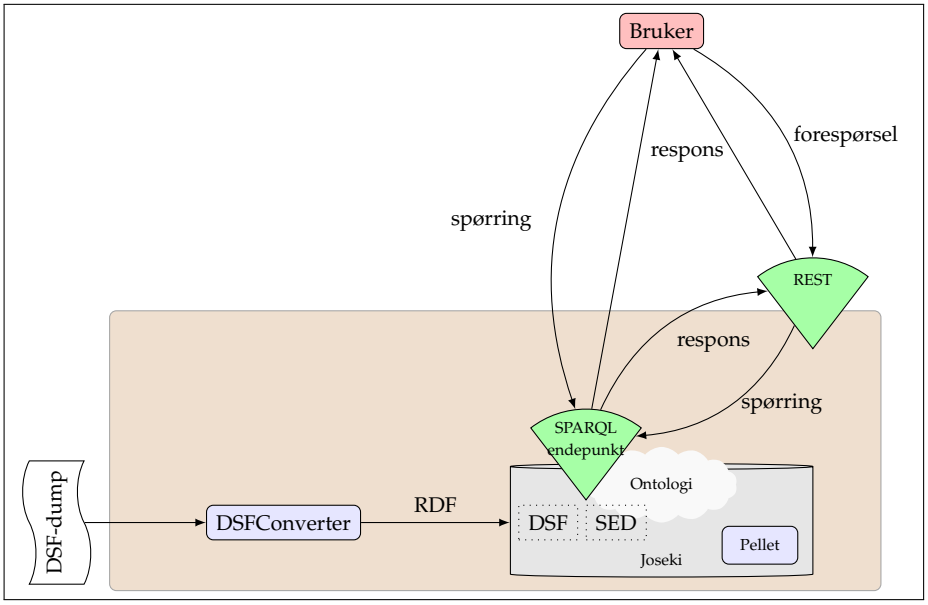
Kapittelet har først i seksjon 5.1 en overordnet beskrivelse av arkitekturen til demonstratoren før seksjon 5.2, 5.3 og 5.4 beskriver de forskjellige komponentene i arkitekturen i større detalj, henholdsvis triple store og SPARQL-endepunkt, resonnerer og et RESTfullt grensesnitt. Til slutt i seksjon 5.5 vises det hvordan demonstratoren kan brukes som et supplement til dagens løsning ved å vise implementasjon av et program som er utviklet for å bidra til å automatisere konverteringsprosessen beskrevet i kapittel 4.

Det eksisterer en rekke forskjellig implementasjoner av alle de forskjellige komponentene til demonstratoren, men kapittelet vil ikke vise og vurdere forskjellige implementasjoner, men kun vise hvilken løsning som er valgt for demonstratoren. Oppgaven er ikke ment å være en studie i forskjellig programvare og verktøy, og en slik studie ville bli alt for omfattende for denne oppgaven. Men dette kapittelet gir et lite bilde av hva som finnes, og for en mer komplett oversikt over verktøy som kan brukes til å jobbe med semantiske teknologier anbefales følgende side: `http://www.w3.org/2001/sw/wiki/Tools`, samt boken *Semantic Web Programming* [38].

5.1 Arkitektur

Figur 5.1 på neste side viser de forskjellige komponentene til demonstratoren. Hovedkomponentene er en triple store som fungerer som lagringskomponent, et SPARQL-endepunkt for konsumering av data ved hjelp av SPARQL-spørringer, et RESTfullt grensesnitt for enkelt å kunne slå opp på enkeltressurser og lister av en ressurstype, og en resonnerer for å kunne

utføre resonnering. På siden av selve demonstratoren, men som en del av systemet, ligger et program som er utviklet for å automatisere konverteringen fra dumpen med folkeregisterdata til RDF.



Figur 5.1: Arkitekturen til demonstratoren.

Mange rammeverk tilbyr ofte komponenter som triple store, SPARQL-endepunkt og resonnerer samlet i en pakke, mens andre tilbyr muligheter for å tilpasse de forskjellige modulene etter hva slags krav som er stilt til løsningen. For eksempel hvis løsningen må håndtere store mengder data og har et krav om rask responstid, men det ikke er så viktig å utføre resonnering over en kompleks ontologi, bør det velges en triple store som kan håndtere nettopp dette, sammen med en enkel og rask resonnerer. Hvis løsningen derimot skal håndtere lite data, men har en kompleks ontologi som det skal kunne resoneres over, bør en annen type triple store velges sammen med en resonnerer som har støtte for en kompleks modell [38, side 144]. Ved å velge en løsning hvor komponentene kan tilpasses og byttes ut, fører dette til at de forskjellige komponentene i arkitekturen blir løst koblet, det vil si at en komponent kan byttes ut med en annen komponent med liten innvirkning på de andre komponentene.

5.2 Joseki

Joseki [46] er en SPARQL-server som bruker Jena [30] og spørreprosessoren ARQ til å tilby et grensesnitt for å ta i mot, behandle og returnere svar på SPARQL-spørringer over HTTP. For å håndtere HTTP-forespørsler benytter Joseki seg av HTTP-serveren Jetty [51]. Jena er et rammeverk for å lage applikasjoner som bruker semantiske teknologier, og inneholder API-er for å opprette og manipulere RDF, håndtere ontologier, flere

regelbaserte resonnerere, og API-er for lagring av RDF-data til disk. ARQ er spørreprosessoren til Jena, som støtter SPARQL, SPARQL 1.1 Query og SPARQL 1.1 Update.

Jena tilbyr flere muligheter for å persistere RDF-dataene. En mulighet er å bruke SDB, som er en lagringskomponent som bruker relasjonelle databaser til å persistere RDF-datene og støtter mange type databaser, blant annet MySQL, PostgreSQL og Microsoft SQL Server. Ved å bruke SDB kan mulighetene som ligger i den underliggende databasen når det gjelder transaksjonsstøtte, sikkerhet, administrasjon og lastbalansering anvendes [75]. En annen mulighet er å bruke TDB, som er en grafbasert triple store-implementasjon. Det er også mulig å bruke en minnemodell som lagringskomponent.

Demonstratoren bruker en minnemodell til å persistere dataene og Joseki som SPARQL-endepunkt, som gjør det mulig å stille SPARQL-spørringer over HTTP til demonstratoren. Siden Joseki bygger på Jena gjør dette det mulig å bruke all funksjonalitet som Jena tilbyr i demonstratoren. En av de største fordelene er tilgangen til spørreprosessoren ARQ som Jena bruker. Siden ARQ støtter SPARQL 1.1 blir det mulig å stille spørringer som benytter seg av aggregering, subspørringer og negasjon. ARQ har i tillegg tilgang til en del ekstra funksjonalitet som gjør den til en attraktiv spørringsprosessor. Dette er blant annet eksterne funksjonsbiblioteker som tilbyr konkatenering av strenger og fritekstsøk. I tillegg støtter Joseki navngitte grafer som gjør det mulig å lagre flere datasett i samme triple store, og skille de ved å tilordne forskjellige navngitte grafer til hvert datasett. Som vi skal se senere er det også mulig å bruke ulike eksterne resonnerere i tillegg til de som er inkludert med Jena. En annen fordel med Joseki er at den er mye brukt av andre prosjekter, og er lett tilgjengelig, samt at den er godt dokumentert. I tillegg er det god ekspertise på Ifi, og testserveren som demonstratoren ligger på har allerede flere andre demonstratorer kjørende som bruker Joseki, og det er dermed enkelt å anvende Joseki også til vår demonstrator.

Grunnen til at demonstratoren bruker en minnemodell er at antall tripler som forvaltes foreløpig er relativt få, men hvis antall tripler øker vil det antageligvis være ønskelig å konfigurere Joseki til å lagre dataene til disk istedenfor å kun ha dem i minne. Ulempen er også at en minnemodell vil føre til at eventuelle data som har blitt lagt inn fjernes hver gang Joseki startes på nytt. Argumentasjonen mot dette er at demonstratoren først og fremst er tenkt brukt til å hente ut og distribuere data, og ikke som mottaker av data.

DSF- og SED-dataene er lagt inn i Joseki ved å knytte en navngitt graf til hvert datasett. Det vil si at DSF-dataene er knyttet til standardgrafen, mens SED-dataene er knyttet til en navngitt graf identifisert av følgende URI: `http://sws.ifi.uio.no/data/dsf/henriwi/sed`. Dette viser hvordan forskjellige datasett kan lagres i samme triple store og allikevel holdes adskilt med tanke på spørringer og tilgang.

Som nevnt er et viktig prinsipp innenfor semantiske teknologier og lenkede data, at URL-ene som anvendes skal være derefererbare, det vil si at en representasjon eller beskrivelse av entiteten returneres når en HTTP-GET forespørsel sendes til URL-en. For ontologien er dette løst ved at ontologiene ligger lagret på den faktiske HTTP URL-en som ontologiene bruker som navnerom. DSF-ontologien bruker URL-en `http://sws.ifi.uio.no/vocab/dsf/henriwi/dsf#` som navnerom, og ved å besøke denne URL-en vil den faktiske ontologien bli returnert. For dataene er dette blitt løst ved å bruke et RESTfullt grensesnitt som returnerer data om individer på en rekke forskjellige formater. Dette grensesnitt blir nærmere beskrevet i seksjon 5.4 på neste side.

Figur 5.2 viser hvordan en SPARQL-spørring kan skrives mot demonstratoren og hvordan dataene videre blir presentert til brukeren ved hjelp av SNORQL[1].

Figur 5.2: Eksempel på en SPARQL-spørring og resultatet av denne.

[1] `https://github.com/kurtjx/SNORQL`

5.3 Pellet

Joseki har, ved hjelp av Jena, støtte for flere innebygde resonnerere. De forskjellige resonnererne har blant annet støtte for RDFS, og forskjellige, men ikke komplette subsett av OWL [62]. Blant annet er det ingen resonnerere som har støtte for predikatet `owl:sameAs`. Som vist i seksjon 4.4.3 på side 65 brukes dette predikatet flere steder i ontologien, blant annet for å beskrive forholdet mellom en kode og dens verdi, og det er derfor viktig at resonnereren som anvendes har støtte for dette predikatet.

Joseki har også mulighet for å bruke en ekstern resonnerer, og demonstratoren benytter Pellet [67] som resonnerer. Dette er en OWL 2-resonnerer som også har støtte for OWL 2-profilene [23]. Pellet er fritt tilgjengelig og har blitt et populært verktøy for å jobbe med OWL, og tilbyr et API for å kunne brukes i applikasjoner som bruker Jena, og siden Joseki bruker Jena, kan Pellet fint brukes sammen med Joseki.

Resonnering er en kostbar prosess og stiller store krav til maskinvare med tanke på CPU og minnebruk. Serveren som demonstratoren kjører på brukes også til flere andre demonstratorer, og det er ikke tilstrekkelige ressurser på denne serveren til at resonnering kan utføres. Løsningen har dermed blitt at resonneringen utføres offline som en del av konverteringsprosessen. Dette er ikke helt ønskelig siden det ikke vil utføres resonnering over data som blir lagt inn i demonstratoren, men argumentet for dette er at demonstratoren som nevnt er tenkt til å brukes til distribusjon og konsumering av data, og ikke som et sted å legge inn nye data.

5.4 Elda

Ved hjelp av Joseki har demonstratoren mulighet for å lagre RDF-data og distribuere disse via et SPARQL-endepunkt som kan motta og behandle SPARQL-spørringer over HTTP. Som diskutert i seksjon 3.3.3 på side 42, gir et SPARQL-endepunkt en kraftig og fleksibel måte å konsumere dataene på, men kan for flere virke kompleks. For å skjule detaljene til SPARQL og gjøre det enklere for applikasjoner og brukere å hente ut data fra demonstratoren, er det derfor brukt et RESTfullt grensesnitt som supplement til SPARQL-endepunktet for konsumering av data. Dette grensesnittet tar i mot HTTP-forespørsler og kommuniserer videre med SPARQL-endepunktet over HTTP før dataene blir returnert til brukeren.

Linked data API (LDA) er en spesifikasjon som tilbyr et brukervennlig webgrensesnitt over lenkede data [26]. Denne spesifikasjonen er et initiativ for å gjøre det enklere å konsumere lenkede data fra SPAQRL-endepunkter, og gjøre det enklere å lage applikasjoner over de publisert dataene ved å anvende standardiserte verktøy. LDA fungerer ved at SPARQL-spørringer bindes opp mot ulike URL-er, og når en HTTP-forespørsel sendes mot en bestemt URL, sendes det en SPARQL-spørring til SPARQL-endepunktet

som er definert i konfigurasjonen, og data fra endepunktet blir returnert via det RESTfulle grensesnittet til brukeren. LDA er godt dokumentert, og gjør det mulig å lage et kraftig RESTfullt grensesnitt. Det finnes flere implementasjoner av LDA og vi har benyttet Elda[2], som er en Java-implementasjon av LDA. Elda er valgt fordi det er lett tilgjengelig og installeres enkelt på servermiljøet som resten av demonstratoren kjører på.

Demonstratoren har konfigurert en rekke URL-er for å vise hvordan det RESTfulle grensesnittet kan anvendes. For eksempel er det bundet en SPARQL-spørring opp mot URL-en `http://sws.ifi.uio.no/data/dsf/henriwi/person`, som returnerer en liste over alle entiteter av typen person. Mens URL-en `http://sws.ifi.uio.no/data/dsf/henriwi/person/01010752171` returnerer en representasjon av entiteten som er identifisert av samme URL. Som vi ser brukes det de samme URL-ene i det RESTfulle grensesnittet som brukes til å identifisere individer i datasettet, og dette fører til at en HTTP-GET forespørsel til en URL i datasettet vil føre til at en representasjon av entiteten blir returnert. Figur 5.3 på neste side viser et uttrekk av HTML-visningen av personen identifisert av URL-en: `http://sws.ifi.uio.no/data/dsf/henriwi/person/01010752171`.

LDA har støtte for en rekke resultatformater, blant annet HTML, RDF/XML, Turtle og JSON. Hvilket resultatformat som blir returnert kan bestemmes på to måter. Enten kan ønsket resultatformat settes som verdi i *Accept*-feltet i HTTP-headeren i forespørselen, eller så kan URL-en tilføyes ønsket innholdstype. For eksempel vil følgende URL returnere JSON: `http://sws.ifi.uio.no/data/dsf/henriwi/person/01010752171.json`, mens følgende URL vil returnere Turtle: `http://sws.ifi.uio.no/data/dsf/henriwi/person/01010752171.ttl`.

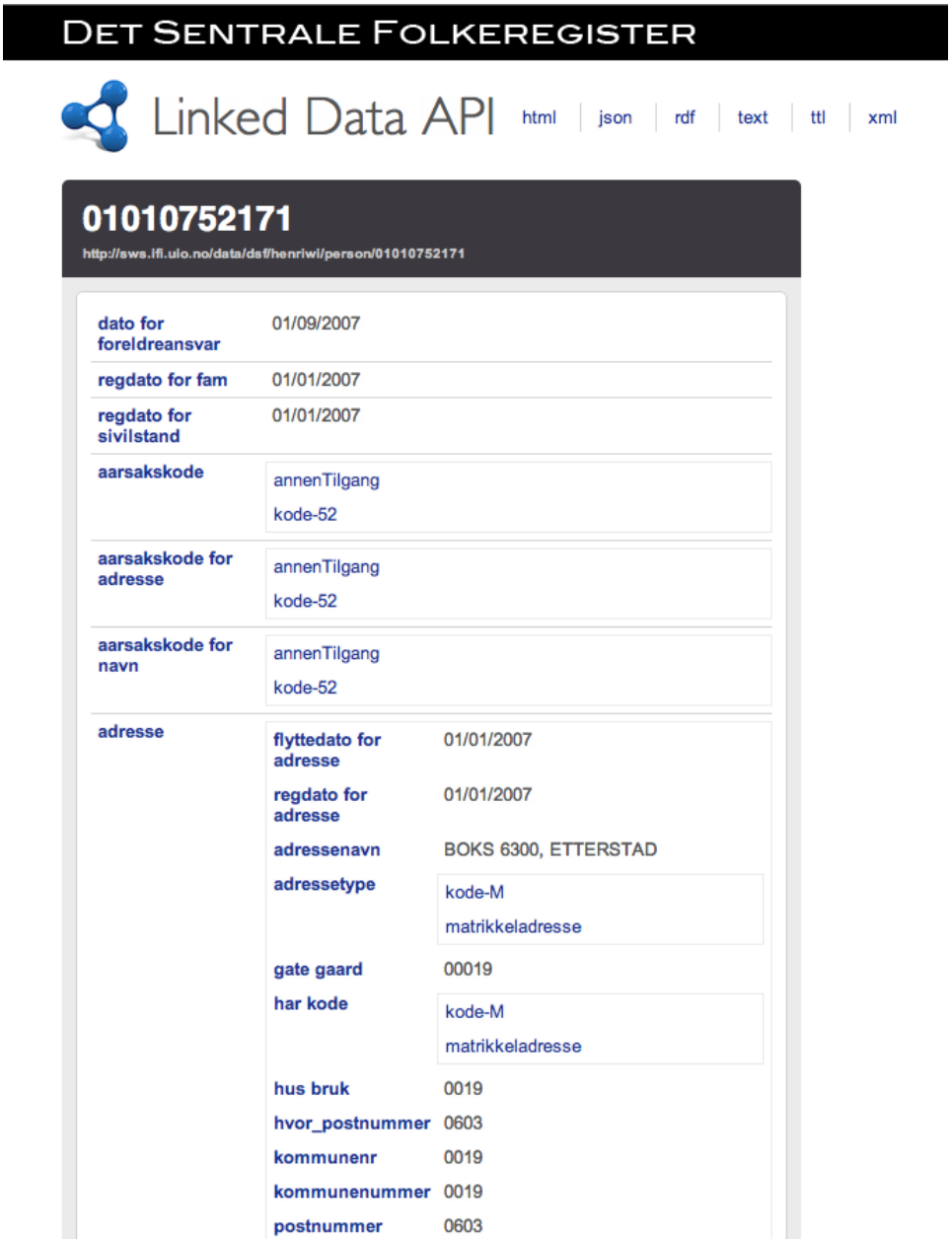
Ved å ha denne tjenesten som et supplement i tillegg til SPARQL-endepunktet kan brukere slå opp på enkeltressurser på en enkel måte, og dermed raskt finne ut om det eksisterer en person med et gitt fødselsnummer i datasettet. I tillegg er det mulig å knytte komplekse SPARQL-spørringer opp mot URL-er, og dermed skjule detaljene ved SPARQL, samtidig som mulighetene SPARQL gir kan utnyttes.

5.5 DSFConverter

Demonstratoren som er implementert er ikke tenkt å erstatte dagens Folkeregister, men være en sidemodul, og tanken er at demonstratoren kan få matet inn data fra DSF med jevne mellomrom via en automatisk oppdateringstjeneste. For å bidra til denne prosessen har vi skrevet et Java-program (DSFConverter) som tar dumpen med folkeregisterdata som input og genererer RDF. Programmet består av fire hovedsteg:

1. Konvertering av rådataene til CSV.

[2]http://code.google.com/p/elda/



Figur 5.3: HTML-visning av en person med fødselsnummer `01010752171` i Elda.

2. Konvertering fra CSV til RDF.
3. Videre transformasjon av RDF-dataene ved hjelp av `CONSTRUCT`-spørringer.
4. Resonnering over dataene og ontologi (valgfri).

Programmet er skrevet i Java og bruker OpenCSV [56], XLWrap [48] og Jena for å konvertere dataene. OpenCSV brukes for å konvertere rådataene til CSV, og er et åpen-kildekode bibliotek for konvertering til og fra kommaseparerte filer. XLWrap er et verktøy for å konvertere tabulære data til RDF, og blir beskrevet nærmere i seksjon 5.5.1 på neste side.

Tanken er at programmet skal være generisk nok til å kunne brukes mot en hvilken som helst dump med data, og ikke bare den vi har mottatt, og programmet kan derfor tilpasses ved hjelp av eksterne konfigurasjonsfiler. For å tilpasse programmet eksisterer det en konfigurasjonsfil hvor en rekke egenskaper kan settes avhengig av ønsket oppførsel til programmet. Konfigurasjonsfilen er en Java properties fil, og inneholder følgende egenskaper:

DUMP En kommaseparert liste med stien til hvor dump-filen(e) ligger. Fra 1 til n dump-filer kan angis.

CSV En kommaseparert liste med stien til hvor CSV-filen(e) skal lagres. Fra 1 til n CSV-filer kan angis, men antall CSV-filer må stemme med antall DUMP-filer.

RECORD_CONFIG En kommaseparert liste med stien til hvor CONFIG-filen(e) ligger. En slik CONFIG-fil inneholder alle feltene som rådataene inneholder og har to verdier. Den første verdien sier hvilket tegn feltet starter på og den andre hvor feltet slutter. Et utdrag fra en slik RECORD-fil vises nedenfor. Navnet på nøkkelen brukes som kolonnenavn i CSV-filen.

```
FODSELSNR=0,11
STATUSKODE=11,12
DATODOED=12,20
```

REMOVE_ALL_WHITESPACES En kommaseparert liste hvor verdien `true` angis hvis det ønskes at alle mellomrom skal fjernes for hver linje i dump-filen. Grunnen til at denne egenskapen er med er fordi de to dumpfilene vi har mottatt skiller seg fra hverandre ved at under konverteringen av den ene dump-filen må mellomrommene fjernes for hver linje, men under konverteringen av den andre skal dette ikke gjøres.

RDF Stien til hvor de konverterte RDF-dataene skal lagres. Filendelsen angir hvilken serialisering som blir benyttet.

OWL_MODEL Stien til hvor ontologien ligger lagret. Denne trenger kun å være angitt hvis det ønskes resonnering og REASON er satt til true.

RDF_INFERED Stien til hvor den resonerte versjonen av RDF-dataene skal lagres. Denne trenger kun å være angitt hvis det ønskes resonnering og REASON er satt til true. Filendelsen angir hvilken serialisering som blir benyttet.

XLWRAP_MAPPING Stien til hvor mappingfilen som brukes av XLWrap ligger lagret.

QUERY_CONFIG Stien til hvor konfigurasjonfilen for `CONSTRUCT`-spørringene ligger lagret. Denne konfigurasjonsfilen inneholder stien til hvor `CONSTRUCT`- og `DELETE`-spørringene som brukes i transformasjonen av RDF-dataene, ligger lagret.

REASON Sannhetsverdi (true eller false) om programmet skal utføre resonnering eller ikke.

5.5.1 XLWrap

XLWrap er et verktøy for å konvertere regneark til RDF, og støtter både Microsoft Excel og åpne dokumenter som kommaseparerte dokumenter (CSV). XLWrap støtter både radbasert konvertering, det vil si at en og en rad behandles om gangen, og konvertering som ikke er basert på rader og multidimensjonale tabeller.

Til konverteringen brukes et templatespråk som brukes til å definere en RDF-mal for hvordan den endelige RDF-grafen skal se ut. For å beskrive RDF-malen brukes RDF-serialiseringen TriG [18]. Malen som brukes til konverteringen består av to deler. Den ene delen beskriver templategrafen som brukes som mal til den ferdige RDF-grafen, mens den andre delen angir hvilke(t) regneark som skal brukes, og hvordan templategrafen skal flyttes rundt i regnearket for å utføre transformeringen.

Figur 5.4 på neste side viser et eksempel på en templategraf. Det kan være flere templategrafer per dokument hvor hver graf navngis, og i eksempelet heter grafen `DSFLoad`. Videre viser figuren hvordan strenger og verdier fra regnearket kan konkateneres. Eksempelet viser hvordan vi oppretter et navngitt subjekt, som består av en URI konkatenert med innholdet i cellen A2. Vi ser også hvordan funksjoner kan brukes til å behandle innholdet i en bestemt celle. I dette tilfellet brukes funksjonen `URLENCODE` som transformerer innholdet av cellen til et format som kan overføres over HTTP. XLWrap inneholder en rekke funksjoner og det er også mulig å definere egne funksjoner. Figuren viser hvordan funksjonen `DATE` kan brukes til å konvertere en literal til en typet literal av typen `xsd:date`. Literaler blir definert på følgende måte: `"A2"^^xl:expr` hvor `"A2"` definerer hvilken celle verdien skal hentes fra.

Figur 5.5 på neste side viser hvordan den delen av dokumentet som angir hvilke(t) regneark som skal brukes, og hvordan templategrafen skal flyttes rundt er definert. Predikatet `xl:template` kan brukes mange ganger per dokument, og her defineres det hvilken fil som skal konverteres og hvilken

```
@prefix xl:   <http://purl.org/NET/xlwrap#> .
@prefix :     <http://sws.ifi.uio.no/data/dsf/henriwi/> .
@prefix dsfv: <http://sws.ifi.uio.no/vocab/dsf/henriwi/dsf#> .

:DSFLoad {
  [ xl:uri "'http://sws.ifi.uio.no/data/dsf/henriwi/person/' &
    URLENCODE(A2)"^^xl:Expr ]
  rdf:type                dsfv:Person ;
  dsfv:fodselsnummer      "A2"^^xl:Expr ;
  dsfv:datoForDoedsfall   "DATE(B2)"^^xl:Expr ; ;
}
```

Figur 5.4: Eksempel på templategraf i TriG-syntaks som brukes av XLWrap.

templategraf som skal anvendes på denne filen. Predikatet `xl:transform` brukes for å angi hvordan templategrafen skal flyttes rundt i regnearket under konverteringen. Klassen `:RowShift` betyr at templategrafen skal anvendes på én og én rad.

```
@prefix xl:   <http://purl.org/NET/xlwrap#> .
@prefix :     <http://sws.ifi.uio.no/data/dsf/henriwi/> .
@prefix dsfv: <http://sws.ifi.uio.no/vocab/dsf/henriwi/dsf#> .

{
  []  rdf:type xl:Mapping ;
      xl:template [
        xl:fileName "file:dump.csv" ;
        xl:templateGraph :DSFLoad ;
        xl:transform [ a xl :RowShift ]
      ].
}
```

Figur 5.5: Eksempel på prosesseringsinstrukser som brukes av XLWrap.

XLWrap tilbyr en serverimplementasjon som kan benyttes til konverteringen, men har også et Java API som sammen med Jena gir et kraftig verktøy for konvertering av tabulære data til RDF. DSFConverter bruker dette API-et til å utføre konverteringen fra CSV til RDF.

5.6 Protégé

Til å utvikle ontologiene har editoren Protégé[3] blitt benyttet. Dette er en åpen-kildekode editor som blant annet gjør det mulig å lage og editere OWL-ontologier i et grafisk grensesnitt. Andre ontologier kan også importeres, enten fra filsystemet eller over HTTP og dermed integreres. Videre kommer Protegé med flere innebygde resonnerere slik at en ontologi kan sjekkes for konsistens i utviklingsmiljøet.

[3] `http://protege.stanford.edu/`

5.7 Oppsummering

Dette kapittelet har gått gjennom implementasjonen av demonstratoren og de forskjellige komponentene som er valgt. For lagring, forvalting og distribuering av RDF-dataene er Joseki benyttet. Dette er en SPARQL-server som gir muligheter for å stille SPARQL-spørringer over HTTP mot dataene. Jena sine API-er blir brukt til blant annet lagring av dataene, og vi har valgt en minnemodell som lagringsmodell. For å bidra til å automatisere konverteringsprosessen fra rådataene til RDF har vi laget et program som, basert på en rekke instillinger, konverterer en eller flere dumpfiler til RDF, ved først å konvertere rådataene til CSV og videre til RDF med verktøyet XlWrap. Programmet har også mulighet for å utføre resonnering over dataene basert på en ontologi, og bruker Pellet til å utføre selve resonneringen.

SPARQL-endepunktet ligger åpent tilgjengelig på følgende URL: `http://sws.ifi.uio.no/sparql/dsf/henriwi`, og kan benyttes ved å sende en SPARQL-spørring som argument til URL-en. Det er også satt opp et generisk webgrensesnitt som gjør det mulig å skrive SPARQL-spørringer, og denne websiden inneholder også noen eksempelspørringer som kan stilles til demonstratoren. Denne siden finnes på følgende URL: `http://sws.ifi.uio.no/sparqler/dsf/henriwi`.

Det RESTfulle grensesnittet er også åpent tilgjengelig og følgende side gir en oversikt over noen av mulighetene dette grensesnittet gir, og kan besøkes for å utforske dataene: `http://sws.ifi.uio.no/data/dsf/henriwi/`. I tillegg ligger rådataene, RDF-dataene, ontologiene og programmet DSFConverter tilgjengelig for nedlasting på følgende side: `http://sws.ifi.uio.no/project/dsf/henriwi/`.

Kapittel 6

Resultater og gevinster

Dette kapitlet beskriver og viser gjennom flere eksempler mulighetene og gevinstene som demonstratoren beskrevet i kapittel 4 og 5 gir. Først viser kapittelet hvordan demonstratoren oppfyller kravene til distribusjon, spørring og modell for dataintegrasjon beskrevet i seksjon 2.6 på side 13, før vi argumenterer for hvordan demonstratoren konkret løser en del av problemene beskrevet i seksjon 2.5 på side 10.

6.1 Distribusjon

Kapittel 2 konkluderte med at en ny distribusjonsmodell for Folkeregisteret må være fleksibel og gjøre det mulig for flere aktører å hente ut oppdaterte data, samtidig som den nye modellen bør muliggjøre enkel integrering av andre heterogene kilder. Ved å konvertere rådataene til RDF vil dataene bli tilgjengeliggjort på et «integrerbart» format, og dermed bidra til enklere dataintegrasjon. Videre vil ulike mekanismer for å hente ut dataene som et SPARQL-endepunkt og et RESTfullt grensesnitt gjøre det mulig å konsumere dataene for flere aktører og applikasjoner.

RDF bruker URI-er til å identifisere entiteter, og vi får dermed identifikatorer som har et globalt omfang. Dette i motsetning til dagens løsning som bruker identifikatorer med et lokalt omfang, som kun er unike innenfor databasen hvor dataene ligger lagret. Videre vil ikke bare entitetene ha en utvetydig identifikator med et globalt omfang, men også relasjonene mellom de ulike entitetene som også er identifisert av URI-er. Dette er en av de store styrkene til RDF, det å kunne danne eksplisitte relasjoner mellom dataelementene hvor relasjonen i tillegg kan ha en formell beskrivelse uttrykt i en ontologi.

Med globale identifikatorer og det faktum at en RDF-graf er en mengde med tripler, kan vanlige mengdeoperasjoner utføres. Dermed er det mulig å slå sammen flere RDF-grafer ved å ta unionen av disse mengdene. I tillegg vil semantiske teknologiers antagelse om NUNA, samt muligheten

for eksplisitte identitetsutsagn, sørge for at to entiteter med forskjellig identifikator verken er like eller ulike før dette er eksplisitt definert. Dette skiller seg klart fra tradisjonelle teknologier som antar UNA, hvor standardoppførselen er at entiteter med forskjellig identifikator blir tolket til å være ulike. NUNA fører til en enklere sammenslåing av datakilder enn med tradisjonell databaseteknologi, hvor en tidkrevende og kompleks transformasjon er nødvendig.

Ved å distribuere heterogene datakilder på en standardisert måte hvor alle datasettene har samme datamodell blir det enklere og mindre tidkrevende å kombinere disse, enn hvis det hadde ligget forskjellige lagringsmodeller til grunn. Dette er en av grunnene til at det er vanskelig å integrere folkeregisterdata og SED-data, fordi SED er XML, mens det ligger en relasjonell database til grunn for TPS hvor folkeregisterdataene ligger lagret. Vi har vist hvordan både tabulære data og data på en trestruktur i form av XML kan distribueres som RDF, og dermed gjøre det enklere å integrere disse datakildene fordi de har den samme underliggende datamodellen. Hvis datakildene skulle kombineres uten å forandre dataformatet, ville dette krevd egen programvare tilpasset hvert enkelt dataformat og kilde. Som vi også har vist kan det lages en automatisk konverteringsprosess som gjør at dataene kan ligge urørt i sin originale kilde, og kun eksponeres til sluttbrukere som RDF. Dette fører til at eksisterende og godt fungerende systemer som brukes til datalagring og forvalting internt i SKD fortsatt kan brukes, mens det kun er integrasjonslaget som er nødvendig å eksponere som RDF. Tradisjonelle teknologier, som datavarehus, kan kombinere relasjonelle databaser, men det er ikke mulig å anvende de samme teknikkene til å kombinere for eksempel XML eller Excelark. Med semantisk teknologi er det derimot mulig å integrere data fra mange kilder, uavhengig av format og lokasjon [24].

Det kan også lages lenker fra folkeregisterdataene til andre RDF-distribuerte datasett. Dette kan enten være andre datasett innenfor brannmurene til SKD, eller offentlige datasett. Dette er blant annet vist ved at folkeregisterdataene har koblinger til eksisterende vokabularer (`?hvor` og FOAF) og andre datasett (geonames og SED). Dette fører til at folkeregisterdataene kan bli utvidet og sluttbrukere vil få tilgang til langt mer informasjon enn det som er mulig ved å bruke tradisjonelle relasjonelle databaser, hvor det ikke er mulig å koble sammen datasett på en lignende måte. Ny informasjon kan dermed tilegnes ved at nye sammenhenger mellom datasett oppdages, sammenhenger som ikke har vært mulig å se tidligere når dataene har ligget lagret isolert uten noen form for sammenkobling.

En annen fordel med RDF er at det er skjemaløst, noe som medfører at det er enkelt å legge til mer data (flere tripler) uten å måtte forandre på det eksisterende datasettet. I relasjonelle databaser ville dette vært en utfordring fordi alle entiteter baserer seg på, og er tett knyttet til, det samme underliggende skjema. I tillegg er relasjonelle databaseskjemaer statiske og lite fleksible, noe som medfører at det er kostbart og vanskelig å endre skjemastrukturen. Det er enklere å utføre endringer i en ontologi

sammenlignet med et databaseskjema [24], og det er ingenting i veien for at en ontologi kan oppdateres eller til og med byttes ut, uten at det er nødvendig å endre på de faktiske dataene. Det er heller ikke nødvendig å måtte utvikle ontologien før selve RDF-dataene kan distribueres, mens relasjonelle databaseskjemaer må være ferdig utviklet før en database kan fylles med de faktiske dataene [24].

RDF er en datamodell, og ikke et eget dataformat, men tilbyr mange lagrings- og distribusjonsformater. Dette gjør det mulig for en rekke forskjellige applikasjoner å benytte seg av dataene, da resultatformatet som returneres kan bestemmes av sluttbruker. RDF og OWL er viktig for å kunne bidra til en fleksibel distribusjonsløsning som bidrar til enklere dataintegrasjon, men det er ikke sikkert det er nødvendig å eksponere kompleksiteten i disse språkene direkte til sluttbrukeren. Derfor vil det være bedre, som demonstratoren gjør, å kunne skjule kompleksiteten til RDF og OWL, og eksponere dataene på formater som eksisterende applikasjoner allerede benytter seg av og kan håndtere.

RDF har også muligheten til å samle for eksempel en etats data i en felles navngitt graf. Ved hjelp av disse navngitte grafene kan et felles lagringssted benyttes på tvers av ulike datakilder, og dette gjør det mulig å sette tilgangskontroll på et subsett av datasettet. Dette gjør det igjen mulig å kunne tilpasse distribusjonsmodellen for å kunne brukes av flere aktører, samtidig som det er mulig å sikre at alle aktørene jobber mot de samme dataene.

6.2 Spørringer

For å kunne lage en fleksibel distribusjonsløsning er det nødvendig med et spørregrensesnitt som fungerer på tvers av applikasjoner og ikke stiller spesielle krav til programmeringsspråk eller rammeverk. I tillegg har NAV et ønske om å kunne stille forskjellige kontrollspørsmål mot både DSF- og SED-dataene, og det er derfor viktig at det er mulig å stille vilkårlige spørringer uten at disse må være satt opp på forhånd. Demonstratoren er satt opp med et SPARQL-endepunkt som gjør at SPARQL-spørringer kan stilles mot de RDF-distribuerte dataene, samt et RESTfullt grensesnitt for å kunne gjøre oppslag på enkeltressurser.

6.2.1 SPARQL-spørringer

SPARQL-spørringer kan enten utføres av brukere gjennom et generisk webgrensesnitt hvor det er mulig å manuelt skrive SPARQL-spørringer, eller spørringene kan integreres i applikasjoner som benytter HTTP til å kommunisere med SPARQL-endepunktet. Det siste er særlig aktuelt for spørringer som blir stilt ofte, men det er likevel en fordel å ha mulighet til å stille spørringer uten at disse er satt opp på forhånd, fordi dette kan gjøre

det mulig å få svar på spørsmål som man ikke har tenkt på at er ønskelig å kunne stille.

Resten av denne seksjonen viser eksempler på flere typer forskjellige spørringer som er mulig å stille til demonstratoren. Vi argumenterer for hvorfor det er viktig at spørringene bør kunne stilles, samt hva slags resultat spørringene gir. I alle eksemplene antar vi at alle prefikser er satt opp korrekt.

Hent informasjon om en person med et bestemt fødselsnummer Denne spørringen henter ut informasjon knyttet til en person med en bestemt identifikator, personens fødselsnummer. Legg også merke til hvordan nøkkelordet `BIND` brukes til å sette verdien til en variabel til kombinasjonen av flere andre verdier. I spørringen nedenfor settes flere adresseverdier sammen til variabelen `?adresse` ved hjelp av den eksterne funksjonen `fn:concat`.

```
SELECT ?slektsnavn ?fornavn ?mellomnavn ?adresse WHERE {
  dsfp:02088600110 dsfv:fornavn ?fornavn ;
                   dsfv:mellomnavn ?mellomnavn ;
                   dsfv:slektsnavn ?slektsnavn ;
                   dsfv:harAktuellAdresse [
                     dsfv:adressenavn ?adressenavn ;
                     dsfv:postnummer ?postnummer ;
                   ] .
  BIND (fn:concat(?adressenavn, ", ", ?postnummer) AS ?adresse)
}
```

Hent personer basert på forskjellige koder Denne spørringen viser hvordan koder kan brukes til å begrense resultatet. Spørringen henter ut info om en persons utvandring for alle personer med statuskode lik `utvandret` og spesifisert registreringstype til sperret adresse, fortrolig.

```
SELECT ?person ?land ?dato
WHERE {
  ?person a dsfv:Person;
  dsfv:harUtvandring [
    dsfv:harUtvandretTilLand ?land;
    dsfv:flyttedatoForUtvandring ?dato .
  ];
  dsfv:harStatuskode registreringsstatus:utvandret;
  dsfv:harSpesifisertRegtype
    spesifisertRegtype:sperretAdresseFortrolig .
}
```

Spørringen over bruker verdien av kodene, men det er også mulig å bruke kodeverdiene direkte. Følgende spørring vil gi samme resultat som den over:

```
SELECT ?person ?land ?dato
WHERE {
  ?person a dsfv:Person;
```

```
    dsfv:harUtvandring [
      dsfv:harUtvandretTilLand ?land;
      dsfv:flyttedatoForUtvandring ?dato .
    ];
    dsfv:harStatuskode registreringsstatus:kode-3;
    dsfv:harSpesifisertRegtype
      spesifisertRegtype:kode-7 .
  }
```

Søk etter navn Denne spørringen viser noen av mulighetene for å søke etter data ved hjelp av regex, og spørringen bruker et ekstern funksjonsbibliotek som tilbys av spørreprosessoren ARQ. Spørringen henter alle personer som har et fornavn som starter med `HE`, og et slektsnavn som starter med `S`. Dette vil være relevant hvis det for eksempel er usikkerhet knyttet til stavemåten til navnet for personen det søkes etter.

```
SELECT ?person ?navn WHERE {
  ?person dsfv:fornavn ?fornavn ;
          dsfv:slektsnavn ?slektsnavn ;
  BIND (fn:concat(?fornavn, " ",?slektsnavn) AS ?navn)
  FILTER (fn:matches(?fornavn,"^HE") &&
          fn:matches(?slektsnavn,"^S"))
}
```

Negasjon Siden demonstratoren støtter SPARQL 1.1, er det mulig å stille spørringer som anvender negasjon. For eksempel vil følgende spørring hente ut identifikatoren til alle personer som *ikke* har en statuskode:

```
SELECT * WHERE  {
  ?p a dsfv:Person .
  FILTER NOT EXISTS {
    ?p dsfv:harStatuskode ?statuskode
  }
}
```

Mulighet til å stille spørringer med negasjon er viktig for å kunne kontrollere dataene, og er en viktig funksjon i det å stille kontrollspørringer. For eksempel kan en forsikre seg om at en person med et gitt fødselsnummer ikke eksisterer i databasen.

Spørring på tvers av DSF og SED For å skille mellom DSF- og SED-dataene har datasettene blitt tilordnet hver sin navngitte graf. DSF-dataene har blitt tilordnet standard-grafen, det vil si at det er disse dataene det blir spurt mot når det ikke angis en navngitt graf i spørringen. SED-dataene har blitt tilordnet følgende navngitte graf: `http://sws.ifi.uio.no/data/dsf/henriwi/sed`. Spørringen nedenfor viser hvordan en spørring kan stilles på tvers av DSF- og SED-dataene. Den returnerer et resultat hvis det finnes noen i DSF som har likt fornavn, etternavn og fars og mors fornavn som noen i SED.

```
SELECT * WHERE {
  ?dsfPerson  a dsfv:Person ;
              dsfv:fornavn ?fornavn ;
              dsfv:slektsnavn ?slektsnavn ;
              dsfv:harFar/dsfv:fornavn ?fornavnFar ;
              dsfv:harMor/dsfv:fornavn ?fornavnMor .

  GRAPH <http://sws.ifi.uio.no/data/dsf/henriwi/sed> {
    ?sedPerson  a sedv:PersonWithoutPIN ;
                sedv:forenames ?fornavn ;
                sedv:familyName ?slektsnavn ;
                sedv:forenameofFather ?fornavnFar ;
                sedv:forenameofMother ?fornavnMor .
  }
}
```

Spørring for å sette to personer som like Som vi skal vise i seksjon 6.3.1 på side 100 kan resonnering anvendes til automatisk å fastslå at personer fra SED og DSF er like. Men dette er også mulig å utføre ved hjelp av SPARQL. Som vist i seksjon 4.6 på side 75 er to personer definert til å være like hvis de har samme personidentifikator. Men, som vi argumenterte for, hadde det vært ønskelig å kunne si at to personer er like hvis de for eksempel har samme navn og at foreldrene også har samme navn. På grunn av strukturen vi har på dataene kan ikke dette defineres i ontologien vår, men det er mulig å utføre dette med SPARQL som spørringen nedenfor viser. Spørringen legger til en trippel som sier at to personer er like hvis de har samme for- og etternavn, samt at foreldrene har samme fornavn og at faren har samme etternavn.

```
INSERT { ?dsfPerson owl:sameAs ?sedPerson . }
WHERE {
  ?dsfPerson  a dsfv:Person ;
              dsfv:fornavn ?fornavn ;
              dsfv:slektsnavn ?slektsnavn ;
              dsfv:harFar/dsfv:fornavn ?fornavnFar ;
              dsfv:harFar/dsfv:slektsnavn ?etternavnFar ;
              dsfv:harMor/dsfv:fornavn ?fornavnMor .

  GRAPH <http://sws.ifi.uio.no/data/dsf/henriwi/sed> {
      ?sedPerson  a sedv:PersonWithoutPIN ;
                  sedv:forenames ?fornavn ;
                  sedv:familyName ?slektsnavn ;
                  sedv:forenameofFather ?fornavnFar ;
                  sedv:forenameofMother ?fornavnMor ;
                  sedv:fatherFamilyNameAtBirth ?etternavnFar .
    }
}
```

Spørring på tvers av DSF og DUF Den nye distribusjonsmodellen til DSF kan potensielt gi muligheter utover det å hente ut folkeregisterdata

og bidra til å løse identifiseringsproblemet. Hvis vi tenker litt frem i tid er det mulig å se for seg at vi har tilgang til flere registre som er distribuert på samme måte som DSF og SED. I dette eksemplet viser vi hvordan vi kan skrive en spørring som spenner over data fra DSF og DUF. Vi tenker oss at DUF inneholder mer informasjon om når en person innvandret til Norge enn det som ligger lagret i DSF. I spørringen antar vi at `dufv:` er prefikset for vokabularet som beskriver dataene i DUF, og at dataene er lagret under den navngitte grafen `http://sws.ifi.uio.no/data/henriwi/dsf/duf`. Spørringen viser hvordan DUF inneholder informasjon om adressen i landet personen utvandret fra, og viser hvordan informasjon fra flere kilder kan kombineres for å hente ut mer informasjon om en person enn det som ligger lagret i ett register.

```
SELECT * WHERE {
  dsf:01010750160 dsfv:navn ?navn ;
    dsfv:harInnvandring [
      dsfv:harInnvandretFraLand ?land ;
      dsfv:regdatoForInnvandring regDato? ;
      dsfv:flyttedatoForInnvandring ?flytteDato .
    ] .
  GRAPH <http://sws.ifi.uio.no/duf> {
    dsf:01010750160 dufv:harAdresseIUtvandretLand [
        dufv:gatenavn ?gatenavn ;
        dufv:postnummer ?postnummer
      ] .
  }
}
```

6.2.2 RESTfullt grensesnitt

Som vist gir SPARQL muligheter for å stille komplekse, vilkårlige spørringer mot dataene. Men SPARQL vil for mange fremstå som komplekst, og denne kompleksiteten bør derfor kunne skjules. Demonstratoren tilbyr derfor også et RESTfull grensesnitt ved siden av SPARQL-endepunktet, og ved hjelp av dette grensesnittet kan enkeltressurser slås opp og saksbehandlere kan raskt fastslå om det for eksempel eksisterer en person med et gitt fødselsnummer eller ikke. I tillegg kan lister over forskjellig type entiteter returneres, for eksempel alle foreldre eller alle ektefeller. Ved å skjule detaljene til SPARQL-endepunktet på denne måten tilbys det et grensesnitt som er enkelt å ta i bruk, men som samtidig kan utvides til et kraftig grensesnitt ved å konfigurere komplekse spørringer knyttet opp mot ulike URL-er.

Tjenesten er satt opp med en rekke endepunkter[1] for å vise ulike muligheter som et slikt grensesnitt gir.

[1] Linked Data API kaller en URL som har en SPARQL-spørring bundet opp mot seg for et endepunkt.

Slå opp på en enkeltressurs basert på fødselsnummeret En HTTP-GET forespørsel til følgende URL vil gi all informasjon knyttet til personen med fødselsnummer `02088600110`:

`http://sws.ifi.uio.no/data/dsf/henriwi/person/02088600110`

Hvis det ikke eksisterer en person med dette fødselsnummeret, vil klienten motta en HTTP 404 respons tilbake, og dermed raskt kunne fastslå at det ikke eksisterer en person med dette fødselsnummeret i datasettet.

Adresse, innvandring, innflytting og utvandring til en person Det er også mulig å navigere seg videre i RDF-grafen knyttet til en person ved å tilføye `/adresse` etter fødselsnummeret. Dette vil føre til at all adresseinformasjon knyttet til den aktuelle personen returneres. Det samme gjelder for innvandring, innflytting og utvandring. Hvis den aktuelle personen ikke har en av disse entitetene knyttet til, seg vil en HTTP 404 respons returneres til klienten.

`http://sws.ifi.uio.no/data/dsf/henriwi/person/02088600110/adresse`

Gi en liste over alle personer Ved å «kutte» bort fødselsnummeret vil endepunktet returnere en liste over alle personer.

`http://sws.ifi.uio.no/data/dsf/henriwi/person`

Gi en liste over individer av en bestemt type Ved å forandre `person` i eksempelet over til for eksempel `far` vil endepunktet returnere en liste over alle fedre i datasettet, eller mer eksakt: alle individer av typen `dsfv:Far`. Dette gjelder også andre typer som `dsfv:Ektefelle` `dsfv:Mor` og `dsfv:Barn`.

`http://sws.ifi.uio.no/data/dsf/henriwi/far`

Grensesnittet støtter en rekke forskjellige resultatformater som bestemmes av forespørselen som sendes av klienten. Dette gjør at resultatformatet kan tilpasses avhengig av klienten som spør etter dataene. For eksempel vil følgende URL returnere resultatet på Turtle-format:

`http://sws.ifi.uio.no/data/dsf/henriwi/person/02088600110.ttl`

Noe som også er verdt å nevne er at ved å distribuere dataene gjennom et slikt RESTfullt endepunkt, som også oppfyller kravene til lenkede data, vil applikasjoner som kan brukes til å utforske lenkede data, også kunne brukes til å utforske folkeregisterdataene. For eksempel kan URIBurner[2] anvendes til å utforske dataene.

[2] `http://uriburner.com/`

6.3 Modell for dataintegrasjon

Dataene distribuert som RDF gir muligheter for å konsumere dataene gjennom et felles grensesnitt, og gjør det enklere å kombinere data fra heterogene kilder. Som nevnt i seksjon 3.2.1 på side 20 tilfører RDF-dataene i seg selv ikke noen mening eller bakgrunnsinformasjon, men ved å legge til ontologien som er utviklet vil bakgrunnsinformasjon bli tilført dataene ved at begreper og relasjonene mellom disse er eksplisitt definert. Som vi skal se senere kan denne informasjonen brukes til å utføre resonnering over DSF-dataene generelt og også i kombinasjon med SED-dataene spesielt. I tillegg til å tilføre bakgrunnsinformasjon, vil ontologiene fungere som dokumentasjon, og kan brukes til å innhente metainformasjon og definisjoner knyttet til de ulike begrepene og rollene i modellen.

Anta at vi ønsker å få en tekstlig beskrivelse av rollen `skolekrets`. Med dagens system har man vært nødt til å gå i dokumentasjonen for å finne dette, men siden DSF-ontologien inneholder denne informasjonen, kan følgende SPARQL-spørring stilles for å hente ut den formelle beskrivelsen av rollen:

```
SELECT ?beskrivelse WHERE {
  dsfv:skolekrets rdfs:comment ?beskrivelse .
}
```

Resultatet av denne spørringer vil være følgende:

beskrivelse
"Skolekretsen til personens adresse"

I tillegg til å skrive SPARQL-spørringer for å hente ut denne type informasjon, kan en HTTP GET-forespørsel sendes til URL-en som beskriver et begrep eller en rolle. For eksempel vil en HTTP-forespørsel til URL-en: `http://sws.ifi.uio.no/vocab/dsf/henriwi/dsf#skolekrets` returnere den delen av ontologien som beskriver `skolekrets`.

En annen interessant spørring vil være å kunne få ut verdien til en aktuell kode. Følgende spørring vil skrive ut alle koder og verdier samt en beskrivelse til koden av typen `Personkode`, sortert på kode. Resultatet av spørringen er vist i tabell 6.1 på neste side.

```
SELECT ?kode ?verdi ?beskrivelse WHERE {
  ?verdi  a dsfv:Personkode ;
          rdfs:comment ?beskrivelse ;
          owl:sameAs ?kode .
} ORDER BY ?kode
```

I tillegg til å tilføre bakgrunnsinformasjon til dataene og fungere som dokumentasjon, vil modellen brukes som et spørregrensesnitt over dataene ved at begrepene i ontologien benyttes til å utforme SPARQL-spørringer. På bakgrunn av disse punktene vil modellen kunne anvendes aktivt i

kode	verdi	beskrivelse
`dsf:personkode#kode-1`	`dsf:personkode#referanseperson`	"Referanseperson"
`dsf:personkode#kode-2`	`dsf:personkode#giftKvinneYngstePartner`	"Gift kvinne/yngste partner som bor sammen med ektefelle/-partneren"
`dsf:personkode#kode-3`	`dsf:personkode#barnHosForeldre`	"Barn som bor sammen med foreldre"

Tabell 6.1: Resultatet av spørringen som henter informasjon om alle personkoder.

den daglige bruken av løsningen, og ikke bare være en modell som for eksempel kun blir brukt under designet av løsningen. Som vi har vist kan ontologier brukes til å relatere begreper på tvers av domener og datakilder, og dermed sy sammen heterogene datakilder på en enkel og elegant måte. Dette gjør også at det ikke er nødvendig å utvikle én felles modell som alle datakildene må mappes mot. Med OWL kan det lages separate modeller for hvert domene eller system, og senere koble disse sammen ved å relatere begreper fra én ontologi opp mot en annen. I tillegg er det ikke nødvendig at alle begrepene relateres, men kun de begrepene som faktisk er felles for de ontologiene som kobles sammen. Et siste poeng er at både dataene, modellen og dermed dokumentasjonen har samme underliggende datamodell, nemlig tripler, og kan dermed lagres og forvaltes av de samme systemene.

6.3.1 Resonnering

Modellen i seg selv gir en gevinst ved å tilføre bakgrunnsinformasjon, samt fungere som dokumentasjon og et spørregrensesnitt. Men den virkelige gevinsten av en slik modell kommer til uttrykk når resonnering anvendes sammen med dataene. Ved å kombinere datakilder og deretter anvende resonnering, kan kunnskap som ikke er lagret eksplisitt i noen av datakildene utledes. Dette kan føre til at nye sammenhenger og ny kunnskap oppdages, kunnskap som ikke er mulig å oppdage ved hjelp av tradisjonelle teknologier, som ikke har noen form for resonnering. Dette gjelder særlig for identifiseringsproblemet hvor vi skal vise hvordan to personer eksplisitt kan settes til å være samme person. Resonnering er også attraktivt generelt fordi det kan brukes til å sjekke om en ontologi er konsistent, samt at det utvider datasettet og legger til implisitt informasjon som ikke er eksplisitt lagret. Ontologien vår anvender også andre eksisterende vokabularer, og sammenhengen mellom vår ontologi og de eksisterende vokabularene vil «sildre» ned på dataene, og på den måten også utvide datasettet.

Resonnering påvirker dataene en rekke steder i DSF-dataene. Et enkelt eksempel er hvordan resonnereren brukes til å utlede kodeverdiene basert på koden. Nedenfor vises et uttrekk fra DSF-dataene av en person som

har en statuskode lik «1» og foreldreansvarskode lik «D». Basert på disse dataene er det ingenting som sier hva verdien til koden er.

```
dsfp:01010750160
    dsfv:harForeldreansvar  dsfv:foreldreansvar#kode-D ;
    dsfv:harStatuskode      dsfv:registreringsstatus#kode-1 ;
```

Den aktuelle delen av ontologien ser slik ut:

```
dsfv:foreldreansvar#kode-D
  owl:sameAs dsvf:foreldreansvar#delt .
dsfv:registreringsstatus#kode-1
  owl:sameAs dsfv:registreringsstatus#bosatt .
```

Resonnereren vil utlede at siden det eksisterer en trippel i ontologien som sier at koden «D» har verdi «delt», og koden «1» har verdi «bosatt», vil det samme uttrekket etter resonneringen se slik ut:

```
dsfp:01010750160
    dsfv:harForeldreansvar  dsfv:foreldreansvar#kode-D,
                            dsfv:foreldreansvar#delt;
    dsfv:harStatuskode      dsfv:registreringsstatus#kode-1,
                            dsfv:registreringsstatus#bosatt ;
```

I tillegg til å utlede informasjon om koder og deres verdier legges det til informasjon om typene til individene i datasettet. Her benyttes det blant annet resonnering over domene og verdiområde, hvor domene og verdiområde er definert for alle rollene i ontologien. For eksempel er domene til rollen `dsf:harBarn` satt til klassen `dsf:Forelder`. Dette fører til at hvis vi har et individ `a` og et individ `b`, og det går en `dsf:harBarn`-relasjon mellom `a` og `b`, vil følgende trippel bli lagt til:

```
  a rdf:type dsf:Forelder .
```

Figur 6.1 på neste side viser hvordan personen med fødselsnummer `01096000371` representeres før resonneringen er utført, og figur 6.2 på side 103 viser samme person etter at resonneringen er utført, hvor de nye elementene er uthevet. Legg spesielt merke til at det er lagt til tripler for alle verdiene til kodene, og at personen har fått lagt til typene `dsfv:Ektefelle`, `dsfv:Forelder` og `dsfv:Far`. I tillegg har adressen fått lagt til typen `dsf:MatrikkelAdresse` og `http://vocab.lenka.no/hvor#Adresse`. Sistnevnte viser hvordan informasjon fra ontologien «sildrer» ned på dataene, og bidrar til å utvide datasettet ved å lenke dataelementer til andre datasett, i dette tilfellet et annet vokabular.

Som vist i seksjon 4.6 på side 75, har flere begreper fra SED-ontologien blitt koblet opp mot begreper i DSF-ontologien. Deretter ble predikatet `owl:hasKey` brukt til å definere at to personer var like hvis de hadde lik personidentifikator. Eksempelet nedenfor viser den aktuelle delen av ontologien som beskriver dette serialisert til Turtle.

```
@prefix dsfp: <http://sws.ifi.uio.no/data/dsf/henriwi/person/> .
@prefix sivilstand: <http://sws.ifi.uio.no/vocab/dsf/henriwi/sivilstand#> .
@prefix aarsakskode: <http://sws.ifi.uio.no/vocab/dsf/henriwi/aarsakskode#> .
@prefix spesRegtype: <http://sws.ifi.uio.no/vocab/dsf/henriwi/spesifisertRegtype#> .
@prefix regStatus: <http://sws.ifi.uio.no/vocab/dsf/henriwi/registreringsstatus#> .
@prefix personkode: <http://sws.ifi.uio.no/vocab/dsf/henriwi/personkode#> .
@prefix adressetype: <http://sws.ifi.uio.no/vocab/dsf/henriwi/adressetype#> .

dsfp:01096000371
  a dsfv:Person;
  dsfv:aarsakskodeForAdresse aarsakskode:kode-99;
  dsfv:aarsakskodeForNavn aarsakskode:kode-02;
  dsfv:familienummer "01096000371";
  dsfv:fodselsnummer "01096000371";
  dsfv:foedestedKommLand "9105";
  dsfv:fornavn "TERJE";
  dsfv:harAktuellAdresse [
    a dsfv:AktuellAdresse;
    dsfv:adressenavn "BOKS 6300, ETTERSTAD";
    dsfv:flyttedatoForAdresse "2009-08-01"^^xsd:date;
    dsfv:gateGaard "00019";
    dsfv:harAdressetype adressetype:kode-M;
    dsfv:husBruk "0019";
    dsfv:kommunenummer "0019";
    dsfv:postnummer "0603";
    dsfv:regdatoForAdresse "2009-08-01"^^xsd:date;
    dsfv:valgkrets "0001"
  ];
  dsfv:harInnvandretFra [
    dsfv:flyttedatoForInnvandring "2001-06-10"^^xsd:date;
    dsfv:harInnvandretFraLand
      <http://sws.ifi.uio.no/vocab/dsf/henriwi/land#kode-105>;
    dsfv:regdatoForInnvandring "2001-06-10"^^xsd:date
  ];
  dsfv:harBarn dsfp:02020650330, dsfp:02058300279, dsfp:02060150225,
    dsfp:02088600110;
  dsfv:harEktefelle dsfp:02086300292;
  dsfv:harPersonkode personkode:kode-1;
  dsfv:harSivilstand sivilstand:kode-2;
  dsfv:mellomnavn "PSA";
  dsfv:harSpesifisertRegtype spesRegtype:kode-0;
  dsfv:harStatuskode regStatus:kode-1;
  dsfv:skolekrets "0001";
  dsfv:slektsnavn "TOPSTAD" .
```

Figur 6.1: En person representert før resonnering.

```
@prefix dsfp: <http://sws.ifi.uio.no/data/dsf/henriwi/person/> .
@prefix sivilstand: <http://sws.ifi.uio.no/vocab/dsf/henriwi/sivilstand#> .
@prefix aarsakskode: <http://sws.ifi.uio.no/vocab/dsf/henriwi/aarsakskode#> .
@prefix spesRegtype: <http://sws.ifi.uio.no/vocab/dsf/henriwi/spesifisertRegtype#>
.
@prefix regStatus: <http://sws.ifi.uio.no/vocab/dsf/henriwi/registreringsstatus#>
.
@prefix personkode: <http://sws.ifi.uio.no/vocab/dsf/henriwi/personkode#> .
@prefix adressetype: <http://sws.ifi.uio.no/vocab/dsf/henriwi/adressetype#> .
@prefix hvor: <http://vocab.lenka.no/hvor#> .

dsfp:01096000371
      a dsfv:Person, dsfv:Ektefelle, dsfv:Forelder, dsfv:Far, owl:Thing;
      dsfv:aarsakskode aarsakskode:kode-99, aarsakskode:korreksjonsmelding,
         aarsakskode:kode-02, aarsakskode:innvandring;
      dsfv:aarsakskodeForNavn aarsakskode:kode-99, aarsakskode:korreksjonsmelding;
      dsfv:aarsakskodeForAdresse aarsakskode:kode-02, aarsakskode:innvandring;
      dsfv:erFarTil dsfp:02020650330, dsfp:02058300279, dsfp:02060150225,
        dsfp:02088600110;
      dsfv:identifikator "01096000371";
      dsfv:familienummer "01096000371";
      dsfv:fodselsnummer "01096000371";
      dsfv:foedestedKommLand "0019";
      dsfv:fornavn "TERJE";
      dsfv:harAktuellAdresse [
        a dsfv:AktuellAdresse, dsfv:MatrikkelAdresse, hvor:Adresse, owl:Thing;
        dsfv:adressenavn BOKS 6300, ETTERSTAD";
        dsfv:erAktuellAdresseFor dsfp:01010750160;
        dsfv:flyttedatoForAdresse "2009-08-01"^^xsd:date;
        dsfv:gateGaard "00019";
        dsfv:harAdressetype adressetype:kode-M, adressetype:matrikkeladresse;
        dsfv:harKode adressetype:kode-M, adressetype:matrikkeladresse;
        dsfv:husBruk "0019";
        dsfv:kommunenummer "0019";
        hvor:kommunenr "0019" ;
        dsfv:postnummer "0603";
        hvor:postnummer "0603" ;
        dsfv:regdatoForAdresse "2009-08-01"^^xsd:date;
        dsfv:valgkrets "0001";
      ];
      dsfv:harInnvandretFra [
        a dsfv:Innvandring, owl:Thing  ;
        dsfv:flyttedatoForInnvandring "2001-06-10"^^xsd:date ;
        dsfv:harInnvandretFraLand
          <http://sws.ifi.uio.no/vocab/dsf/henriwi/land#kode-105> ;
        dsfv:regdatoForInnvandring "2001-06-10"^^xsd:date
      ];
      dsfv:harBarn dsfp:02020650330, dsfp:02058300279, dsfp:02060150225,
        dsfp:02088600110;
      dsfv:harEktefelle dsfp:02086300292;
      dsfv:harKode sivilstand:kode-2, sivilstand:gift, regStatus:kode-1,
        regStatus:bosatt, spesRegtype:kode-0, spesRegtype:vanligBosatt,
        personkode:kode-1, personkode:referanseperson;
      dsfv:harPersonkode personkode:kode-1, personkode:referanseperson;
      dsfv:harSivilstand sivilstand:kode-2, sivilstand:gift;
      dsfv:harSpesifisertRegtype spesRegtype:kode-0, spesRegtype:vanligBosatt;
      dsfv:harStatuskode regStatus:kode-1, regStatus:bosatt;
      dsfv:mellomnavn "PSA";
      dsfv:navn "TERJE", "PSA", "TOPSTAD";
      dsfv:personidentifikator "01096000371";
      dsfv:skolekrets "0001";
      dsfv:slektsnavn "TOPSTAD";
      owl:sameAs dsfp:01010750160 .
```

Figur 6.2: Personen i figur 6.1 på forrige side etter resonnering, hvor de nye elementene er uthevet.

```
dsfv:personidentifikator a owl:ObjectProperty .
dsfv:fodselsnummer a owl:ObjectProperty .
dsfv:pinReceivingInstitution a owl:ObjectProperty .
dsfv:pinSendingInstitution a owl:ObjectProperty .

dsfv:fodselsnummer rdfs:subPropertyOf dsfv:personidentifikator .
sedv:pinReceivingInstitution rdfs:subPropertyOf dsfv:personidentifikator .
sedv:pinSendingInstitution rdsf:subClassOf dsfv:personidentifikator .

dsfv:Person owl:hasKey ( dsfv:personidentifikator ) .
```

Videre vises et uttrekk av en person fra SED representert som RDF:

```
sedp:6 a sedv:PersonWithPin ;
    sedv:birthDate "1985-03-28"^^xsd:date  ;
    sedv:pinSendingInstitution "28038500106" ;
    sedv:hasGender gender:male ;
    sedv:forenames "JØRGEN" ;
    sedv:familyName "JOHANSEN" .
```

Til slutt vises et uttrekk av en person (som egentlig er den samme personen) fra DSF representert som RDF:

```
dsfp:28038500106
      a       dsfv:Person ;
      dsfv:harFar dsf:02033400198 ;
      dsfv:fornavn "JØRGEN" ;
      dsfv:harMor dsf:03088600230 ;
      dsfv:slektsnavn "JOHANSEN" ;
      dsfv:fodselsnummer "28038500106";
```

Siden både `dsfv:fodselsnummer` og `sedv:pinSendingInstitution` er subroller av `dsfv:personidentifikator`, vil individene over få lagt til en relasjon beskrevet av rollen `dsfv:personidentifikator` under resonnering. Deretter vil en resonnerer som blir anvendt på DSF- og SED-dataene fastslå at de personene som har samme personidentifikator må være samme person. I eksemplene har begge personene samme personidentifikator, og følgende trippel vil bli lagt til:

```
dsfp:28038500106 owl:sameAs sedp:6 .
```

6.4 Løsninger på utfordringer knyttet til dagens distribusjonsmodell

Seksjon 2.5 på side 10 pekte på flere utfordringer med dagens distribusjonsmodell og denne seksjonen oppsummerer problemene, og argumenterer for hvordan den nye distribusjonsmodellen løser en del av disse.

- Dumpen med folkeregisterdata inneholder kun et uttrekk, og dette fører til opprettelser av skyggeregister, samt at ID-forvaltningen hos NAV har tilgang til færre datafelt enn ID-forvaltningen hos SKD.
- Saksbehandlere i blant annet NAV får ikke jobbet direkte mot den sentrale databasen med folkeregisterdata.

Demonstratoren gjør det mulig for flere å jobbe direkte mot dataene som ligger i den sentrale databasen til DSF, uten å være avhengig av eksterne distributører eller å måtte vente på overføring av en dump med data. Selv om demonstratoren som er utviklet kun baserer seg på et uttrekk av DSF, er det ingenting i veien for at større deler av DSF kan distribueres på samme måte. Dette er imidlertid ikke bare et teknisk spørsmål, men også et spørsmål om hvilke deler av DSF som kan distribueres til hvilke aktører. For eksempel har de ulike aktørene som mottar folkeregisterdata ulike behov, og det er sikkert at det ikke er nødvendig for alle å ha tilgang til alt som ligger i DSF. Poenget er at den nye distribusjonsløsningen gjør det mulig å få tilgang til flere dataelementer, fordi man ikke er låst til kun de dataelementene som blir distribuert som en del av dumpen. Ved å muliggjøre tilgang til flere av dataelementene som ligger i DSF vil behovet for skyggeregister fjernes, noe som vil bidra til økt datakvalitet fordi det ikke er nødvendig å holde flere registre oppdatert og synkronisert.

Demonstratoren som er utviklet er ikke ment å erstatte dagens Folkeregister, men fungere ved å ha en prosess som automatisk konverterer og distribuerer RDF-data fra den sentrale DSF-databasen. Dermed vil sluttbrukere kunne få tilgang til oppdaterte data, samtidig som den sentrale databasen til DSF ikke trenger å eksponeres direkte mot sluttbruker. Et viktig poeng er at ett SPARQL-endepunkt som demonstratoren er satt opp med i dag ikke vil være tilstrekkelig. Det er ca. 1300 virksomheter i statlig, kommunal og privat sektor som mottar data fra Folkeregisteret [2, side 23], og en slik sentralisert løsning vil ikke være i nærheten av å kunne skalere godt nok til å tjene alle disse brukerne. Men det er ingenting i veien for at flere endepunkter og konverteringsprosesser kan settes opp parallelt, og at alle disse endepunktene distribuerer data fra samme sentrale database. Videre kan en tenke seg en mekanisme som tar i mot og videresender forespørsler til de forskjellige endepunktene der det er ledig kapasitet. En slik distribuert løsning vil fortsatt gjøre det mulig å jobbe mot oppdaterte data (fordi alle endepunktene distribuerer data fra samme kilde) samt gjøre det mulig å distribuere alle datafeltene som eksisterer i DSF.

For identifiseringsproblemet vil den nye distribusjonsmodellen føre til at ID-forvalterne i NAV får et bedre datagrunnlag for identifiseringsprosessen, som igjen vil føre til en sikrere identifiseringsprosess. Et problem som fortsatt vil gjelde er at ID-forvalterne hos SKD har tilgang til flere registre enn det ID-forvalterne hos NAV har. En løsning på dette kan være å distribuere de andre registrene som SKD har tilgang til på samme måte som demonstratoren distribuerer folkeregisterdata. Dette vil føre til at NAV får et enda større datagrunnlag å utføre identifisering på, og siden alle registrene distribueres som RDF, vil det være enkelt å kombinere dataene og skrive

spørringer som går på tvers av disse datasettene.

- Det er store utfordringer knyttet til semantisk interoperabilitet og dataintegrasjon. Dette kommer konkret til uttrykk i identifiseringsproblemet hvor to heterogene kilder, DSF og SED, skal kombineres.

- Det er et behov for å kunne beskrive metadata på en måte som gjør det mulig å gjenbruke denne informasjonen på tvers av systemer og etater.

En av grunnene for utfordringene knyttet til semantisk interoperabilitet og dataintegrasjon skyldes de ulike datamodellene som ligger til grunn for DSF og SED. For NAV sitt vedkommende ligger det en relasjonell database med sin tabulære struktur til grunn for DSF-dataene, mens SED-dataene distribueres som XML og har dermed en trestruktur som datamodell. Ved å distribuere begge kilder som RDF har vi vist hvordan disse to heterogene kildene enklere kan kombineres ved at begge kildene nå har den samme underliggende datamodellen, identifikatorer med et globalt omfang, samt antagelsen om NUNA.

I tillegg til ha en felles datamodell trengs det en utvetydig definisjon av begrepene i de kildene som skal integreres, og hvordan disse begrepene er relatert. Videre er det nødvendig, for å løse utfordringen med semantisk interoperabilitet, å ha en formell beskrivelse av dataene som sikrer en enighet mellom tilbyderen og mottageren av dataene. Ved eksplisitt å definere begreper og relasjoner mellom disse med OWL, samt gi en formell tekstlig beskrivelse av disse begrepene og relasjonene, vil ontologiene som er utviklet være med på å bidra til å fylle behovet for metadata som eksisterer i dag. De vil også bidra til å løse utfordringene knyttet til dataintegrasjon og semantisk interoperabilitet, da ontologiene vil bidra til å skape en felles forståelse for begreper fra forskjellige domener.

- Identifiseringsprosessen er en manuell prosess, det vil si at saksbehandlere manuelt må søke etter personene i TPS basert på data som kommer i SED-filene. Dette tar mye tid, og det kan også være større sannsynlighet for feil, enn hvis det hadde vært en automatisert prosess.

Vi har vist hvordan resonnering kan brukes til automatisk å fastslå at to personer er like. Det å automatisk fastslå at to personer fra forskjellige kilder er like og at dette legges inn i datasettet, kan bidra til en enklere identifiseringsprosess. OWL har noen begrensninger med tanke på hva slags nøkler som er mulig å lage, og det er blant annet ingen mulighet for å ha rollekjeder bestående av dataroller som nøkler. Dette, og måten DSF-dataene er strukturert, fører til at det ikke er mulig å lage en nøkkel som sier at to personer er like hvis for eksempel to personer har samme foreldre. Men vi har argumentert for måten dataene er strukturert på ved at det utnytter RDF-modellen bedre, samt at det fører til at det ikke lagres duplikatinformasjon.

SPARQL kan derimot brukes til å stille en spørring på tvers av DSF og SED som sjekker om flere personer fra DSF og SED har samme foreldre. Videre kan SPARQL brukes til å legge til en trippel som eksplisitt sier at disse personene er samme person. Som nevnt flere ganger er demonstratoren først og fremst tenkt til å brukes til datadistribusjon og ikke tenkt som et sted å legge inn data. Det er allikevel viktig å nevne muligheten SPARQL har til å legge inn data, og dermed vise potensialet en slik demonstrator har til også å brukes til å oppdatere og legge inn data. I tillegg til å bruke SPARQL til å identifisere personer og eksplisitt si at to personer er like, kan SPARQL brukes til å stille kontrollspørringer mot DSF-dataene for å kvalitetssikre identifiseringsprosessen. Selv om resonneringen bidrar til å kunne automatisere identifiseringsprosessen, er det viktig å kunne stille kontrollspørringer for å kvalitetssikre at identifiseringen har blitt utført korrekt.

6.5 Oppsummering

Ved å distribuere dataene som RDF får vi globale, utvetydige identifikatorer både for entiteter og relasjonene mellom disse. Dette, sammen med antagelsen om NUNA og RDF sin datamodell, bidrar til en enklere integrering av data fra heterogene kilder enn det som er mulig med tradisjonelle teknologier som relasjonelle databaser. Ved hjelp av SPARQL kan kraftige vilkårlige spørringer stilles på tvers av DSF- og SED-dataene som om man stiller spørringer mot én kilde. I kombinasjon med eksterne funksjonsbibliotek har man et kraftig spørringsverktøy tilgjengelig. Det RESTfulle grensesnittet som tilbys skjuler detaljene ved SPARQL og gjør det enkelt for saksbehandlere å slå opp på enkeltressurser eller ressurser av en gitt type. Ontologiene som er utviklet tilfører bakgrunnsinformasjon til dataene, og vil også fungere som dokumentasjon. Resonnering utvider datasettet med informasjon som ikke er eksplisitt lagret, og kan også anvendes til å fastslå at to personer er like basert på personens personidentifikator.

Demonstratoren gjør det mulig å jobbe mot oppdaterte data ved å ha en automatisk konverteringsprosess som eksponerer data fra den sentrale DSF-databasen. I tillegg vil det være mulig å utvide demonstratoren slik at sluttbrukere ikke er knyttet til de dataelementene som distribueres med dumpen med folkeregisterdataene, men kan få tilgang til flere dataelementer fra DSF. Dette vil føre til at ID-forvalterne i NAV får et større og bedre datagrunnlag for identifiseringen. Ved å ha integrert SED- og DSF-ontologiene vil dette bidra til enklere dataintegrasjon, og ontologiene vil også kunne brukes for å dekke behovet for metadata.

Kapittel 7

Utfordringer og begrensninger

Kapittel 6 har beskrevet hvilke resultater og gevinster den nye demonstrator vil gi. Dette kapittelet beskriver begrensninger med demonstratoren som er utviklet, samt generelle utfordringer knyttet til semantiske teknologier som er erfart gjennom arbeidet med demonstratoren. Dette gjelder både begrensninger og utfordringer knyttet til teknologien og verktøy, men også knyttet til prinsippene som ligger til grunn for semantiske teknologier.

7.1 Ny og umoden teknologi

Det er ofte utfordringer knyttet til nye teknologier, både med tanke på tekniske begrensninger, men også knyttet til kunnskap om den nye teknologien. Dette er også tilfellet for semantiske teknologier. Som vi har vist kan semantiske teknologier være med på å løse utfordringer knyttet til dataintegrasjon og semantisk interoperabilitet, men på tross av dette har ikke teknologiene blitt vanlig å bruke i ordinære systemer. For det første er det en viss mangel når det kommer til modenhet av verktøy, som kan føre til at flere ikke ønsker å bruke teknologiene i kritiske systemer hvor de er avhengig av gode og pålitelig implementasjoner. Denne erfaringen støttes også av Dau [24] som i artikkelen *Semantic technologies for enterprises* peker på at mange verktøy knyttet til semantiske teknologier er resultater av forskningsprosjekter, og har derfor en mangel på dokumentasjon og vedlikehold. På den andre siden kommer det bare flere og flere ulike implementasjoner av for eksempel ulike triple store og SPARQL-endepunkt på markedet. I tillegg kommer aktører som Oracle og IBM på banen med produkter rettet mot industrien.

Et annet problem kan være at teknologiene er vanskelig å selge inn på grunn av mangel på gode demonstratorer som viser mulighetene, potensialet og særegenhetene til teknologiene. Hvis det er mulig å peke på vellykkede prosjekter som anvender semantisk teknologi, vil det bli enklere for andre å bruke teknologiene i sine prosjekter.

Disse punktene samsvarer med min erfaring knyttet til å lage demonstratoren som er presentert i oppgaven. Min erfaring er at det eksisterer en rekke forskjellige type verktøy, men at det er en begrensing når det kommer til modenheten av verktøyene og implementasjonene. Et problem har derfor vært å finne god dokumentasjon på hvilke verktøy som bør anvendes til hvilke formål. Her har både veileder og andre ressurspersoner på Ifi vært til stor hjelp, men jeg ser det er vanskelig å skaffe seg oversikt over dette feltet hvis man ikke har tilgang til personer som har erfaring på dette området.

Som vi har vist er resonnering et viktig begrep innenfor semantiske teknologier, og gir en fordel sammenlignet med tradisjonelle teknologier, hvor all kunnskap eksplisitt må lagres i databasen for at det skal kunne presenteres for brukeren. En utfordring med resonnering er ytelsen, og selv med relativt enkle modeller og små datamengder kan resonnering ta veldig lang tid. Dette kan føre til at det går utover den ønskede responstiden fra systemet. Det forskes mye på hvor komplekse modeller kan være og samtidig kunne utføre resonnering innenfor gitte kompleksitetsklasser. Men det krever mye erfaring og kunnskap om hvordan resonnerere fungerer for å vite grunnen til at resonnering i et bestemt prosjekt med en bestemt modell går sakte. I tillegg vil hastigheten variere avhengig av hvilken resonnerer som er valgt. Men det forskes også mye på dette området, og dette sammen med at maskinvare blir bedre og bedre, vil føre til at resonnering bare vil gå raskere og raskere.

En annen grunn til at semantiske teknologier ikke er så mye brukt som mange har spådd, kan være at det mangler en klar formulering av mål og en forståelse for forretningsverdien til semantiske teknologier [47]. Det har vært mye «hype» rundt semantiske teknologier og den semantiske weben, og dette kan ha ført til at flere har trodd at semantiske teknologier skal kunne løse «alle» problemer knyttet til dataintegrasjon og distribusjon. Teknologiene er derimot ikke en løsning på alle mulig problemer, men kan bidra til økte forretningsverdier på noen punkter. Utfordringen er å klare å se nøyaktig hvor, og med hvor mye verdien kan økes.

Dau [24] sier også at det er vanskelig å kvantitativt måle henholdsvis fordeler og ulemper med semantisk teknologi, noe som fører til at det er vanskelig å vise til kvantitative fordeler ved semantiske teknologier sammenlignet med for eksempel tradisjonell databaseteknologi. Et eksempel som trekkes frem er hvordan relasjonelle databaser lenge har hatt en standardisert metode for å måle ytelsen[1], mens det for semantiske teknologier ikke har eksistert en liknende standardisert måte å måle ytelse på. Det har blitt utviklet flere metoder for dette[2], men det eksisterer ikke én standard som det gjør for relasjonelle databaser. En av grunnene til dette kan være at semantiske teknologier er mer komplekse enn relasjonelle databaser, og at mange faktorer som har innvirkning på ytelse og skalerbarhet må taes med.

[1] Transaction Processing Performance Council — `http://www.tpc.org/`

[2] For en oversikt se her: `http://www.w3.org/wiki/RdfStoreBenchmarking`

Et viktig poeng er at punktene knyttet til utfordringer med ny teknologi er problemer av mindre karakter. Grunnen til dette er at det stadig utvikles flere og mer modne verktøy som kan anvendes i applikasjoner som bruker semantiske teknologier. Det å sammenligne ytelse og skalerbarhet med relasjonelle databaser vil heller ikke være helt rettferdig, da semantisk teknologi er en forholdsvis ny teknologi sammenlignet med relasjonelle databaser. De store utfordringene vil derfor kanskje heller være knyttet til prinsippene som ligger til grunn for semantiske teknologier som en *åpen-verden semantikk*, en antagelse om *ikke-unike navn* og det å beskrive semantikk.

7.2 Åpen-verden semantikk og antagelsen om ikke-unike navn

En åpen-verden semantikk og en antagelse om ikke-unike navn, er to viktig prinsipper innenfor semantiske teknologier, og det er kanskje på disse punktene semantiske teknologier skiller seg mest ut fra tradisjonelle teknologier. Siden tradisjonelle teknologier som relasjonelle databaser antar en lukket verden og har en antagelse om unike navn, vil det være utfordrende for utviklere og brukere som er vant med disse teknologiene å lære, og ikke minst utnytte, prinsippene som ligger til grunn for semantiske teknologier.

Antagelsen om ikke-unike navn er godt egnet for dataintegrasjon, fordi det der ikke er ønskelig å anta at to entiteter med forskjellig identifikator verken er like eller ulike før dette er eksplisitt definert. For identifiseringsproblemet passer denne antagelsen dermed godt. Når det gjelder antagelsen om en åpen-verden semantikk passer denne antagelsen også godt for dataintegrasjon fordi det der er ønskelig å kunne dra slutninger basert på ufullstendig informasjon. På den andre siden kan denne antagelsen også by på utfordringer. Hvis folkeregisterdataene sees på isolert sett, kan det diskuteres om en åpen-verden semantikk er riktig antagelse. Folkeregisteret bør sees på som et komplett datasett, det vil si at hvis en person ikke eksisterer i Folkeregisteret, så har ikke personen noen tilknytning til Norge. På den andre siden hvis Folkeregisteret sees på som et datasett som ikke er komplett, fordi andre datasett fra andre etater kan supplere folkeregisterdataene, vil en antagelse om en åpen verden være korrekt.

Det kan likevel være ønskelig å kunne bruke semantiske teknologier og samtidig ha en lukket-verden semantikk, for å kunne si at et utsagn er usant når noe ikke er sant i henhold til databasen. Det eksisterer noen muligheter for å «tvinge» inn en lukket verden ved for eksempel å la klassen `owl:Thing` (som er superklassen til alle andre klasser) være lik en enumerasjon av alle individer i datasettet. Dette gjør at antall kjente individer begrenses til kun de som eksisterer i modellen. Ulempen med dette er at det antageligvis vil være veldig kostbart, fordi en resonnerer er nødt til å gå gjennom alle individene i modellen og sjekke at de faktisk er medlem av denne

anonyme klassen. En ting er å sjekke alle individer som er eksplisitt definert i modellen, men det kan også være individer som etter resonnering også skal være en del av denne klassen.

7.3 Utfordringer knyttet til beskrivelse av semantikk

En stor utfordring gjennom arbeidet med oppgaven, har vært å lage gode ontologier for både DSF og SED. På grunn av begrenset erfaring med ontologiutvikling og modellering generelt, har det vært en stor utfordring og finne ut hvordan jeg skulle klare å modellere på en god måte. En svakhet med dagens modell er at den inneholder få utsagn knyttet til definisjoner av ulike begreper. En ontologi bør kunne svare på spørsmål som hva det er som kjennetegner en person. Det vil si hvilke attributter må en person ha, hvilke er valgfrie og hvilke relasjoner har en person til andre begreper. Ved å ha tilgang på bedre dokumentasjon og domeneeksperter tror jeg at det ville vært en større sannsynlighet for at en bedre modell, hvor begreper er klarere definert, hadde blitt utviklet. Samtidig er det ingen garanti for at tilgang til mer dokumentasjon hadde ført til en bedre modell. Det er heller ikke noen fasit når det gjelder modellering, og det kan være flere forskjellige modeller som kan være like riktige. Modellen som er utviklet er etter min mening ikke feil, og det kan godt være at den kan videreutvikles uten at det er nødvendig å starte på nytt.

Generelt er det å beskrive semantikk utfordrende, og det eksisterer ingen automatiske måter å generere ontologier for et gitt domene på. Dette fører til at ontologiutvkling er tidkrevende og kostbart, da det vil kreve opplæring eller bruk av eksterne ressurser til å lage ontologier. Det krever også ressurser fra de personene som virkelig kan domene som skal modelleres, såkalte domeneeksperter. Men for virkelig å kunne utnytte semantiske teknologier og de mulighetene de gir, er det viktig at det brukes tid på å lage gode ontologier og at det brukes tid på å få disse gode og korrekte. Ved å ha gode modeller vil det være mulig å enklere kunne integrere datakilder, og anvende resonnering. Som vi har vist vil også denne bakgrunnsinformasjonen kunne brukes som dokumentasjon. Et annet poeng er at det ikke eksisterer en standardisert metode for ontologiutvkling [24]. Vi har presentert én metode i kapittel 4, og det eksisterer en rekke andre, men ingen av disse har blitt en *de-facto* standard for ontologiutvkling.

Et annet poeng er at det ikke er alt som er enkelt eller mulig å modellere med OWL. En måte å videreutvikle dagens modell på kan være å ta utgangspunkt i lovverket og prøve å modellere dette. For eksempel vil *Lov om folkeregistrering* være en naturlig lov å ta utgangspunkt i. Problemet med å prøve å modellere lovtekster i OWL, er at OWL er dårlig til å beskrive ord som *kan* og *bør*. Dette er ofte ord man finner igjen i lovverk og et eksempel fra folkeregisterloven er paragraf 4 som sier: «*[...] For andre personer kan det fastsettes enten et fødselsnummer eller et D-nummer når det foreligger et*

begrunnet behov for det. [...]». Her ser vi bruken av *kan* som ikke er enkelt å oversette til utsagn i OWL.

Likevel er det viktig å presisere at dette ikke bør stoppe en fra å ta i bruk semantiske teknologier. En grunn til at disse teknologiene ikke blir tatt i bruk, kan være at mange tenker at modellene som skal utvikles må være helt ferdige og perfekte før man kan begynne å se på selve dataene som skal distribueres. Som nevnt i seksjon 3.3.4 på side 43, kan det være hensiktsmessig å ha en «nedenfra-og-opp»-tilnærming til problemet istedenfor en «ovenfra-og-ned»-tilnærming når semantiske teknologier skal anvendes. Denne oppgaven har hatt en slik «nedenfra-og-opp»-tilnærming, da vi først har konsentrert oss om å konvertere dataene, og deretter har sett på utviklingen av ontologiene. Dette gjør at man kan få distribuert dataene som RDF og få utnyttet de mulighetene det i seg selv gir, og videre utvikle ontologiene i iterasjoner og dermed legge til stadig mer semantikk. Et viktig uttrykk er at «a little semantics goes a long way», som første gang ble nevnt av James A. Hendler [40]. Dette betyr at selv med litt semantikk, i denne sammenheng en liten ontologi, vil man kunne få utrettet mye, og mye mer enn uten noe semantikk overhodet.

7.4 Oppsummering

Semantiske teknologier er, sammenlignet med mer tradisjonelle teknologier som relasjonelle databaser, en forholdsvis ny teknologi, og som med andre nye teknologier er det utfordringer knyttet til semantiske teknologier. Dette kommer til uttrykk i en mangel på modne verktøy som kan brukes i systemer som er avhengig av god skalerbarhet og stabilitet. I tillegg kan det være vanskelig å se nøyaktig hvilken forretningsverdi semantiske teknologier har sammenlignet med eksisterende systemer, og dette kan føre til at man ikke ønsker å ta i bruk disse teknologiene i systemer hvor man har gode stabile løsninger på plass.

Videre er det knyttet utfordringer til prinsippene om en åpen verden og ikke-unike navn. Prinsippene skiller seg ganske radikalt fra mer tradisjonelle teknologier, og det vil være utfordrende for brukere å utnytte disse prinsippene, og å se fordelene ved dem.

Semantiske teknologier gjør det mulig å lage ontologier som eksplisitt definerer begreper og relasjoner mellom disse, men dette er utfordrende, tidkrevende og kostbart arbeid. Det krever mye opplæring eller innhenting av eksterne ressurser for å kunne utvikle ontologier, og det eksisterer ingen metoder for automatisk å generere ontologier ut i fra forskjellige typer dokumentasjon.

Kapittel 8

Konklusjon

8.1 Oppsummering av arbeidet

Problemstillingen for oppgaven er:

> *Hvordan er semantiske teknologier egnet til å utvikle en ny, forbedret distribusjonsmodell for Folkeregisteret, og hvordan vil denne modellen egne seg til å integrere andre heterogene datakilder?*

Oppgaven har sett på hvordan semantiske teknologier egner seg til å løse utfordringer knyttet til distribusjon og integrasjon av folkeregisterdata. I tillegg til å se på generelle utfordringer har oppgaven også presentert et konkret case, identifiseringsproblemet, som viser hvordan utfordringer knyttet til distribusjon og integrasjon gjør seg gjeldende hos NAV. Etter å ha beskrevet utfordringene presenterte oppgaven en overordnet beskrivelse av en ny distribusjonsløsning som fokuserte på tre hovedpunkter: distribusjon, spørringer og modell for dataintegrasjon.

Semantiske teknologier er av flere nevnt som teknologier som kan bidra til å løse flere av utfordringene knyttet til blant annet dataintegrasjon og semantisk interoperabilitet. For å vise og vurdere hvordan disse teknologiene egner seg til dette konkrete caset, har vi implementert en distribusjonsløsning som anvender forskjellige deler av semantisk teknologi, og deretter vist resultater og gevinster som løsningen gir samt diskutert utfordringer og begrensinger.

Oppgaven har vist at semantiske teknologier på flere områder egner seg til å utvikle en ny forbedret distribusjonsmodell. Fordeler som globale identifikatorer, antagelsen om ikke-unike navn, eksplisitte identitetsutsagn og samme underliggende datamodell gjør det enklere å integrere heterogene datakilder enn det som er mulig med tradisjonelle teknologier. I tillegg vil egenskaper som at RDF er skjemaløst føre til at det er enkelt å legge til og fjerne data, samt at det er enklere og mindre kostbart å endre på ontologien enn et tradisjonelt databaseskjema. I tillegg er RDF en datamodell, noe som betyr at dataene kan serialiseres til en rekke formater, som gjør det mulig

for en rekke forskjellige applikasjoner å konsumere dataene.

Gjennom et SPARQL-endepunkt kan kraftige, vilkårlige spørringer stilles mot dataene. Samtidig er det mulig å skjule detaljene og kompleksiteten til RDF, OWL og SPARQL ved å tilby et RESTfullt grensesnitt som samtidig kan være et kraftig grensesnitt for sluttbrukere og applikasjoner. Disse to måtene å konsumere data på gir til sammen en fleksibel og kraftig distribusjonsløsning. Til kommunikasjon og dataoverføring anvendes HTTP-protokollen som gir et grensesnitt som ikke stiller spesielle krav til rammeverk eller programmeringsspråk.

Til slutt vil ontologiene som er utviklet bidra med bakgrunnsinformasjon ved eksplisitt å definere begreper og relasjoner mellom disse, samtidig som modellene vil kunne brukes til å lagre og forvalte metadata, og fungere som dokumentasjon. Med ontologien kan resonnering utføres både generelt ved å sjekke om ontologien er konsistent og utvide datasettet, samt spesielt ved å fastslå om to personer fra to heterogene datakilder er samme person.

8.2 Videre arbeid

Det vil være flere punkter som kan være interessante å jobbe videre med i forbindelse med dette caset. Denne seksjonen presenterer noen av de punktene jeg anser som de viktigste.

Utvide ontologiene

Ontologiene som er utviklet er ganske magre og langt fra komplette, og det er store muligheter for å utvide disse. For at dette skal kunne gjøres er det nødvendig med mer dokumentasjon enn hva vi har fått tilgang til, og helst tilgang til domeneeksperter. I tillegg kunne det vært interessant å se mer på integrering av datakildene. En utfordring som ofte er med dataintegrasjon er at samme type data kan lagres på forskjellig format. For eksempel kan fødselsdatoen i ett system lagres på et annet format enn i et annet system, selv om datoen er den samme. For at to datoer på forskjellig format skal kunne sammenlignes er det derfor nødvendig med en omskriving slik at datoene først sammenlignes når de er på samme format. Selv med dataene på RDF og en OWL-modell som relaterer ulike begreper, vil ikke en slik utfordring kunne løses enkelt med OWL. Som vi har sett kan OWL brukes til å eksplisitt definere sammenhenger mellom to domener, men kan ikke brukes direkte til en omskriving, som er nødvendig i dette tilfellet.

Analysere og vurdere ulike verktøy og implementasjoner

Demonstratoren inneholder lite data, og det er derfor ikke nødvendig å stille store krav til valg av verktøy og implementasjoner. Hvis demonstratoren skal utvides, ville det vært interessant å analysere og vurdere hvilke

komponenter som kan egne seg i en eventuell produksjonssatt utgave av løsningen.

Sikkerhet og policy

Dataene som ligger i Folkeregisteret har strenge krav knyttet til hvem som skal få tilgang til de forskjellige dataene. Denne oppgaven har ikke sett noe på sikkerhetsaspektet, rett og slett fordi det ikke har vært mulig å få plass til dette i denne oppgaven. Et viktig poeng er at siden HTTP-protokollen brukes til kommunikasjon og dataoverføring vil alle mekanismer som er tilgjengelig for denne protokollen, slik som HTTPS, være tilgjengelig for vår demonstrator. Oppgaven har videre nevnt at navngitte grafer kan bidra til å sette adgangskontroll på deler av dataene, men har ikke gått i detalj rundt hvordan dette kan løses ved hjelp av for eksempel eksisterende policy-språk.

Konvertering av SED til RDF

Vi har ikke brukt tid på å konvertere SED-dataene til RDF på samme måte som vi har brukt tid på å konvertere DSF-dataene til RDF. Det kunne vært interessant å lage et lignende program for å konvertere SED-data til RDF. Dette hadde da vært nødvendig med en grundigere analyse av de forskjellige SED-flytene samt se på de forskjellige SED-ene som eksisterer. I tillegg hadde det vært nødvendig å få tilgang til eksempeldata.

Koble demonstratoren opp mot eksisterende systemer

Vi har vist og dokumentert gevinstene og mulighetene demonstratoren gir, og et naturlig neste steg kan være å la eksisterende systemer og applikasjoner forsøke å hente ut data fra demonstratoren. Det ville vært interessant å se hvordan demonstratoren ville fungert til dette formålet, og om og isåfall hva som måtte endres for å få det til å fungere.

Bibliografi

[1] Arbeids- og administrasjonsdepartementet. Fra bruk til gjenbruk. Rapport, august 2008. 1, 2, 10

[2] Arbeidsgruppe for utveksling av grunndata på personinformasjonsområdet. Utveksling av grunndata på personinformasjonsområde. Rapport, juni 2007. 2, 6, 10, 11, 13, 105

[3] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi og Peter F. Patel-Schneider, redaktører. *The Description Logic Handbook: Theory Implementation and Applications*. Cambridge, august 2007. 37

[4] Jie Bao, Elisa F. Kendall, Deborah L. McGuinnes og Peter F. Patel-Schneider. OWL 2 Web Ontology Language — Quick Reference Guide. `http://www.w3.org/TR/owl2-quick-reference/`, oktober 2009. Besøkt 3. april 2012. 37

[5] Robert Battle og Edward Benson. Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST). *Web Semant.*, 6:61–69, februar 2008. ISSN 1570-8268. 42, 45

[6] Dave Beckett. RDF/XML Syntax Specification. `http://www.w3.org/TR/rdf-syntax-grammar`, 2004. Besøkt 2. januar 2012. 24

[7] David Beckett og Tim Berners-Lee. Turtle — Terse RDF Triple Language. `http://www.w3.org/TeamSubmission/turtle`, 2011. Besøkt 2. januar 2012. 25

[8] Michael K. Bergman. WOA: A New Enterprise Partner for Linked Data. `http://www.mkbergman.com/459/woa-a-new-enterprise-partner-for-linked-data/`, oktober 2008. Besøkt 1. mai 2012. 45

[9] Michael K. Bergman. Advantages and Myths of RDF. `http://www.mkbergman.com/wp-content/themes/ai3/files/2009Posts/Advantages_Myths_RDF_090422.pdf`, 2009. 21

[10] Michael K. Bergman. The Fundamental Importance of Keeping an ABox and TBox Split. `http://www.mkbergman.com/489/ontology-best-practices-for-data-driven-applications-part-2/`, mai 2009. Besøkt 28. april 2012. 37

[11] T. Berners-Lee, R. Fielding og L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. http://tools.ietf.org/html/rfc3986, januar 2005. Besøkt 8. mars 2012. 21

[12] Tim Berners-Lee. Notation 3 Logic. http://www.w3.org/Designissues/Notation3.html, 2005. Besøkt 2. januar 2012. 25

[13] Tim Berners-Lee. Linked Data. http://www.w3.org/Designissues/LinkedData.html, 2006. Besøkt 18. januar 2012. 44

[14] Tim Berners-Lee. Linked Open Data. http://www.w3.org/2008/Talks/0617-lod-tbl/, 2008. Besøkt 3. april 2012. 43

[15] Tim Berners-Lee. Putting Government Data online. http://www.w3.org/DesignIssues/GovData.html, juni 2009. Besøkt 12. april 2012. 44

[16] Tim Berners-Lee og Mark Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*. HarperBusiness, november 2000. 17

[17] Diego Berrueta og Jon Phipps. Best Practice Recipes for Publishing RDF Vocabularies. http://www.w3.org/TR/swbp-vocab-pub/, august 2008. Besøkt 28. april 2012. 50

[18] Chris Bizer og Richard Cyganiak. The TriG Syntax. http://www4.wiwiss.fu-berlin.de/bizer/trig/, 2007. Besøkt 11. januar 2012. 25, 87

[19] Christian Bizer, Tom Heath og Tim Berners-Lee. Linked Data — The Story So Far. *International journal on Semantic Web and Information Systems*, 5(3):1–22, 2009. 43

[20] Christian Bizer og Andreas Schultz. The R2R Framework: Publishing and Discovering Mappings on the Web. I *First International Workshop on Consuming Linked Data (COLD2010)*. 53

[21] Jeremy Carroll, Ivan Herman og Peter F. Patel-Schneider. OWL 2 Web Ontology Language — RDF-Based Semantics. http://www.w3.org/TR/owl2-rdf-based-semantics/, oktober 2009. Besøkt 3. april 2012. 38

[22] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman og Sanjiva Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. http://www.w3.org/TR/wsdl20/, 2007. Besøkt 3. april 2012. 41

[23] Clark & Parsia, LLC. Pellet: OWL 2 Reasoner for Java. http://clarkparsia.com/pellet/, 2011. Besøkt 14. april 2012. 83

[24] Frithjof Dau. Semantic technologies for enterprises. I *Proceedings of the 19th international conference on Conceptual structures for discovering knowledge*, ICCS'11, side 1–18, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-22687-8. 92, 93, 109, 110, 112

[25] M. Elkstein. Learn REST: A Tutorial. `http://rest.elkstein.org/2008/02/rest-as-lightweight-web-services.html`, juni 2011. Besøkt 28. april 2012. 42

[26] Epimorphics Ltd. Linked data API. `http://www.epimorphics.com/web/projects/linked-data-api`, 2012. Besøkt 18. april 2012. 83

[27] Jérôme Euzenat og Pavel Shvaiko. *Ontology matching*. Springer-Verlag, Heidelberg (DE), 2007. ISBN 3-540-49611-4. 75

[28] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. Doktorgradsoppgave, University of California, 2000. 42

[29] Foaf project. The Friend of a Friend (FOAF) project. `http://www.foaf-project.org/`. Besøkt 16. april 2012. 34

[30] The Apache Software Foundation. Apache Jena. `http://incubator.apache.org/jena/`, 2011. Besøkt 2. april 2012. 80

[31] Paul Gearon, Alexandre Passant og Axel Polleres. SPARQL 1.1 Update. `http://www.w3.org/TR/sparql11-update`, 2012. Besøkt 10. januar 2012. 33

[32] Jan Grant og Dave Beckett. RDF Test Cases — N-Triples. `http://www.w3.org/TR/rdf-testcases/#ntriples`, 2004. Besøkt 2. januar 2012. 25

[33] Kendall Grant Clark, Lee Feigenbaum og Elias Torres. SPARQL Protocol for RDF. `http://www.w3.org/TR/rdf-sparql-protocol/`, januar 2008. Besøkt 13. mars 2012. 41

[34] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar og Yves Lafon. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). `http://www.w3.org/TR/soap12-part1/`, april 2007. Besøkt 1. mars 2012. 42

[35] Steve Harris og Andy Seaborne. SPARQL 1.1 Query Language. `http://www.w3.org/TR/sparql11-query`, 2012. Besøkt 10. januar 2012. 31, 32

[36] Patrick Hayes. RDF Semantics. `http://www.w3.org/TR/rdf-mt/`, februar 2004. Besøkt 14. mars 2012. 35

[37] Tom Heath og Christian Bizer. *Linked Data — Evolving the Web into a Global Data Space*. Morgan & Claypool, 2011. 44, 50

[38] John Hebeler, Matthew Fisher, Ryan Blace og Andrew Perez-Lopez. *Semantic Web Programming*. Wiley Publishing, Inc., 2009. 41, 79, 80

[39] Sandra Heiler. Semantic interoperability. *ACM Computing Surveys*, 27(2):271–273, juni 1995. ISSN 0360-0300. 11

[40] James A. Hendler. A Little Semantics Goes a Long Way. http://www.cs.rpi.edu/~hendler/LittleSemanticsWeb.html. Besøkt 21. april 2012. 113

[41] Pascal Hitzler, Markus Krötzsch og Rudolph Sebastian. *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC, 2010. 20, 22, 26, 29, 30, 34, 35, 36, 37, 61, 62, 69

[42] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider og Sebastian Rudolph. OWL 2 Web Ontology Language: Primer. http://www.w3.org/TR/owl-primer/, oktober 2009. Besøkt 13. mars 2012. 37

[43] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider og Sebastian Rudolph. OWL 2 Web Ontology Language — Primer. http://www.w3.org/TR/owl2-primer/, oktober 2009. Besøkt 3. april 2012. 37, 38

[44] Hans Christian Holte. Dyr og dårlig datahåndtering. http://www.idg.no/computerworld/article157722.ece, februar 2012. Besøkt 3. april 2012. 2

[45] Matthew Horridge, Simon Jupp, Georgina Moulton, Alan Rector, Robert Stevens og Chris Wroe. A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools Edition 1.1. http://tools.ietf.org/html/rfc4180, oktober 2007. 67

[46] Joseki. Joseki — A SPARQL Server for Jena. http://www.joseki.org. Besøkt 21. april 2012. 80

[47] John Kuriakose. Building real world solutions with Semantic Technology — putting the pieces together. http://www.infosysblogs.com/infosys-labs/2011/01/building_real_world_solutions.html, januar 2011. Besøkt 30. april 2012. 110

[48] Andreas Langegger og Wolfram Wöß. XLWrap — Querying and Integrating Arbitrary Spreadsheets with SPARQL. I *8th International Semantic Web Conference (ISWC2009)*, oktober 2009. 86

[49] Maurizio Lenzerini. Data integration: a theoretical perspective. I *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '02, side 233–246, New York, NY, USA, 2002. ACM. ISBN 1-58113-507-6. 14

[50] Frank Manola og Eric Miller. RDF Primer. http://www.w3.org/TR/rdf-primer/, februar 2004. Besøkt 19. desember 2011. 20, 21, 22

[51] Mort Bay Consulting. jetty — jetty WebServer. http://jetty.codehaus.org/jetty/, april 2011. Besøkt 21. april 2012. 80

[52] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue og Carsten Lutz. OWL 2 Web Ontology Language — Profiles. http://www.w3.org/TR/owl2-profiles/, oktober 2009. Besøkt 3. januar 2011. 40

[53] Boris Motik, Peter F. Patel-Schneider og Bernardo Cuenca Grau. OWL 2 Web Ontology Language — Direct Semantics. http://www.w3.org/TR/owl2-direct-semantics/, oktober 2009. Besøkt 3. april 2012. 38

[54] NAV. Vedlegg 2 til kapittel 2 — Hovednr 45 — Veiledning til A-SED . http://www.nav.no/rettskildene/Vedlegg/Vedlegg+2+til+kapittel+2+-+Hovednr+45+-+Veiledning+til+A-SED.301506.cms, mars 2011. Besøkt 30. mars 2012. 9

[55] NAV. NAV. http://www.nav.no/Om+NAV/NAV, 2012. Besøkt 7. mars 2012. 7

[56] Open CSV. opencsv. http://opencsv.sourceforge.net/, juli 2011. Besøkt 21. april 2012. 86

[57] Tim O'Reilly. What Is Web 2.0 — Design Patterns and Business Models for the Next Generation of Software. http://oreilly.com/web2/archive/what-is-web-20.html, september 2005. Besøkt 29. februar 2012. 45

[58] Kevin R. Page, David C. De Roure og Kirk Martinez. REST and Linked Data: a match made for domain driven development? I *2nd International Workshop on RESTful Design*, mars 2011. 45

[59] Eric Prud'hommeaux og Carlos Buil-Aranda. SPARQL 1.1 Federated Query. http://www.w3.org/TR/sparql11-federated-query/, november 2011. Besøkt 16. april 2012. 41

[60] Eric Prud'hommeaux og Andy Seaborne. SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/, 2012. Besøkt 7. april 2012. 26

[61] RDF Working Group. RDF. http://www.w3.org/RDF/, 2004. Besøkt 1. februar 2012. 20

[62] Dave Reynolds. Jena 2 Inference support. http://jena.sourceforge.net/inference/, mars 2010. Besøkt 14. april 2012. 83

[63] Jack Rusher. Triple Store. http://www.w3.org/2001/sw/Europe/events/20031113-storage/positions/rusher.html, 2001. Besøkt 3. april 2012. 40

[64] Semantic Web. SPARQL endpoint. http://semanticweb.org/wiki/SPARQL_endpoint, 2009. Besøkt 18. januar 2012. 41

[65] Semicolon. Ontologibasert dataintegrasjon og -utveksling. http://semicolon.wiki.ifi.uio.no/Ontologibasert_dataintegrasjon_og_-utveksling, november 2009. Besøkt 12. april 2012. 15

[66] Y. Shafranovich. Common Format and MIME Type for Comma-Separated Values (CSV) Files. http://tools.ietf.org/html/rfc4180, oktober 2005. Besøkt 22. mars 2012. 48

[67] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur og Yarden Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics:*

Science, Services and Agents on the World Wide Web, 5(2):51 – 53, 2007. Software Engineering and the Semantic Web. ISSN 1570-8268. 83

[68] Skatteetaten. Om Skatteetaten. `http://www.skatteetaten.no/no/Alt-om/Om-Skatteetaten/`. Besøkt 24. april 2012. 6

[69] Audun Stolpe. ?Hvor: Norske addresser og steder med maksimal granularitet. `http://vocab.lenka.no/hvor`, april 2012. Besøkt 17. april 2012. 69

[70] Audun Stolpe og Martin G. Skjæveland. Preserving Information Content in RDF using Bounded Homomorphisms. I Elena Simperl et al., redaktører, *ESWC 2012*, bind 7295 av *LNCS*, side 72–86, Heidelberg, 2012. Springer. 54

[71] Store norske leksikon. arbeids- og velferdsforvaltningen. `http://snl.no/.sml_artikkel/arbeids-_og_velferdsforvaltningen`. Besøkt 24. april 2012. 7

[72] Store norske leksikon. folkeregister. `http://snl.no/folkeregister`, februar 2009. Besøkt 24. april 2012. 6

[73] Store norske leksikon. NAV. `http://snl.no/NAV`, januar 2012. Besøkt 24. april 2012. 7

[74] R. Thurlow. RPC: Remote Procedure Call Protocol Specification Version 2. `http://tools.ietf.org/html/rfc5531`, mai 2009. Besøkt 1. mars 2012. 42

[75] W3C. Jena, a Java RDF API and toolkit. `http://www.w3.org/2001/sw/wiki/Jena`, januar 2011. Besøkt 14. april 2012. 81

[76] W3C OWL Working Group. OWL 2 Web Ontology Language — Document Overview. `http://www.w3.org/TR/owl2-overview`, oktober 2009. Besøkt 3. januar 2011. 37, 38

[77] Don Wells. Acceptance Tests. `http://www.extremeprogramming.org/rules/functionaltests.html`, 1999. Besøkt 28. april 2012. 65

[78] Wikipedia. Det sentrale folkeregister. `http://no.wikipedia.org/w/index.php?title=Det_sentrale_folkeregister&oldid=8834397`, 2011. Besøkt 24. april 2012. 5

[79] Erik Wilde og Michael Hausenblas. RESTful SPARQL? You name it!: aligning SPARQL with REST and resource orientation. I *Proceedings of the 4th Workshop on Emerging Web Services Technology*, WEWST '09, side 39–43, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-776-9. 42, 45

[80] Olav Anders Øvrebø. Fakta først. Viderebruk av datakilder i offentlig sektor: potensial og hindringer. Rapport, januar 2012. 2, 11

Vedlegg A

Nedlastinger

Prosjektside http://sws.ifi.uio.no/project/dsf/henriwi

RDF av Folkeregisteret http://sws.ifi.uio.no/project/dsf/henriwi/dsf-clean.ttl

Resonnert RDF av Folkeregisteret http://sws.ifi.uio.no/project/dsf/henriwi/dsf.ttl

RDF av SED http://sws.ifi.uio.no/project/dsf/henriwi/sed-clean.ttl

Resonnert RDF av SED http://sws.ifi.uio.no/project/dsf/henriwi/sed.ttl

Ontologier til DSF og SED http://sws.ifi.uio.no/vocab/dsf/henriwi/

DSFConverter http://sws.ifi.uio.no/project/dsf/henriwi/dsfconverter

DSFConverter (ZIP) http://sws.ifi.uio.no/project/dsf/henriwi/dsfconverter.zip

Vedlegg B

CONSTRUCT-spørringer

Dette vedlegget viser alle CONSTRUCT-spørringene som er benyttet i forbindelse med konverteringen av rådataene til RDF. I tillegg til CONSTRUCT-spørringene vises det hvordan dataene ser ut henholdsvis før og etter at spørringene har blitt utført. Dette vedlegget er ment som et supplement til seksjon 4.2.2 på side 53, og deler av teksten overlapper det som er skrevet der.

Hver seksjon starter med å gi en kort argumentasjon for hvilken tilbakebetingelse (p1, p2 eller p3) som er oppfylt. Videre blir det for hver seksjon vist fire oversikter. Den første viser den aktuelle delen av RDF-grafen etter den første konverteringen (den naive algoritmen). Den andre viser selve CONSTRUCT-spørringen, mens den tredje viser hvilken strukturbevarende funksjon (mapping) som er benyttet. Til slutt vises dataene etter at CONSTRUCT-spørringen er utført.

B.1 Adresse

Mappingen som er blitt anvendt oppfyller kravene til p1. Dette er fordi alle elementer som er i målgrafen, også er relatert i kildegrafen, og det eneste som gjøres er å omskrive predikatene. I tillegg har den blanke noden fått en type, og dette er også tillatt, siden `rdf:type` ikke er i verdiområde til mappingen.

Adressedelen av RDF-grafen etter den første konverteringen:

```
dsfp:01010752171
  dsfv:adressenavn "BOKS 6300, ETTERSTAD" ;
  dsfv:harAdressetype dsf:adressetype#kode-M ;
  dsfv:flyttedatoForAdresse "2007-01-01"^^xsd:date ;
  dsfv:gateGaard "00019" ;
  dsfv:husBruk "0019" ;
  dsfv:kommunenummer "0019" ;
  dsfv:postnummer "0603" ;
  dsfv:regdatoForAdresse "2007-01-01"^^xsd:date ;
```

CONSTRUCT-spørring som brukes til å raffinere adressedelen:

```
CONSTRUCT { ?person dsfv:harAktuellAdresse [
              rdf:type dsfv:AktuellAdresse ;
              dsfv:kommunenummer ?kommunenummer ;
              dsfv:regdatoForAdresse ?regdato ;
              dsfv:flyttedatoForAdresse ?flyttedato ;
              dsfv:gateGaard ?gateGaard ;
              dsfv:husBruk ?husBruk ;
              dsfv:bokstavFestenr ?bokstavFestenr ;
              dsfv:undernr ?undernr ;
              dsfv:adressenavn ?adressenavn ;
              dsfv:harAdressetype ?adressetype ;
              dsfv:tilleggsadresse ?tilleggsadresse ;
              dsfv:postnummer ?postnummer ;
              dsfv:valgkrets ?valgkrets ;
            ]
}
WHERE {
  OPTIONAL { ?person dsfv:kommunenummer ?kommunenummer }
  OPTIONAL { ?person dsfv:regdatoForAdresse ?regdato }
  OPTIONAL { ?person dsfv:flyttedatoForAdresse ?flyttedato }
  OPTIONAL { ?person dsfv:gateGaard ?gateGaard }
  OPTIONAL { ?person dsfv:husBruk ?husBruk }
  OPTIONAL { ?person dsfv:bokstavFestenr ?bokstavFestenr }
  OPTIONAL { ?person dsfv:undernr ?undernr }
  OPTIONAL { ?person dsfv:adressenavn ?adressenavn }
  OPTIONAL { ?person dsfv:harAdressetype ?adressetype }
  OPTIONAL { ?person dsfv:tilleggsadresse ?tilleggsadresse }
  OPTIONAL { ?person dsfv:postnummer ?postnummer }
  OPTIONAL { ?person dsfv:valgkrets ?valgkrets }
}
```

Den strukturbevarende funksjonen til adressedelen:

```
  dsfv:kommunenummer ↦ dsfv:harAktuellAdresse, dsfv:kommunenummer,
  dsfv:regdatoForAdresse ↦ dsfv:harAktuellAdresse, dsfv:regdatoForAdresse,
  dsfv:flyttedatoForAdresse ↦ dsfv:harAktuellAdresse, dsfv:flyttedatoForAdresse,
  dsfv:gateGaard ↦ dsfv:harAktuellAdresse, dsfv:gateGaard,
  dsfv:husBruk ↦ dsfv:harAktuellAdresse, dsfv:husBruk,
  dsfv:bokstavFestenr ↦ dsfv:harAktuellAdresse, dsfv:bokstavFestenr,
  dsfv:undernr ↦ dsfv:harAktuellAdresse, dsfv:undernr,
  dsfv:adressenavn ↦ dsfv:harAktuellAdresse, dsfv:adressenavn,
  dsfv:adressetype ↦ dsfv:harAktuellAdresse, dsfv:adressetype,
  dsfv:tilleggsadresse ↦ dsfv:harAktuellAdresse, dsfv:tilleggsadresse,
  dsfv:postnummer ↦ dsfv:harAktuellAdresse, dsfv:postnummer,
  dsfv:valgkrets ↦ dsfv:harAktuellAdresse, dsfv:valgkrets
```

Adressedelen av RDF-grafen etter raffineringen:

```
  dsfp:01010752171
    dsfv:harAktuellAdresse [
      a dsfv:AktuellAdresse ;
      dsfv:adressenavn "BOKS 6300, ETTERSTAD" ;
      dsfv:harAdressetype dsf:adressetype#kode-M ;
```

```
      dsfv:flyttedatoForAdresse "2007-01-01"^^xsd:date ;
      dsfv:gateGaard "00019" ;
      dsfv:husBruk "0019" ;
      dsfv:kommunenummer "0019" ;
      dsfv:postnummer "0603" ;
      dsfv:regdatoForAdresse "2007-01-01"^^xsd:date ;
    ] ;
```

B.2 Innvandring

Mappingen som er blitt anvendt oppfyller kravene til p1. Dette er fordi alle elementer som er i målgrafen, også er relatert i kildegrafen, og det eneste som gjøres er å omskrive predikatene. I tillegg har den blanke noden fått en type, og dette er også tillatt, siden `rdf:type` ikke er i verdiområde til mappingen.

Informasjon om innnvandring etter den første konverteringen:

```
  dsfp:01014600139
    dsf:innvandretFraLand dsf:land#kode-103 ;
    dsf:regdatoForInnvandring "1988-01-01"^^xsd:date ;
    dsf:flyttedatoForInnvandring "1988-01-01"^^xsd:date ;
```

`CONSTRUCT`-spørring som brukes til å raffinere en persons innvandring:

```
CONSTRUCT { ?person dsfv:harInnvandring [
                rdf:type dsfv:Innvandring ;
                dsfv:harInnvandretFraLand ?land ;
                dsfv:regdatoForInnvandring ?regdato ;
                dsfv:flyttedatoForInnvandring ?flyttedato ;
            ]
}
WHERE {
    OPTIONAL { ?person dsfv:harInnvandretFraLand ?land }
    OPTIONAL { ?person dsfv:regdatoForInnvandring ?regdato }
    OPTIONAL { ?person dsfv:flyttedatoForInnvandring ?flyttedato }
}
```

Den strukturbevarende funksjonen til innvandringsdelen:

```
  dsfv:harInnvandretFraLand ↦ dsfv:harInnvandring, dsfv:harInnvandretFraLand,
  dsfv:regdatoForInnvandring ↦ dsfv:harInnvandring, dsfv:regdatoForInnvandring,
  dsfv:flyttedatoForInnvandring ↦ dsfv:harInnvandring, dsfv:flyttedatoForInnvandring
```

Innvandringsdelen av RDF-grafen etter raffineringen:

```
  dsfp:01014600139
    dsfv:harInnvandring [
      a dsfv:Innvandring ;
      dsfv:harInnvandretFraLand dsf:land#kode-103 ;
      dsfv:regdatoForInnvandring "1988-01-01"^^xsd:date ;
```

```
      dsfv:flyttedatoForInnvandring "1988-01-01"^^xsd:date ;
    ] ;
```

B.3 Utvandring

Mappingen som er blitt anvendt oppfyller kravene til p1. Dette er fordi alle elementer som er i målgrafen, også er relatert i kildegrafen, og det eneste som gjøres er å omskrive predikatene. I tillegg har den blanke noden fått en type, og dette er også tillatt, siden `rdf:type` ikke er i verdiområde til mappingen.

Informasjon om utvandring etter den første konverteringen:

```
  dsfp:14092500250
    dsfv:harUtvandretTilLand dsf:land#kode-103 ;
    dsfv:regdatoForUtvandring "1995-10-01"^^xsd:date ;
    dsfv:flyttedatoForUtvandring "1995-10-01"^^xsd:date ;
```

`CONSTRUCT`-spørring som brukes til å raffinere en persons utvandring:

```
CONSTRUCT { ?person dsfv:harUtvandring[
                rdf:type dsfv:Utvandring ;
                dsfv:harUtvandretTilLand ?land ;
                dsfv:regdatoForUtvandring ?regdato ;
                dsfv:flyttedatoForUtvandring ?flyttedato ;
            ]
}
WHERE {
    OPTIONAL { ?person dsfv:harUtvandretTilLand ?land }
    OPTIONAL { ?person dsfv:regdatoForUtvandring ?regdato }
    OPTIONAL { ?person dsfv:flyttedatoForUtvandring ?flyttedato }
}
```

Den strukturbevarende funksjonen til utvandringsdelen:

```
  dsfv:harUtvandretTilLand ↦ dsfv:harUtvandring, dsfv:harUtvandretTilLand,
  dsfv:regdatoForUtvandring ↦ dsfv:harUtvandring, dsfv:regdatoForUtvandring,
  dsfv:flyttedatoForUtvandring ↦ dsfv:harUtvandring, dsfv:flyttedatoForUtvandring
```

Innvandringsdelen av RDF-grafen etter raffineringen:

```
  dsfp:14092500250
    dsfv:harUtvandring [
      dsf:harUtvandretTilLand dsf:land#kode-103 ;
      dsf:regdatoForUtvandring "1995-10-01"^^xsd:date ;
      dsf:flyttedatoForUtvandring "1995-10-01"^^xsd:date ;
    ] ;
```

B.4 Innflytting

Mappingen som er blitt anvendt oppfyller kravene til p1. Dette er fordi alle elementer som er i målgrafen, også er relatert i kildegrafen, og det eneste som gjøres er å omskrive predikatene. I tillegg har den blanke noden fått en type, og dette er også tillatt, siden `rdf:type` ikke er i verdiområde til mappingen.

Informasjon om innflytting etter den første konverteringen:

```
  dsfp:01014600139
    dsfv:innflyttetFraKommune "0022" ;
    dsfv:flyttedatoForInnflytting "2005-05-05"^^xsd:date ;
    dsfv:regdatoForInnflytting "2005-05-05"^^xsd:date .
```

`CONSTRUCT`-spørring som brukes til å raffinere en persons innflytting:

```
CONSTRUCT { ?person dsfv:harInnflytting [
                rdf:type dsfv:Innflytting ;
                dsfv:innflyttetFraKommune ?kommune ;
                dsfv:regdatoForInnflytting ?regdato ;
                dsfv:flyttedatoForInnflytting ?flyttedato ;
            ]
}
WHERE {
    OPTIONAL { ?person dsfv:innflyttetFraKommune ?kommune }
    OPTIONAL { ?person dsfv:regdatoForInnflytting ?regdato }
    OPTIONAL { ?person dsfv:flyttedatoForInnflytting ?flyttedato }
}
```

Den strukturbevarende funksjonen til innflyttingsdelen:

```
dsfv:innflyttetFraKommune ↦ dsfv:harInnflytting, dsfv:innflyttetFraKommune,
dsfv:regdatoForInnflytting ↦ dsfv:harInnflytting, dsfv:regdatoForInnflytting,
dsfv:flyttedatoForInnflytting ↦ dsfv:harInnflytting, dsfv:flyttedatoForInnflytting
```

Innflyttingsdelen av RDF-grafen etter raffineringen:

```
  dsfp:01014600139
    dsfv:harInnflytting [
      a dsfv:Innflytting ;
      dsfv:innflyttetFraKommune "0022" ;
      dsfv:flyttedatoForInnflytting "2005-05-05"^^xsd:date ;
      dsfv:regdatoForInnflytting "2005-05-05"^^xsd:date ;
    ]  ;
```

B.5 Postadresse

Mappingen som er blitt anvendt oppfyller kravene til p1. Dette er fordi alle elementer som er i målgrafen, også er relatert i kildegrafen, og det eneste som gjøres er å omskrive predikatene. I tillegg har den blanke noden fått

en type, og dette er også tillatt, siden `rdf:type` ikke er i verdiområde til mappingen.

Informasjon om postadresse etter den første konverteringen:

```
dsfp:01070150367
  dsfv:adresse1 "UTLANDET"  ;
  dsfv:adresse2 "LANGTVEKK" ;
  dsfv:adresse3 "0603 BORTE" .
```

`CONSTRUCT`-spørring som brukes til å raffinere en persons postadresse:

```
CONSTRUCT { ?person dsfv:harPostadresse [
                rdf:type dsfv:Postadresse ;
                dsfv:adresse1 ?adresse1 ;
                dsfv:adresse2 ?adresse2 ;
                dsfv:adresse3 ?adresse3 ;
            ]
}
WHERE {
    OPTIONAL { ?person dsfv:adresse1 ?adresse1 }
    OPTIONAL { ?person dsfv:adresse2 ?adresse2 }
    OPTIONAL { ?person dsfv:adresse3 ?adresse3 }
}
```

Den strukturbevarende funksjonen til postadressedelen:

```
dsfv:adresse1 ↦ dsfv:harPostadresse, dsfv:adresse1,
dsfv:adresse2 ↦ dsfv:harPostadresse, dsfv:adresse2,
dsfv:adresse3 ↦ dsfv:harPostadresse, dsfv:adresse3
```

Postadressedelen av RDF-grafen etter raffineringen:

```
dsfp:01070150367
  dsfv:harPostadresse [
    a dsfv:Postadresse ;
    dsfv:adresse1 "UTLANDET"  ;
    dsfv:adresse2 "LANGTVEKK" ;
    dsfv:adresse3 "0603 BORTE" .
  ] ;
```

B.6 Far, Mor og Ektefelle

Som nevnt i seksjon 4.2.2 på side 53 så eksisterer det ikke et gyldig p-map for disse `CONSTRUCT`-spørringene. Grunnen er at teorien ikke sier noe om hvordan informasjon kan fjernes på en konservativ måte. Predikatene `dsfv:farsNavn` og `dsfv:farsStatsborgerskap` blir fjernet fra RDF-grafen hvis det eksisterer en relasjon `dsfv:harFar`. Dette er fordi navnet og statsborgerskapet til faren kan hentes ved å navigere seg fra personen til faren, og derfra videre til navnet eller statsborgerskapet. Hvis det ikke eksisterer en `dsfv:harFar`-relasjon, men kun `dsfv:farsNavn` eller

`dsfv:farsStatsborgerskap`, blir det ikke lagt til en `dsfv:harFar`-relasjon, og `dsfv:farsNavn` og `dsfv:farsStatsborgerskap` blir beholdt.

Selv om det ikke eksisterer et gyldig p-map for disse spørringene, har vi har likevel valgt å utføre disse fordi det gir en bedre utnyttelse av datamodellen til RDF, og fordi det er enkelt å se at dataene ikke blir forvrengt under transformasjonene.

`CONSTRUCT`-spørring som brukes til å raffinere en persons far:

```
CONSTRUCT { ?person dsfv:harFar ?far }
WHERE {
    ?person dsfv:harFar ?far .
    OPTIONAL { ?person dsfv:farsNavn ?navn }
    OPTIONAL { ?person dsfv:harFarsStatsborgerskap ?statsb }
}
```

`CONSTRUCT`-spørring som brukes til å raffinere en persons mor:

```
CONSTRUCT { ?person dsfv:harMor ?far }
WHERE {
    ?person dsfv:harMor ?far .
    OPTIONAL { ?person dsfv:morsNavn ?navn }
    OPTIONAL { ?person dsfv:harMorsStatsborgerskap ?statsb }
}
```

`CONSTRUCT`-spørring som brukes til å raffinere en persons ektefelle:

```
CONSTRUCT { ?person dsfv:harEktefelle ?ektefelle }
WHERE {
    ?person dsfv:harEktefelle ?ektefelle ;
    OPTIONAL { ?person dsfv:ektefellePartnerNavn ?navn }
    OPTIONAL { ?person
                  dsfv:harEktefellePartnerStatsborgerskap ?statsb }
}
```

Vedlegg C

Ontologier

C.1 Ontologi for DSF

Hele ontologien er tilgjengelig på: `http://sws.ifi.uio.no/vocab/dsf/henriwi/dsf#`. URI-en `http://sws.ifi.uio.no/vocab/dsf/henriwi/dsf#` er forkortet til «:».

Klasser

```
:AktuellAdresse
  :MatrikkelAdresse
  :OffisiellAdresse
:Innflytting
:Innvandring
:Kode
  :Aarsakskode
  :Adressetype
  :Arbeidstillatelse
  :Foreldreansvar
  :Kirkemedlem
  :Personkode
  :Registreringsstatus
  :Samemanntall
  :Sivilstand
  :SpesifisertRegtype
  :Umyndiggjort
:Person
  :Barn
  :Ektefelle
  :Forelder
    :Far
    :Mor
:Postadressse
:Utvandring
```

Objektroller

:aarsakskode
  :aarsakskodeForAdresse
  :aarsakskodeForNavn
  :aarsakskodeForAdresse
:erAktuellAdresseFor ≡ :harAktuellAdresse$^-$
:harAktuellAdresse ≡ :erAktuellAdresseFor$^-$
:harBarn ≡ :harForelder$^-$
  :erFarTil ≡ :harFar$^-$
  :erMorTil ≡ :harMor$^-$
:harEktefelle (symmetrisk)
:harEktefelleStatsborgerskap
:harFarsStatsborgerskap
:harForelder ≡ :harBarn$^-$
  :harFar ≡ :erFarTil$^-$
  :harMor ≡ :erMorTil$^-$
:harInnlytting
:harInnvandretFraLand
:harInnvandring
:harKode
  :erMedlemIKirken
  :erUmyndiggjort
  :harAdressetype
  :harArbeidstillatelse
  :harForeldreansvar
  :harPersonkode
  :harSamemanntall
  :harSivilstand
  :harSpesifisertRegtype
  :harStatsborgerskap
  :harStatuskode
:harLandkodeForPostadresse
:harMorsStatsborgerskap
:harPostadresse
:harUtvandretTilLand
:harUtvandring

Dataroller

:adresse
  :adresse1
  :adresse2
  :adresse3
:adressenavn
:bokstavFestenr
:bolignummer
:datoForArbeidstillatelse
:datoForDoedsfall
:datoForForeldreansvar
:datoForNyttFnr
:datoForSamenmanntall

```
:datoForSpesifisertRegType
:datoForTidligereFnr
:datoForUmyndiggjoring
:dufId
:ektefellePartnerNavn
:familienummer
:farsNavn
:flyttedatoForAdresse
:flyttedatoForInnflyttng
:flyttedatoForInnvandring
:flyttedatoForUtvandring
:foedested
:foedestedKommLand
:fremkonnummer
:gateGaard
:grunnkrets
:husBruk
:identifikator
  :personidentifikator
    :fodselsnummer
    :nyttFodselsnummer
    :tidligereFnrDnr
:innflyttetFraKommune
:kommunenummer
:morsNavn
:navn
  :forkortetNavn
  :fornavn
  :mellomnavn
  :slektsnavn
  :slektsnavnUgift
:postnummer
:regdatoForAdresse
:regdatoForFam
:regdatoForInnvandring
:regdatoForNavn
:regdatoForPostadresse
:regdatoForSivilstand
:regdatoForStatsb
:regdatoForUtvandring
:skolekrets
:tilleggsadresse
:trygdekontornr
:undernr
:valgkrets
```

C.2 Ontologi for SED

Hele ontologien er tilgjengelig på: `http://sws.ifi.uio.no/vocab/dsf/henriwi/sed#`. Prefikset `sedv:` angir URI-en `http://sws.ifi.uio.no/vocab/dsf/henriwi/sed#`.

Klasser

```
sedv:Gender
sedv:Person
  sedv:PersonWithPin
  sedv:PersonWithoutPin
sedv:SED
```

Objektroller

```
sedv:hasGender
sedv:hasPerson
```

Dataroller

```
sedv:birthDate
sedv:caseNumberSending
sedv:dateSent
sedv:familyName
sedv:familyNameAtBirth
sedv:fatherFamilyNameAtBirth
sedv:forenameOfFather
sedv:forenameOfMother
sedv:forenames
sedv:forenamesAtBirth
sedv:institutionCode
sedv:institutionName
sedv:motherFamilyNameAtBirth
sedv:pinReceivingInstitution
sedv:pinSendingInstitution
sedv:placeOfBirth
```