AIDaS - Automatisk identifisering av stedsnavn i nyhetstekster

av
Ellen Røyneberg

Institutt for datateknikk og informasjonsvitenskap,
Norges teknisk- naturvitenskapelige universitet, Våren 2005

Sammendrag

I geografiske informasjonsgjenfinningssystem vektlegges ord og fraser som angir en geografisk lokasjon på jordens overflate. Det er ofte behov for å kunne identifisere stedsnavn, slik at disse kan være med på å danne grunnlaget for indeksering av dokumenter.

Systemet AIDaS identifiserer stedsnavn i norske nyhetstekster. AIDaS har oppnådd svært gode resultater med precison og recall på henholdsvis 80,7 og 86,7 prosent. Andre systemer som identifiserer stedsnavn i norske tekster har ikke oppnådd fullt så gode resultater. AIDaS er med andre ord et system som er godt egnet til å identifisere stedsnavn i norske nyhetstekster.

Utviklingen av AIDaS bygger på mitt høstprosjekt der jeg utviklet en abstrakt løsningsmetode for å identifisere stedsnavn. Da denne løsningsmetoden ble implementert i begynnelsen av dette prosjektet viste det seg imidlertid at den inneholdt noen svakheter. Med utgangspunkt i disse svakhetene og et oppdatert litteraturstudium utviklet jeg en ny løsningsmetode som også ble implementert. Systemet som ble implementert er kalt AIDaS.

Litteraturstudiet tok for seg to engelske systemer som fikk svært gode resultater under MUC-7, Language Technology Group System og NetOwl Extractor System. Jeg så også på det norske systemet ARNER. AIDaS er utviklet basert på ideer fra disse systemene og en abstrakt løsningsmetode utviklet i høstprosjektet.

AIDaS er et regelbasert system som identifiserer stedsnavn i norske nyhetstekster. Det vil si at AIDaS bruker regler for å klassifisere egennavnene i tekstene. AIDaS bruker også flere ressurser, blant annet Oslo-Bergen taggeren.

Reglene AIDaS bruker er utviklet med tanke på språket som brukes i nyhetstekster. Reglene ble utviklet ved hjelp av et treningssett fra Bergens Tidende. I slike tekster blir personer ofte godt introdusert med tittel og eventuelt hvilken organisasjon de har tilknytning til. Ved hjelp av konteksten er det dermed lett å lage regler som kan klassifisere personnavn og organisasjonsnavn som typen *annet*. Stedsnavn blir imidlertid ikke godt presentert i nyhetstekster. Men siden det er lett å finne typen *annet* i fra konteksten er det mulig å lage generelle regler som kan skille sted fra typen *annet*.

AIDaS bruker som nevnt flere ressurser. Den viktigste ressursen er Oslo-Bergen taggeren. Oslo-Bergen taggeren er en ekstern ressurs som tilordner et ord en eller flere ordklasser. Alle ordene som er i en tekst blir tagget og informasjonen brukes når egennavnene klassifiseres. Som hjelpemiddel under klassifiseringen bruker AIDaS også semantiske sett og lister med egennavn. De semantiske settene er grupperinger av ord som har lignende semantisk innhold og som kan hjelpe til med klassifiseringen. Listene med egennavn brukes sammen med reglene for å bestemme hvilken type et egennavn er.

AIDaS ble evaluert på en samling bestående av 50 nyhetsartikler, også fra Bergens Tidende. Før evalueringen begynte ble samlingen lest gjennom manuelt og egennavn ble klassifisert i to typer, *stedsnavn* og *annet*. Resultatet til AIDaS ble sammenlignet med den manuelle klassifiseringen. Evalueringen viste at AIDaS er et system som er

godt egnet til å identifisere stedsnavn i nyhetstekster. Skal systemet derimot skal brukes på andre samlinger, som for eksempel skjønnlitteratur, er det helt klart nødvendig med flere regler. Dette fordi personer og organisasjoner ofte blir presentert med for eksempel titler eller lignende i nyhetstekster.

Forord

Denne rapporten er skrevet i forbindelse med min diplomoppgave i sivilingeniør-studiet, datateknikk ved NTNU. Jeg vil gjerne takke hovedveileder Ingeborg Sølvberg og hjelpeveileder Øyvind Vestavik for deres veiledning gjennom prosjektet. Sølvberg har hjulpet meg med strukturering og organisering av oppgaven, mens Vestavik har bidratt med kommentarer på innhold og hjelp til tekniske utfordringer. På denne måten har deres veiledning vært med å øke kvaliteten på rapporten og systemet mitt.

Jeg vil også takke Kristin Hagen ved Tekstlaboratoriet, Universitet i Oslo for hjelp med å skaffe semantiske sett og Paul Meurer ved Universitet i Bergen for hjelp med Oslo-Bergen taggeren.

Til slutt vil jeg takke Jeanine Lilleng, Anders Hustoft og min far, Inge Røyneberg, for korrekturlesning av rapporten.

Innhold

Figurliste

Tabelliste

1 Innledning

Geografiske informasjonsgjenfinningssystem er et spesialtilfelle av et tradisjonelt informasjonsgjenfinningssystem, hvor ord og fraser som angir en geografisk lokasjon vektlegges. Egennavn er en gruppe ord som kan gi mye informasjon om innholdet i tekster. I et geografisk informasjonsgjenfinningssystem er derfor stedsnavn en viktig ressurs.

Egennavn er en spesiell type ordklasse, og kan ikke behandles på samme måte som andre ord. For eksempel er det ikke mulig å oversette dem, sjekke dem for rettskriving eller skifte dem ut med synonymer. [2] Egennavn er en åpen ordklasse som vokser med en konstant rate. Siden man ikke vet hvilke egennavn man har i fremtiden, er det vanskelig å lage en grammatikk for dem. [3]

På tross av disse utfordringene har forskning på dette området ført til utvikling av flere systemer, blant annet engelskspråklige, som har oppnådd svært gode resultater. Det er også utviklet noen norskspråklige systemer, men det gjenstår fremdeles en del forskning før disse oppnår tilsvarende resultater. En av grunnene til dette kan være at det er vanskeligere å skille mellom personnavn og stedsnavn på norsk i forhold til resten av Europa. Fra gammelt av har det vært vanlig at personer har samme etternavn som gårdsnavnet, og gårdsnavn er i flere tilfeller stedsnavn den dag i dag. Det vil med andre ord si at det vil være flere etternavn i en liste med norske stedsnavn enn for tilsvarende utenlandske lister. [4]

Stedsnavn [4], [5], [6], [7] har flere funksjoner i norsk språk, men først og fremst er det adresser som hjelper oss å orientere oss geografisk. Stedsnavnene oppstod etter hvert som det ble behov for dem for å kunne beskrive naturen. Uten stedsnavn var det ikke mulig for folk å fortelle hvor de hadde vært, og det var heller ikke lett å forklare folk veien rundt i bygde Norge. På samme måte som personnavn identifiserer personer, identifiserer stedsnavn steder. Definisjonen på et stedsnavn er i følge Lov om stadnamn som følger [6]:

> **§ 1.** *Definisjonar*
>
> *Stadnamn tyder i denne lova namn på geografiske punkt, liner og område som kan kartfestast.*

Denne oppgaven fokuserer på å utvikle en prototyp som kan identifisere stedsnavn i norske nyhetstekster på grunnlag av løsningsmetoden i beskrevet av Røyneberg [1], heretter kalt høstprosjektet. Stedsnavnene som identifiseres skal brukes i forbindelse med indeksering av tekster i geografiske informasjonsgjenfinningssystem. Det er derfor viktig å skille stedsnavn fra personnavn og organisasjonsnavn, slik at man får best mulig grunnlag for indekseringen.

Et viktig hjelpemiddel for å løse denne typen problem er å se på konteksten rundt de forskjellige egennavnene. Ofte gir forfattere informasjon som sier hvilken type et egennavn er. Men det finnes også egennavn som ikke har noen informasjon rundt seg, fordi forfatteren mener at dette bør være kjent for leseren. [8], [9] Det er derfor viktig å vite hvilket domene stedsnavnene skal identifiseres i, slik at egennavnene som burde være kjent i domenet blir lagt i lister. For eksempel, dersom domenet er Trondheims-

området, vil det sannsynligvis ikke være informasjon som tilsier at Lade er et stedsnavn, men hvis Lade blir brukt som et personnavn vil det ofte være informasjon om dette i teksten. Det er derfor viktig å først lete etter informasjon i konteksten før man tyr til oppslag i lister over stedsnavn.

For å bestemme hvilken type et egennavn er, må det utarbeides regler som kan gi en indikasjon på dette. Disse reglene må ta hensyn til konteksten, norsk grammatikk og ikke minst den interne strukturen til et egennavn. For eksempel kan det være en idé å ha en liste over typiske norske fornavn. Hvis det finnes en egennavnsekvens som inneholder to ord hvor det første er et fornavn i en liste, er det en stor sannsynlighet for at dette egennavnet er et personnavn. Selv om denne oppgaven i utgangspunktet går ut på å identifisere stedsnavn vil det være regler som identifiserer personnavn og organisasjonsnavn, som typen *annet*. Ved å lage regler for typen *annet*, vil det ved eliminasjon være enklere å bestemme om et egennavn er et stedsnavn eller ikke.

DEL I: Teori og bakgrunn

2 Høstprosjektet

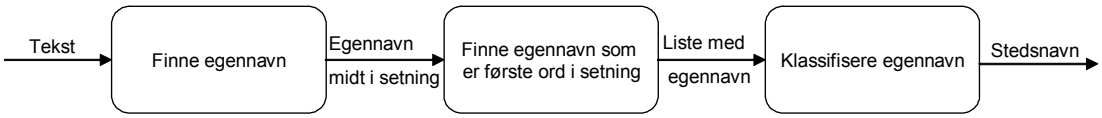
Høsten 2004 hadde jeg et prosjekt i forbindelse med faget TDT4710 Informasjonsforvaltning fordypningsemne, høstprosjektet, hvor jeg utviklet en abstrakt løsningsmetode for å identifisere stedsnavn i norske tekster. Det er derfor naturlig at jeg starter med å presentere høstprosjektet i denne oppgaven. Løsningsmetoden er basert på en litteraturundersøkelse av tre systemer for det engelske språket og et prosjekt for de skandinaviske språkene som vist nedenfor:

- Nominator [2], [10]
- Oki Information Extraction System [11], [12]
- MUlti Source Entity Finder [13], [14], [15]
- Nomen Nescio Project [16], [17], [18]

Dette kapittelet inneholder en beskrivelse av løsningsmetoden samt hvordan jeg planla å videreutvikle den.

2.1 Løsningsmetoden

Løsningsmetoden kan ses på som et rammeverk som deles inn i tre moduler, som vist i Figur 1.



Figur 1 Oversikt over løsningsmetoden [1].

Som vi kan se av Figur 1 tar den første modulen inn en tekst og identifiserer egennavnene i denne. Modulen identifiserer ord eller sekvenser av ord som er skrevet med en stor forbokstav. Modulen identifiserer dermed ikke egennavn som er første ord i en setning, bare ord som er plassert midt i setningen. Når egennavnene skal klassifiseres er det viktig å se på konteksten de er i. Modulen lager derfor en lenke fra egennavnet til dets tilhørende setning. Når hele teksten er gjennomsøkt, er det klart for neste modul.

I den andre modulen tas det hånd om første ord i setningen. Dette gjøres ved å generere en liste med alle ord som starter en setning. Disse ordene sammenlignes med egennavnene som ble identifisert i forrige modul. Hvis et av ordene i listen også er et egennavn, blir det sett på som et egennavn. Med denne algoritmen er det imidlertid muligheter for å miste egennavn der dette bare forekommer som første ord i setningen. På tross av denne svakheten har algoritmen fått gode resultater i andre systemer.

Den siste modulen klassifiserer egennavnene ved hjelp av eksternt bevis, nærmere forklart i kapittel 4.1. Dette gjøres ved hjelp av noen forhåndsdefinerte regler som ser på ordene rundt egennavnene. Reglene bruker ikke listeoppslag som hjelpemiddel under klassifiseringen. Selv om modulen i utgangspunktet er beregnet på å identifisere stedsnavn, kan den også inneholde regler for å klassifisere egennavn inn i andre typer.

2.2 Konklusjon og videre arbeid

Et av de viktigste valgene jeg gjorde i høstprosjektet var at løsningsmetoden ikke skulle bruke lister som inneholder egennavn under klassifiseringen. Dette fordi at andre norske systemer som bruker lister ikke har oppnådd så gode resultater som engelske systemer. Siden løsningsmetoden ikke ble implementert i høstprosjektet var det vanskelig å vite hvordan dette valget ville påvirke resultatet. Jeg mente imidlertid at det burde være gode muligheter for at løsningsmetoden skulle oppnå gode resultater siden alle egennavn behandles individuelt. På denne måten kan egennavnet få tildelt den typen som er best egnet i den aktuelle konteksten.

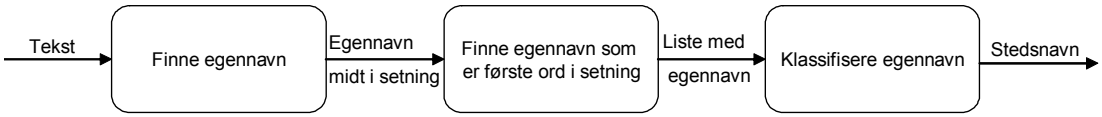
Arbeidet jeg gjorde i høstprosjektet kan ses på som et forstudium til denne oppgaven. For å forbedre løsningsmetoden er det nødvendig å se hvordan den fungere i praksis. I tillegg ville jeg gjøre et nytt litteraturstudium for å se nærmere på de to systemene som kom på første og andre plass i MUC-7, Language Technology Group System og NetOwl Extractor System. Ved å implementere løsningsmetoden og foreta et oppdatert litteraturstudium, mente jeg at det burde være gode muligheter for å lage et system som kunne oppnå gode resultater.

3 Forprosjekt

Før jeg kunne begynne å videreutvikle løsningsmetoden fra høstprosjektet, implementerte og testet jeg den. Kapittelet presenterer det arbeidet jeg gjorde i forbindelse med dette forprosjektet.

3.1 Implementasjon

Løsningsmetoden ble implementert i Java j2re1.4.2_06. Systemet var svært enkelt og inneholdt kun fem klasser, *Kontroller*, *EgennavnGjenkjenner*, *Regler*, *Egennavn* og *FilLeser*. I de følgende kapitlene forklares hva som skjer i de forskjellige klassene med referanse til oversikten over løsningsmetoden i Figur 2. (Figur 2 er en gjengivelse av Figur 1.)



Figur 2 Oversikt over løsningsmetoden [1].

3.1.1 Finne egennavn

Klassen *Kontroller* tar inn en tekst som skal klassifiseres med hensyn på egennavn. *Kontroller* oppretter så en instans av klassen *EgennavnGjenkjenner* (*EG*). *EG* tar inn hele teksten, og deler den opp i setninger. Når teksten er delt opp i setninger med tilhørende ord er det klart for å finne egennavnene i teksten.

Selv om håndteringen av første ord i setning ikke skjer i denne modulen, legges de ordene som er første ord i setning inn i en liste. Det vil si at *EG* opererer med to lister, en for første ord i setning og en for egennavn midt i setningen. Siden egennavn kan være en sekvens, sjekkes det også om de neste ordene har en stor forbokstav. Hvis dette er tilfelle blir ordene slått sammen til en egennavnsekvens. Alle egennavn som blir identifisert har en del informasjon som er viktig å ta vare på for å kunne klassifisere dem i den siste fasen. Når *EG* finner et egennavn lager den et *Egennavn-objekt*. Et *Egennavn-objekt* inneholder selve navnet, setningsnummer og posisjonen i setningen.

3.1.2 Finne egennavn som er første ord i setning

Når *EG* har funnet alle egennavnene som er midt i teksten er det klart for å sjekke om noen av ordene i listen over første ord i setning også er egennavn. Dette gjøres ved å sammenlikne ordene med egennavnene som allerede er identifisert. Hvis det viser seg at et ord er likt et egennavn, lager *EG* et *Egennavn-objekt* ut av det og legger objektet til listen med egennavn.

Under implementeringen viste det seg at algoritmen for å håndtere første ord i setning inneholdt noen svakheter som diskuteres videre i kapittel 3.2.

3.1.3 Klassifisere egennavn

Når alle egennavnene i teksten er funnet sendes listen med egennavn og teksten til klassen *Regler*. Klassen bruker lister med verb, substantiv eller lignende som brukes i sammenheng med de forskjellige egennavntypene sted, personnavn og organisasjons-

navn. Disse listene ligger lagret på disk som rene tekstfiler, og leses inn ved hjelp av *FilLeser* klassen. Når alle listene er lest inn sjekkes egennavnene opp mot reglene. Jeg implementerte kun de reglene som ble beskrevet i høstprosjektet. Når alle egennavnene er sjekket opp mot reglene gis det ut fire lister, stedsnavnliste, personnavnliste, organisasjonsnavnliste og en liste med de resterende egennavnene som ikke ble klassifiserte.

3.2 Identifiserte problemer

Tanken bak løsningsmetoden var at den skulle brukes på forskjellige typer tekster ved å bytte ut reglene i den tredje modulen. Løsningsmetoden fungerte godt på scenarioteksten fra høstprosjektet, men det viste seg at den inneholdt enkelte svakheter når teksten ble endret. Disse svakhetene reflekterte ikke svakheter med reglene, men svakheter i håndteringen av første ord i setningen og egennavn-sekvenser.

I den første modulen ble det presisert at det er viktig å ha en lenke til setningen som egennavnet tilhører. Det som derimot ikke ble nevnt var at det også er viktig å ta vare på posisjonen egennavnet har i setningen. Videre er det viktig at en egennavnsekvens blir sett på som et ord. Dette kan forklares nærmere med et eksempel fra scenario teksten [1]:

> *De to vennene **Hilde Tonstad** og Sofie Olsen bestemte seg derfor for å gå en tur i Hålandsmarka*.

I dette eksempelet skal vi se på klassifiseringen av egennavnet *Hilde Tonstad*. Her ser vi at egennavnet er i posisjon (3,4), siden første ord i setning er posisjon 0. Egennavnet skal klassifiseres ved hjelp av følgende regel [1]:

> *Hvis et egennavn er etterfylt av en konjunksjon, et personnavn og et personverb skal det klassifiseres som et personnavn.*

Når man har posisjonen til egennavnet er det enkelt å finne de ordene man er ute etter i regelen, man vet nøyaktig hvor man skal lete. Derfor skulle man ikke tro at det er et problem å klassifisere egennavnet. Det som imidlertid viser seg å være et problem er dersom *Sofie Olsen* ikke blir sett på som ett ord. Selv om *Sofie* er et personnavn er ikke *Olsen* et personverb. Dette medfører at *Hilde Tonstad* ikke blir klassifisert. Dersom *Sofie Olsen* derimot blir sett på som ett ord vil det ikke være noe problem. Det er derfor viktig å ta vare på posisjonen egennavnet har i setningen og å slå sammen en egennavnsekvens til et ord.

Når det gjelder håndteringen av første ord i setning ble algoritmen litt for enkel. Den fungerer i og for seg godt i det aktuelle scenarioet, men det viser seg at den inneholder noen svakheter. For eksempel ble det i høstprosjektet gitt en kort forklaring på hvordan man skal håndtere første ord i en setning hvis det er en sekvens [1]:

> *"Hvis en setning for eksempel begynner med ordene 'Når Hilde' tar modulen med begge ordene i listen. Når modulen sammenligner 'Når Hilde' med listen med egennavn, sjekker den først om hele sekvensen er representert. Hvis sekvensen er i listen, blir den lagt til listen med*

> *egennavn, hvis ikke sjekker modulen om 'Når' er med i listen på samme måte som tidligere."*

En ting som ikke ble nevnt her er hva man gjør med *Hilde* i slike tilfeller. Hvis *Hilde* ikke er del av det første ordet, burde *Hilde* legges til listen med egennavn, siden det er et ord med stor forbokstav som er midt i en setning. En annen svakhet som oppstår kan vises med følgende eksempel:

> ***Sofie Olsen*** *gikk en tur i Hålandsmarka. Det var kjempefint vær ute og Sofie bestemte seg derfor for å kontakte Hilde.*

Som vi ser her, er *Sofie Olsen* en egennavnsekvens som begynner en setning. *Sofie* er representert et annet sted i teksten, men ikke *Sofie Olsen*. Dette fører til at *Sofie* og *Olsen* blir lagt til egennavnlisten separat. Når man så klassifiseres i forhold til reglene i høstprosjektet er det kun *Olsen* som blir klassifisert som et personnavn. Spørsmålet her blir da om man skal behandle en *første ord i setning sekvens* som en sekvens hvis det første ordet er et egennavn. En måte å løse problemet på er å legge til en regel for dette i den tredje modulen som sier at *Sofie Olsen* blir sett på som en egennavnsekvens hvis *Olsen* er klassifisert som et personnavn og hvis *Sofie* forekommer et annet sted i teksten og er et personnavn. Eventuelt kan man identifisere personnavn ved hjelp av den indre strukturen til et egennavn.

I den andre modulen kom det ikke klart frem hvordan man bør behandle slike situasjoner som nevnt ovenfor. Dette gjelder også andre typer for egennavnsekvenser:

1. Første ord i setning er en sekvens, men det er ikke egennavnet det sammenlignes med.
2. Første ord i setning og egennavnet det sammenlignes med er en sekvens.
3. Første ord i setning er ikke en sekvens, men det er egennavnet det sammenlignes med.

Det er mulig å behandle disse situasjonene på samme måte. Hvis det finnes noen navn som er like vil dette navnet tas med i egennavnlisten, for eksempel:

1. *Hilde Marie Tonstad* sammenlignes med *Hilde*.
2. *Hilde Marie Tonstad* sammenlignes med for eksempel *Hilde Pettersen*.
3. *Hilde* sammenlignes med *Hilde Tonstad*.

Alle disse tre tilfellene fører til at *Hilde* blir lagt til listen, Når det gjelder de to første tilfellene blir *Marie Tonstad* lagt til egennavnlisten, siden dette er en sekvens som har stor forbokstav og er midt i en setning, selv om hele egennavnet egentlig er *Hilde Marie Tonstad*. I det andre tilfellet ser vi at det representerer to forskjellige personer. Fornavnet til *Hilde Marie Tonstad* vil bli sett på som en forekomst av *Hilde Pettersen*, selv om dette ikke er tilfellet.

3.3 Oppsummering og avgrensninger

I forrige kapittel ble løsningsmetoden fra høstprosjektet presentert. Løsningsmetoden ble implementert som et forprosjekt til denne oppgaven for å se hvordan den fungerer i praksis. Under implementeringen viste det seg at løsningsmetoden inneholdt noen

svakheter som det var nødvendig å forbedre for å kunne bruke løsningsmetoden i praksis.

De største problemene med løsningsmetoden er håndtering av første ord i setning og egennavnsekvenser. Å forsøke å løse problemene med gruppering av egennavn ville være en svært vanskelig oppgave og dermed ta fokus bort fra identifiseringen av stedsnavn. Det hadde derfor vært hensiktsmessig å bruke en ekstern ressurs som kan foreta denne grupperingen for meg.

De fleste problemene med løsningsmetoden viste seg å komme i forbindelse med klassifiseringen av personnavn og organisasjonsnavn. Siden denne oppgaven fokuserer på å identifisere stedsnavn kommer jeg til å kun klassifisere i to kategorier, *sted* og *annet*.

4 State of the art

Dette kapittelet inneholder relevant bakgrunnsmateriale for å videreutvikle løsningsmetoden fra høstprosjektet. I kapittel 4.1 presenteres to typer bevis som kan brukes for å klassifisere egennavn, før jeg ser nærmere på følgende tre systemer i kapittel 4.2, 4.3 og 4.4:

1. Language Technology Group System
2. NetOwl Extractor System
3. ARNER

Language Technology Group System og NetOwl Extractor System, kom på henholdsvis første og andre plass i NE oppgaven under MUC-7 konferansen [19]. Det er interessant å se hvilke teknikker som gjorde at systemene fikk så gode resultater. Det siste systemet som presenteres er ARNER [5]. ARNER er et navneentitetssystem som identifiserer stedsnavn, personnavn og verk i norske tekster. Til slutt inneholder kapittel 4.5 en diskusjon om hvorvidt lister er en viktig ressurs som bør brukes i systemer som skal identifisere og klassifisere egennavn.

4.1 Internt og eksternt bevis

Når man skal klassifisere egennavn er det mulig å benytte seg av to typer bevis for å bestemme type, internt og eksternt. Internt bevis [3] trekkes kun fra selve egennavnet. For eksempel er forkortelser som *Ltd., Mr.*, *AS* og lignende interne bevis. Bruk av lister eller gazetteers blir også sett på som interne bevis. Det er ofte vanlig at informasjonsekstraksjonssystem (IE system) bruker denne typen bevis under klassifisering av egennavn. For en beskrivelse av IE, se høstprosjektet.

Egennavn er forskjellig fra de fleste andre ordklasser. Det er ikke alltid man kan finne ut hvilken type et egennavn er uten å se på konteksten rundt navnet. Egennavnklassen er en åpen klasse, det vil si at den hele tiden vokser og at det er vanskelig å vedlikeholde en liste over alle egennavn. [3] Det er vanskelig å lage en leksikalsk grammatikk for egennavn først og fremst fordi man ikke vet hvilke egennavn som kommer i fremtiden. Man blir derfor avhengig av å se på informasjonen rundt egennavnet. Denne type informasjon blir kalt eksternt bevis. [3]

Vi kan si at navn bare er en måte å referere til individer av spesifikke typer, med karakteristiske egenskaper og som er del av karakterisktiske hendelser. Ved hjelp av konteksten kan man med en stor grad av sikkerhet bestemme hvilken type et egennavn er. I flere tilfeller vil derfor et eksternt bevis overskrive et internt bevis, som vist i eksempelet nedenfor:

*The shop **Mr. Tall** is a perfect place for men over 2 meters.*
*Butikken **Mr. Tall** er et perfekt sted for menn over 2 meter.*

Som vi kan se her ville det interne beviset klassifisert *Mr. Tall* som et personnavn på grunn av forkortelsen *Mr.*, men i denne sammenhengen passer det ikke helt. Ved hjelp av eksternt bevis er det mulig å se at *Mr. Tall* er navnet på en butikk, og at det derfor skal klassifiseres som et organisasjonsnavn. Som vi ser fra dette eksempelet er kontekst viktig når man skal klassifisere egennavn.

4.2 Language Technology Group System

Systemet som Language Technology Group (LTG) utviklet, LTG System [20], består av gjenbrukbare teksthåndteringsverktøy. Et verktøy kan ses på som en modul bestående av en strøm med inndata og utdata. Hvert verktøy gjør en spesifikk jobb, og kan kombineres med andre verktøy.

4.2.1 Preprosessering

Det første verktøyet som brukes i LTG systemet er en tokeniserer. Inndataen er en strøm av ord som deles opp i ord eller tokens. Det er verdt å merke seg at en token nødvendigvis ikke bare er ett ord, men kan være en sekvens, som for eksempel *Tony Blair Jr.* [20]. Utdataen til verktøyet er en strøm av ord som er delt inn i tokens.

Det neste verktøyet som brukes er en POS-tagger (Part of Speech). Ved hjelp av Hidden Markov Modeling tilegner POS-taggeren en token den mest sannsynlige ordklassen. POS-taggeren som brukes i LTG systemet har en avansert modul for å håndtere ukjente ord, som har vist seg å være viktig for å gjenkjenne egennavn. For at POS-taggeren skulle gi mer informasjon som er nyttige for å gjenkjenne egennavn, ble det lagt til noen utvidelser. For eksempel sjekkes alle ord som begynner med en stor forbokstav om ordet også eksister en annen plass i dokumentet med liten forbokstav eller om ordet eksisterer med liten forbokstav i en ordbok. I tillegg er det også lagt til en modul som legger til noen semantiske tagger som er nyttige i MUC sammenheng.

4.2.2 Klassifisering av egennavn

Som nevnt er egennavn mer komplekse og mer kontekstavhengige enn andre ord, og derfor må egennavnene behandles på en annen måte. Lister er et nyttig verktøy å bruke når man skal skille mellom stedsnavn, organisasjoner og lignende, men man må være kritisk til den informasjonen man får. Et egennavn kan nødvendigvis ikke kun klassifiseres som en type, for eksempel kan *Washington* både være et personnavn og et stedsnavn, avhengig av konteksten. I slike tilfeller hvor det ikke er opplagt hvilken type et egennavn er, er det vanlig at forfatteren gir denne informasjon. Man kan derfor regne med at egennavnet er av samme typen i resten av dokumentet, dersom forfatteren ikke gir ny informasjon som kan tyde på at egennavnet får en ny type. For eksempel hvis *Washington* blir brukt som et stedsnavn i teksten, vil sannsynligvis forfatteren ikke begynne å bruke navnet for å omtale en person hvis ikke ny informasjon blir lagt til konteksten. Av den grunn mener LTG at identifisering av kontekst, eksternt bevis, er svært viktig under klassifisering av egennavn.

For å kunne klassifisere egennavn som ikke har tilstrekkelig med eksternt bevis bruker LTG systemet internt bevis i form av lister. Noen av listene er dynamiske, det vil si at egennavnene som klassifiseres legges til og brukes for å klassifisere andre forekomster i samme dokument. For eksempel dersom *Washington* blir brukt som et personnavn i teksten vil det legges til listen med personnavn og brukes når resten av egennavnene i den aktuelle teksten skal klassifiseres. Egennavnene som blir lagt til de dynamiske listene blir imidlertid ikke tatt med videre når et nytt dokument skal klassifiseres.

LTG systemet går gjennom følgende fem faser for å klassifisere egennavnene (identifisere ENAMEX element):

1. Pålitelige (Sure-fire) regler
2. Delvis treff 1
3. Avslappet (Relaxed) regelbruk
4. Delvis treff 2
5. Titteltildeling

Pålitelige regler er kontekst avhengige, og aktiveres kun hvis et egennavn er omringet av en kontekst som taler sterkt for en bestem type. Eksempel på slike regler er vist i Tabell 1. I tillegg til disse reglene kan man bruke forkortelser som *Ltd*., *Inc*, *Mr*. og lignede, men når man bruker slike former for interne bevis er det viktig å sjekke dette med konteksten i denne fasen.

Tabell 1 Eksempel på pålitelige regler [20].

Kontekst regel	**Type**	**Eksempel**
Xxxxx+ is a? JJ* REL	Person	Yuri Gromov is a former director
Personnavn is a JJ* REL	Person	John White is beloved brother
Xxxxx+ a JJ* PROF	Person	White, a retired director,
Xxxxx+ ,? whose REL	Person	Numberg, whose stepfather
Xxxxx+ himself	Person	White himself
Xxxxx+, DD+	Person	White, 33
Shares of Xxxxx+	Org	Shares of Eagle
PROF of/at/with Xxxxx+	Org	Director of Trinity Motors
In/at LOC	Sted	In Washington
Xxxxx+ area	Sted	Beribidjan area

Notasjon: Xxxxx+ er en sekvens av ord med stor forbokstav; DD er et tall; PROF er en profesjon; REL er et relativ pronomen; JJ* er en sekvens av null eller flere adjektiv; LOC er et kjent stedsnavn; Personnavn er et gyldig personnavn gjenkjent av en grammatikk.

Hvis det ikke er informasjon i konteksten som kan si noe for eller mot en type blir ikke ordet klassifisert, selv om egennavnet kan være representert i en liste for en bestemt type. For eksempel blir *Washington* kun klassifisert som et stedsnavn hvis konteksten rundt det antyder et sted, som i dette tilfellet [20]:

in the Washington area
i Washingtonområdet

Denne fasen håndterer ikke sekvenser som inneholder første ord i en setning. Grunnen til dette er at man ikke kan bestemme med noen grad av sikkerhet om det første ordet i sekvensen er del av egennavnet eller ikke. Et eksempel på dette er som følger [20]:

Suspended Ceiling Contractors Ltd.

Det er ikke mulig å bestemme om *Suspended* er et ord som er del av organisasjonsnavnet, siden konteksten ikke kan gi noe informasjon om dette.

Delvis treff 1 består av to verktøy. Det første verktøyet samler in alle egennavnene som ble klassifisert i den foregående fasen. Deretter genererer vertøyet alle delvise ordninger av egennavnet, men ordningen på ordene beholdes. Hvis noen av de delvise ordningene finnes i teksten blir disse markert. Sekvensen *Lockheed Martin Productions* ville fått tildelt følgende ordninger [20]:

- *Lockheed Martin Productions*
- *Lockheed Martin*
- *Lockheed Productions*
- *Martin Productions*
- *Lockheed*
- *Martin*
- *Productions*

Hvis sekvensen ble klassifisert som et organisasjonsnavn i forrige modul er det stor sannsynlighet for at forekomster av de delvise ordningene også er et organisasjonsnavn. Derimot er det også mulig at den delvise ordningen *Martin* er et personnavn. For å være på den sikre siden, tar det neste verktøyet inn de markerte sekvensene, og ser på konteksten rundt hver sekvens. Den informasjonen som er viktigst å ta tak i er posisjonen i setningen, om ordene i sekvensen kan skrives med små forbokstaver og om de forekommer med små forbokstaver i teksten. Denne informasjonen blir sendt til beslutningsmodulen som attributter til sekvensen. Beslutningsmodulen bestemmer så om forekomsten er en organisasjon ved hjelp av en maksimum entropi modell.

Avslappet reglebruk bruker de samme reglene som ble brukt i den første fasen, men denne gangen med et mer avslappet forholdt til konteksten. Fasen tar i bruk leksikon og informasjon fra egennavn som allerede er klassifisert. I denne fasen vil for eksempel egennavnsekvenser som ligner et personnavn bli klassifisert hvis det første ordet i sekvensen er i en liste med typiske fornavn mens resten av sekvensen er ukjent. Det ville ikke vært mulig å gjøre denne klassifiseringen i den første fasen da konteksten ikke gir tilstrekkelig informasjon.

Fasen prøver også å løse konjunksjonsproblemet. Dette gjøres ved å sjekke om delene av konjunksjonsutrykket er brukt i teksten. Hvis det ikke er tilfelle blir konjunksjonen betraktet som en del av egennavnet. Fasen håndterer også problemet med første ord i setningen på en lignende måte.

Til slutt vil fasen klassifisere organisasjoner, steder eller lignende som er tilgjengelige i lister, uten å sjekke om det samsvarer med konteksten. På dette stadiet er egennavnene som er avhengige av kontekst allerede klassifisert. Det vil si at man ikke trenger å frykte at egennavn blir klassifisert feil i forhold til konteksten.

Delvis treff 2 utfører en ny delvis ordning av sekvenser slik som beskrevet i den andre fasen. På dette tidspunktet har systemet brukt alle tilgjengelige ressurser og det som da står igjen er å se om det finnes noen egennavn som ikke er blitt klassifisert, og

som er del i en annen sekvens. For eksempel blir *White* identifisert som et personnavn hvis *James White* allerede er identifisert som et personnavn.

Titteltildeling er den siste fasen og den tar høyde for tittelen til teksten. Det er ofte vanlig at flere av ordene i titler skrives med stor forbokstav, uavhengig om de er egennavn eller ikke. I denne fasen sjekkes det om det finnes andre forekomster av ordene i teksten, som fult eller delvis treff. Hvis dette er tilfelle får ordet i tittelen tildelt samme type.

4.2.3 Evaluering

Som nevnt tidligere kom LTG systemet på første plass i MUC-7 konferansen. Dette kapittelet presenterer resultatene som systemet fikk, med en tilhørende evaluering hentet fra beskrivelsen av systemet [20]. Evalueringsmålene som er brukt under evalueringen presenteres i kapittel 8.2.

LTG systemet klassifiserer som nevnt egennavn ved hjelp av fem forskjellige faser. Etter den første fasen fikk systemet en precision på ca 96 – 98 prosent, men en relativt lav recall. Det vil si at den første fasen ikke klassifiserer mange egennavn, men de som blir klassifisert er rett. I neste fase, delvis treff 1, går recall dramatisk opp, og etter hvert som systemet går gjennom de forskjellige fasene klassifiseres flere egennavn. Etter at tekstene har gått gjennom alle fasene får systemet med en samlet precision og recall (F – mål) på 93,39 prosent. Tabell 2 viser en oversikt over resultatene LTG systemet fikk i typene person, organisasjon og sted.

Tabell 2 Utdrag av evalueringsresultatet til LTG System [19] [20].

	Antall mulige	**Antall funn**	**Antall riktig**	**Recall i %**	**Precision i %**
Person	883	872	842	95	97
Organisasjon	1854	1784	1692	91	95
Sted	1308	1326	1239	95	93

Person kategorien ga ikke noen store problemer for LTG systemet. Ved noen anledninger klassifiserte systemet feil på grunn av at sekvensen lignet et personnavn, selv om det ikke var det. Men de fleste feilene var ord som kun forekom i teksten en gang, og som ikke hadde nok informasjon rundt seg til å foreta en klassifisering.

Organisasjonene førte til noen feilklassifiseringer. For eksempel var det ikke alltid systemet klassifiserte rett når uttrykket *Granada* ble brukt for organisasjonen *Granada Group Plc*. Grunnen til dette er at *Granada* også er et sted og systemet mente at det var mest sannsynlig at det ble referert til stedet og ikke organisasjonen. Det ble også gjort noen feilklassifiseringer på grunn av feil håndtering av konjunksjonsutrykk.

Sted var den kategorien LTG var mest misfornøyd med, da de fleste feilene var forårsaket av ett ord, *Columbia*. *Columbia* er et stedsnavn i listene til LTG systemet, men i denne sammenhengen var det ment som navnet på et romfartøy. I utgangspunktet skulle derfor ikke *Columbia* bli klassifisert som et stedsnavn, men på grunn av følgende setning gikk det imidlertid ikke slik [20]:

a satellite 13 miles from Columbia
en satellitt 13 miles fra Columbia

Denne konteksten gir inntrykk av at *Columbia* er et stedsnavn, og derfor vil den første fasen klassifisere det som et stedsnavn siden navnet også er representert i en liste med stedsnavn. Av totalt 55 feil var 30 av dem grunnet navn på forskjellige romfartøy.

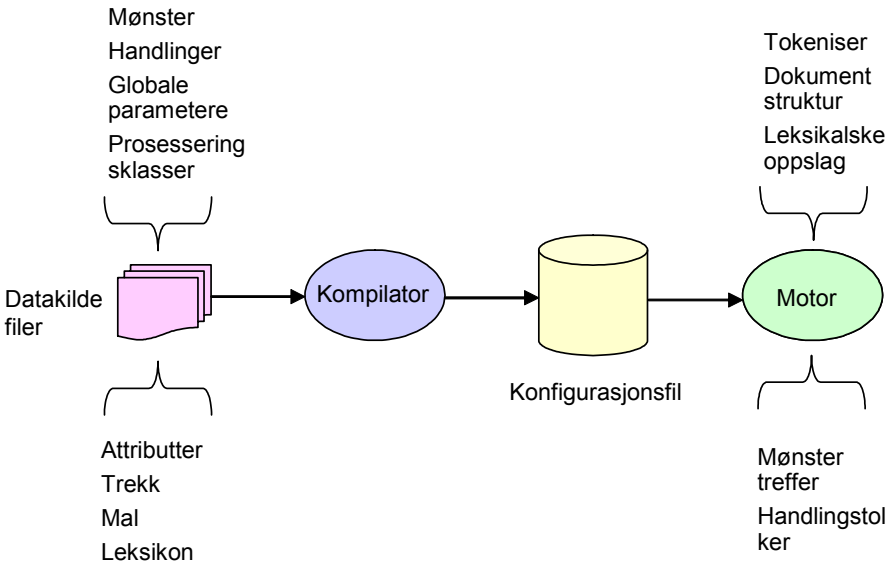
4.3 NetOwl Extractor System

NetOwl Extractor System [8] (NOES) er utviklet av Isoquest og er et kommersielt system, som deltok i MUC-7. Systemet består av en høyhastighets C++ motor som analyserer tekst basert på en konfigurasjonsfil som inneholder regler og leksikon. For å gjenkjenne egennavnene i tekstene brukes Isoquest NameTag Configuration, deretter må man tilpasse NOES sine tagger til MUC-7 NE taggene.

4.3.1 Systembeskrivelse

Arkitekturen til NOES separerer prosesseringsmotoren fra konfigurasjonsdata som spesifiserer hvilken prosessering man skal utføre. Motoren har derfor ingen konfigurasjonsspesifikk kode eller informasjon. Konfigurasjonsfilen inneholder spesifikk prosessering og annen data som er nødvendig for å gjenkjenne egennavn.

Figur 3 gir en oversikt over NOES. Som vi kan se, lages det en konfigurasjonsfil ut fra datakildefiler som inneholder forskjellige attributter og parametere. Attributtene og parametrene gjør at det er mulig å få individuelle prosesseringer av forskjellige domener. I tillegg får man også et mer effektivt minnebruk og initialiseringstiden reduseres.



Figur 3 Oversikt over NetOwl Extractor kompilering og utføring [8].

Konfigurasjonsfilen i Figur 3 inneholder karakteristikker som brukes av motoren under prosessering. For eksempel er det leksika og mønsterregler som bestemmer hva motoren kan gjenkjenne, malene og handlingsdefinisjonene bestemmer hva motoren

skal hente ut og prosesseringsklassene bestemmer hvilke prosesseringsfaser motoren skal utføre.

4.3.2 Klassifisering

Konfigurasjonsfilen brukes for å gjenkjenne egennavn. En slik gjenkjenning krever spesiell lingvistisk kunnskap om strukturen og ordningen til forskjellige typer navn. For eksempel består personnavn vanligvis av et fornavn, i noen tilfeller et mellomnavn og et etternavn. Men denne informasjonen alene er ikke tilstrekkelig for å gjenkjenne egennavn, det er også svært viktig å se på hvordan navnene forekommer i tekstene.

Det er ikke alltid det er informasjon i tekstene som kan hjelpe til med klassifiseringen. Derfor bruker NOES lister (leksika) for å identifisere dem. Disse inneholder ofte kjente navn slik som for eksempel landet *Frankrike*. Det er ikke mulig å si at *Frankrike* er et stedsnavn kun ved å se på selve ordet, og siden landet er så kjent utelater ofte forfatteren å gi informasjon i konteksten. Derfor er det viktig å vite i hvilket domene man skal gjenkjenne egennavn i, slik at man kan lage en liste over egennavn som er antatt kjente blant leserne og som derfor ikke har noe eksternt bevis. NOES bruker dermed både internt og eksternt bevis for å klassifisere egennavn.

For å gjenkjenne forkortelser av egennavn har NOES regler som genererer de mest sannsynlige forkortelsene av egennavn, som man deretter kan bruke for å klassifisere dem. Det er derfor ikke nødvendig at listene med egennavn skal inneholde forkortelse av egennavnene.

NOES har en egen fase, regelkonkurranse, for å bestemme hvilken type et egennavn mest sannsynlig er. For å gjennomføre denne konkurransen må hver regel ha vekter som sier hvor stor sannsynlighet det er for at egennavnet er av en viss type. Disse vektene summeres i forhold til typer, en høy vekt indikerer strekt bevis, lav vekt indikerer svakt bevis, mens negativ vekt er en sterk indikasjon på at egennavnet ikke er av denne typen. Fasen tar også i betraktning egennavnene som er i leksikonene. Konkurransen avsluttes ved at man velger den typen som har fått høyest sum.

4.3.3 Evaluering

Under MUC-7 deltok NOES på to forskjellige gjennomkjøringer. Den første gjenomkjøringen var den offisielle hvor man fokuserer på å klassifisere flest egennavn, det vil si man vil oppnå de beste resultatene uavhengig av kjøretiden. I den andre gjennomkjøringen, valgfri gjennomkjøring, ble kun 20 prosent av reglene brukt for å øke ytelsen. Det vil si at systemet fikk dårligere resultater, men raskere utføring. For en oversikt over resultatene NOES fikk under disse gjennomkjøringene se Tabell 3.

Tabell 3 Evalueringsresultatet til NetOwl Extraction System [8].

	Recall i %	**Precision i %**	**F-mål i %**	**CPU Tid i sek**	**Hastighet Meg/time**
Offisiell	90	93	91,60	3,6	382
Valgfri	74	93	82,61	2,7	513

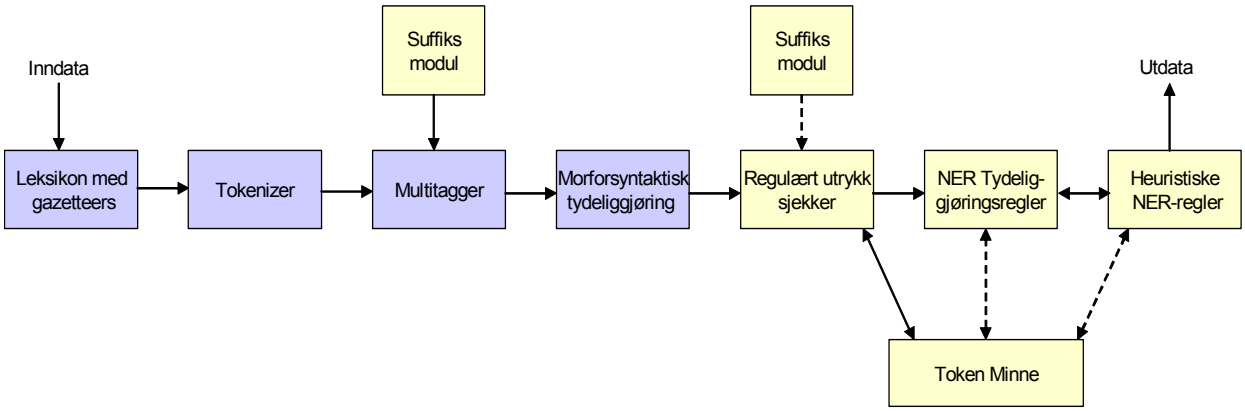
Som vi kan se av Tabell 3 er det flere egennavn som ikke blir klassifisert når man ikke bruker alle reglene, men precision forblir den samme. Tabell 4 viser et utdrag av evalueringsresultatet til den offisielle gjennomkjøringen.

Tabell 4 Utdrag av evalueringsresultatet til den offisielle gjennomkjøringen av NOES [19].

	Antall mulige	Antall funn	Antall riktig	Recall i %	Precision i %
Organisasjon	1852	1829	1620	87	89
Person	885	858	835	94	97
Sted	1380	1291	1222	93	95
Totalt	11498	11062	10333	90	93

4.4 Automatic Rulebased Named Entity Recognizer for Norwegian

Automatic Rulebased Named Entity Recognizer for Norwegian (ARNER) [5] er et navneentitetssystem for norske tekster. Systemet er en videreutvikling av Oslo-Bergen taggeren, hvor det er lagt til noen regler og sett i Oslo-Bergen taggeren slik at den kan brukes i forbindelse med navnegjenkjenning. Figur 4 viser en oversikt over ARNER systemet, hvor de blå delene er fra Oslo-Bergen taggeren, mens de gule er nye deler som er lagt til for navnegjenkjenning. De stiplede pilene illustrerer fremtidig interaksjon mellom de forskjellige modulene i systemet. Ved å legge til de nye delene får Oslo-Bergen taggeren semantisk informasjon.



Figur 4 Oversikt over ARNER [5].

4.4.1 Semantikk

Den eneste semantiske informasjonen ARNER har i sine lister er merkelapper som sier hvilken type et egennavn er. Andre ord og uttrykk har ikke denne form for merkelapp.

For å få semantikk inn i ARNER, grupperes substantiv, verb og andre ord i sett som typisk hører til en bestemt kategori. De semantiske settene inneholder ord med liknende semantisk innhold. For eksempel er det mulig å gruppere preposisjonene *østover*, *øst* og lignende sammen, siden disse ofte refererer til retning i forbindelse med stedsnavn.

Noen verb foretrekke argumenter av en bestemt type. For eksempel foretrekker verbet å *bjeffe* substantivet en *hund*. Slike semantiske regulariteter kalles *seleksjons-preferanser*. Man kan bruke seleksjonspreferanser i forbindelse med klassifisering av stedsnavn, for eksempel verb som indikerer bevegelse sammen med ordene *til* og *fra*: [5]

Dina flyktet fra Sarajevo.

Som vi kan se av eksempelet over, indikerer ordene *flyktet fra* et stedsnavn. Men i noen tilfeller vil ikke slike regler fungere: [5]

Dina kjørte fra Peter.

I utgangspunktet er *kjørte* et verb som sammen med ordene *til* eller *fra* vil indikere et stedsnavn, men som vi ser av eksempelet er ikke dette alltid tilfelle. Man må derfor ta høyde for slike situasjoner når man skal klassifisere stedsnavn på grunnlag av seleksjonspreferanser.

Å gruppere ord som har lik semantisk innhold i en kontekst kan være farlig. Et ord kan ha forskjellige betydninger i forhold til hvilken kontekst det er i, og det er derfor farlig å generalisere. Løsningen på dette problemet er å lage subsett av noen sett. Subsettene inneholder ord som er farlige i noen kontekster, slik at disse ikke tas med i betraktning når man klassifiserer i spesielle kontekster.

4.4.2 Regler for semantisk tagging

Dette kapittelet inneholder en del eksempler på regler for å klassifisere egennavn som blir brukt i ARNER [5]. Reglene som brukes i ARNER kalles CG-regler. En CG-regel er en regel som bygger på betingelsesgrammatikk (constraint grammar).

ARNER har tre forskjellige typer navneentitietsregler:

1. Mappingregler
   Gir alle mulige semantiske kategorier til et egennavn uten noen betingelser.
2. Seleksjonsregler
   Velger en semantisk kategori, hvis alle kontekstbetingelser er tilfredsstilte.
3. Forkastningsregler (discarding)
   Forkaster kategorier som ikke kan brukes i den aktuelle sammenhengen.

De fleste CG-reglene i ARNER bruker eksternt bevis for å klassifisere. Hvis man skal identifisere stedsnavn uten bruk av lister er man avhengige av regler som ser på konteksten. Et eksempel på en regel som ikke bruker lister under klassifisering: [5]

```
(@w =n! (&sted)
        (1 abj#komma/%være%)
        (*2 abj#ikke-adv-adj-det-prep-komma LRO)
        (NOT LRO abj#%vei%)
        (LRO abj#sted-subst LRO))
```

Denne regelen klassifiserer ordet på posisjon 0 hvis:

1. Ordet på posisjon 1 er et *komma* eller en variant av verbet *være*.
2. Finn et ord som er til høyre for ordet i posisjon 2 som ikke er et adverb, adjektiv, determinant, preposisjon eller komma og gi ordet benevnelsen LRO.
3. LRO kan ikke være ordet *vei*.
4. LRO er medlem av listen som inneholder substantiv typisk for stedsnavn.

Regelen kan klassifisere følgende setninger: [5]

Hauktjern er et sted.
Hauktjern er det mest fantastiske og ikke minst fantasifulle stedet jeg vet om.

I den første setningen ser vi at ordet på posisjon 1, *er*, er en variant av verbet *være*. Ordet som er på posisjon 2 er av typen determinant, og får derfor ikke tildelt benevnelsen LRO. Derimot tilfredsstiller ordet på posisjon 3, *sted*, betingelsene og får tildelt betegnelsen LRO. Siden LRO ikke er ordet *vei* og *sted* er et typisk substantiv for stedsnavn, blir Hauktjern klassifisert som et stedsnavn.

På samme grunnlag som klassifiseringen over blir *Hauktjern* i den andre setningen også klassifisert som et stedsnavn. Man kan gjøre dette da ordene *det mest fantastiske og ikke minst fantasifulle* blir oversett av regelen.

I noen tilfeller kan det være til hjelp å bruke lister med stedsnavn for å klassifisere. Nedenfor følger et eksempel på en regel som bruker lister i tillegg til eksternt bevis for å klassifisere et stedsnavn: [5]

```
(@w =n! (&sted)
        (0 abj#sted)
        (-2 abj#%reise%)
        (-1 prep))
```

Denne regelen vil være til hjelp i følgende tilfeller: [5]

Lisa reiser til Italia
Lisa reiser til Maria.

Som vi kan se vil regelen klassifisere *Italia* som et stedsnavn, men kun hvis *Italia* er med i en liste over stedsnavn. Siden regelen baserer seg på at egennavnet som skal klassifiseres må være i en liste med stedsnavn, slipper man dermed en feilklassifisering i neste tilfelle. Uten en slik liste ville det vært umulig å se forskjell på *Italia* og *Maria*.

Det er også mulig å lage regler kun på grunnlag av interne bevis, for eksempel ved å se på suffikset til stedsnavn: [5]

```
(@w =n!h1 (&sted)
        (0 abj#sted-suffiks)
        (NOT 0 abj#person))
```

Denne regelen klassifiserer et egennavn som et stedsnavn hvis slutten av navnet inneholder et stedssuffiks. Eksempel på slike suffiks er *skog*, *park* eller lignende. Det er viktig å merke seg at egennavnet ikke blir klassifisert dersom egennavnet er klassifisert som en person. Det vil si at dette er en regel som bør kjøres etter alle reglene som bruker eksterne bevis er kjørt for å unngå feilklassifiseringer.

4.4.3 Evaluering

ARNER har svært spesifikke regler som fungerer godt der de skal [18]. Tabell 5 gir en oversikt over evalueringsresultatet til ARNER. Som vi kan se fikk systemet en meget god recall, men en lav precision.

Tabell 5 Utdrag av evalueringsresultatet til ARNER [5].

	Recall i %	**Precision i %**
Person	98,26	57,72
Sted	99,56	28,30
Organisasjon	90,63	20,03
Totalt	96,90	18,94

4.5 Bruk av lister

En viktig ting man må ta stilling til når man skal lage et system som skal klassifisere egennavn er om man skal bruke lister (les: lister eller gazetteers) for å bestemme type. Egennavn er en ordklasse som hele tiden vokser med en konstant rate og det er derfor vanskelig å oppdatere listene. I Norge finnes det flere millioner stedsnavn [4] og listene måtte derfor over tid blitt uendelig. Selv om det er vanskelig å vedlikeholde lister er det flere som bruker dem, men da gjerne i tillegg til andre regler. Det finnes også en del systemer som ikke bruker lister i det hele for å slippe å drive med vedlikehold, og som baserer seg utelukkende på den informasjonen som er i teksten.

I høstprosjektet ble det konkludert med å ikke bruke lister som et hjelpemiddel under klassifiseringen. Jeg skal nå ta opp denne problemstillingen for å se om dette er et hensiktsmessig valg. Grunnlaget for diskusjonen er hentet fra tester av LTG systemet med og uten bruk av lister [9], høstprosjektet og beskrivelsen av NetOwl Extraction System [8].

Det er ved flere anledninger stilt spørsmål om hvor viktig det er å bruke lister. Noen systemer bruker store lister med flere tusen egennavn. For eksempel inneholdt listen som Isoquest systemet brukte i MUC-7 110 000 egennavn. Det er derimot vist at systemytelsen ikke endret seg mye ved å redusere listene, se Tabell 6. Som vi kan se av tabellen har størrelsen på listene kun en liten betydning på ekstraksjonsytelsen. Mikheev, Grover og Moens [9] hevder også at NOES fikk en betraktelig økt ytelse med bare 42 egennavn.

Tabell 6 Resultatene til NetOwl Extractor System med varierende listestørrelser [8].

	Antall egennavn	**Prosess størrelse (Meg)**	**F – mål**
Maksimum	110 000	7,0	91,60
Medium	25 000	4,0	91,45
Minimum	9000	3,7	89,13

4.5.1 Enkelt system – kun listeoppslag

For å finne ut hvor viktig lister er i klassifisering av egennavn utviklet LTG [9] et enkelt system som kun bruker lister. Listene ble laget ved hjelp av treningssett, et sett av manuelt lagde lister og til slutt en kombinasjon av disse to listene. Den typen som fikk best resultat ved bruk av lister var stedsnavn. Tabell 7 viser en oversikt over resultatene man fikk ved hjelp av lister.

Tabell 7 Resultatene ved kun bruk av lister [9].

	Treningssett lister		**Manuelt lagde**		**Kombinasjon**	
	Recall	Precision	Recall	Precision	Recall	Precision
Organisasjon	49	75	3	51	50	72
Person	26	92	31	81	47	85
Sted	76	93	74	94	86	90

4.5.2 Kombinasjon av regler og lister

Etter å ha laget et system som kun brukte lister, tok LTG utgangspunkt i LTG systemet som ble utviklet for MUC-7. På denne måten fikk man mulighet til å kombinere regler og lister av forskjellige størrelser. Det vil si at man klassifiserer egennavn ved hjelp av interne og eksterne bevis. For å finne ut hvor stor innvirkning listene har på resultatet ble det kjørt fire tester:

1. Fulle lister
2. Uten lister
3. Noen stedsnavn
4. Begrensede lister

Resultatene til alle de fire testene er vist i Tabell 8.

Tabell 8 Resultatene fra tester med og uten lister [9].

	Fulle lister		**Uten lister**		**Noen stedsnavn**		**Begrensede lister**	
	Recall	Prec.	Recall	Prec.	Recall	Prec.	Recall	Prec.
Org	90	93	86	85	87	89	87	90
Person	96	98	90	95	90	97	92	97
Sted	95	94	46	59	85	90	91	92

I den første testen, *fulle lister*, brukte man de samme listene som ble brukt under MUC-7. Listene inneholdt 4900 stedsnavn, cirka 30 000 navn på organisasjoner og cirka 10 000 fornavn på personer. Siden korpuset som ble brukt i denne testen var litt

annerledes enn korpuset fra MUC-7 var ikke resultatet identisk med det offisielle MUC resultatet.

For å se hvor stor innvirkning store lister hadde på resultatet, var den andre testen *uten lister*. Det var fremdeles mulig å bruke eksterne bevis og noen former for interne bevis, som for eksempel *Inc*. og *Mr*.. Som forventet ble resultatet for organisasjons- og personnavn svært bra. Grunnen til dette er at disse typene ofte har mye ekstern og intern informasjon, slik at det blir enkelt å klassifisere dem. Stedsnavn derimot har ikke like mye informasjon i konteksten, dermed ble resultatet betraktelig dårligere enn med bruk av fulle lister.

Siden stedsnavn ikke fikk så gode resultater uten bruk av lister, bestemte man seg for å ha lister med *noen stedsnavn*. Denne listen kan ses på et supplement for å gi bedre resultat på klassifisering av stedsnavn. Listen inneholdt 200 svært kjente stedsnavn, og ble lagt til i LTG systemet, noe som førte til et betydelig bedre resultat.

Den siste testen som ble gjort inneholdt *begrensede lister*. Som man kunne se av forrige test økte resultatet bare ved å legge til noen kjente stedsnavn. Det ble derfor bestemt å lage lister ved hjelp av et treningssett på 30 artikler. Artiklene var innen samme domene som testsamlingen, og man håpte derfor å trekke ut kjente egennavn innen det domenet. Når et egennavn ble klassifisert ble det lagt til en liste slik at det kan brukes for å klassifisere andre egennavn i en annen tekst. Stedsnavnene som ble funnet i de 30 artiklene ble lagt til de 200 fra den foregående testen. Som forventet ble resultatene forbedret, og dette selv om listene kun inneholdt egennavn fra 30 artikler.

Som vi kan se av resultatene i Tabell 8 fikk systemet relativt gode resultater uten bruk av lister for typene person og organisasjon. Stedsnavn derimot fikk skuffende resultat. Hvis det kun legges til et lite antall stedsnavn i en liste ble resultatene til stedsnavn betraktelig bedre. Grunnen til dette [9] er at de fleste stedsnavn er kjente og det er derfor ikke nødvendig for forfatteren å gi informasjon i konteksten til leseren. Slik informasjon er ikke implisitt kunnskap for et navnegjenkjenningssystem, derfor kan en liste med kjente stedsnavn tilby denne kunnskapen.

4.6 Oppsummering

LTG og Isoquest har utviklet to svært gode systemer for å identifisere og klassifisere egennavn. Systemene løser problemstilingen på forskjellige måter, men hvis man ser etter, brukes de samme grunnprinsippene. Selv om systemene er utviklet til å gjenkjenne egennavn i engelske tekster bør det være mulig å benytte disse prinsippene i et system for norske tekster.

ARNER er mer ulik LTG systemet og NOES. En grunn til dette er at ARNER er en utvidelse av Oslo-Bergen taggeren, ikke et helt system. Det vil derfor ikke være like gunstig å bruke oppbygging til ARNER som utgangspunkt for å utvide løsningsmetoden fra høstprosjektet. Derimot kan reglene i ARNER være en god pekepinne på hvilke regler som bør inkluderes i et norsk system for identifikasjon av stedsnavn. En ulempe med reglene i ARNER er at de er spesifikke. Å lage spesifikke regler er tidkrevende og det er vanskelig å forutse hva som kan komme i en ukjent tekst. Jeg valgte derfor å prøve å lage regler som er generelle, men som allikevel ville gi gode resultater.

Som nevnt tidligere, konkluderte jeg i høstprosjektet at jeg ikke skulle bruke lister med stedsnavn som et hjelpemiddel under klassifiseringen. Etter diskusjonen om bruk av lister mener jeg at det er god grunn for å bruke lister med egennavn når løsningsmetoden skal videreutvikles.

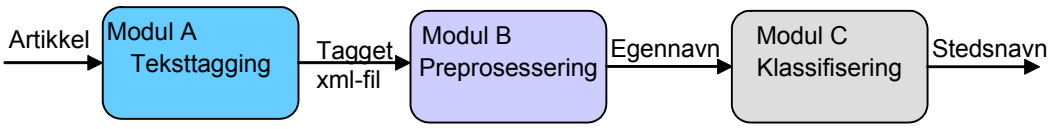
DEL II: Mitt system - AIDaS

5 AIDaS og ressurser

Løsningsmetoden fra høstprosjektet inneholdt noen svakheter som ble synlig da den ble realisert. På grunnlag av disse svakhetene og et oppdatert litteraturstudium har jeg utarbeidet en ny løsningsmetode. Løsningsmetoden danner grunnlaget for systemet jeg har implementert, AIDaS (Automatisk identifikasjon av stedsnavn). Dette kapittelet inneholder en overordnet beskrivelse av AIDaS, i tillegg gis det en oversikt over hvilke ressurser AIDaS bruker.

5.1 Overordnet beskrivelse

Figur 5 viser en overordnet oversikt av AIDaS. Systemet består av tre moduler, A *Teksttagging*, B *Preprosessering* og C *Klassifisering*. Etter at en tekst har gått gjennom disse modulene, får man ut en liste over stedsnavn i teksten.



Figur 5 Overordnet oversikt over AIDaS.

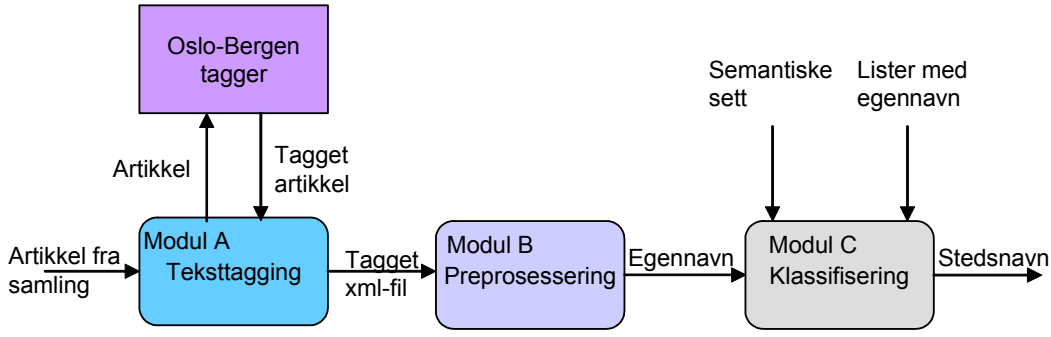
Hvis vi sammenligner AIDaS med løsningsmetoden fra høstprosjektet i Figur 1 kapittel 2.1, kan vi se at det er gjort noen endringer. Modulen *Finne egennavn* har fått navnet *B Preprosessering* i den nye løsningen, mens modulen *Finne egennavn som er første ord i setning* er fjernet. Funksjonaliteten til den modulen er nå fordelt mellom modul *B* og *C*.

5.2 Ressurser i AIDaS

Figur 6 viser hvilke ressurser AIDaS bruker. Som vi kan se av figuren bruker AIDaS følgende ressurser:

1. Oslo-Bergen taggeren
2. Artikkelsamlinger
3. Lister med egennavn
4. Semantiske sett

De følgende kapitlene inneholder en beskrivelse av samtlige ressurser.



Figur 6 AIDaS med ressurser.

5.2.1 Oslo-Bergen taggeren

Oslo-Bergen taggeren [5], [21] er utviklet på Tekstlaboratoriet ved Universitetet i Oslo. I dag videreutvikles og vedlikeholdes taggeren av Paul Meurer ved Universitet i Bergen.

Taggeren er en *part-of-speach* (POS) tagger som kan ta inn en ukjent norsk tekst og analysere den. Det vil si at hvert ord i teksten får tildelt en tagg som angir hvilken grammatisk informasjon det har, som for eksempel ordklasser. I utgangspunktet er man kun interessert i at et ord får tildelt en tagg, men da flere ord i det norske språket er tvetydige, kan dette være vanskelig. I slike tilfeller må taggeren velge det som er mest sannsynlig i det aktuelle tilfellet. [21]

Som forklart i høstprosjektet finnes det to typer taggealgoritmer, regelbaserte taggere og stokastiske taggere. Oslo-Bergen taggeren er en føringsbasert tagger (constraint-based tagger). Føringsbaserte taggere er regelbaserte taggere som har lingvistiske regler for hver tydeliggjøring (disambiguering). Oslo-Bergen taggeren lager ikke fraser for å gi ordene en ordklasse, den ser kun på forholdet mellom enkeltord. [21]

Før Oslo-Bergen taggeren begynner å tilegne ordene en ordklasse må inndataen preprosesseres. I preprosesseringen skjer følgende [21]:

- Skille ut overskrifter
- Gjenkjenne setninger
- Gjenkjenne datoer
- Gjenkjenne koordinerte sammensetninger (for eksempel *etter- og videreutdanning*)
- Gjenkjenne faste uttrykk (for eksempel at *blant annet* blir sett på som et ord)

Etter at teksten er preprosessert går den gjennom en morfosyntaktisk multitagger. Denne taggeren prøver å finne alle ordene i teksten i en leksikalsk database. Ordene får tildelt alle treffene man finner i databasen. For ord som ikke får treff prøver man å se om det er et sammensatt ord ved hjelp av et sammensetningsprogram. Hvis det viser seg at ordet er sammensatt slår man kun opp det siste leddet i ordet. Hele sammensetningen får tildelt ordklassene som det siste leddet. Multitaggeren gjenkjenner også egennavn hvis de finnes i en forhåndsdefinert ordliste. Til slutt prøver man å velge hvilken ordklasse som passer best til det bestemte ordet, ved hjelp av tydeliggjørings taggere (disambiguerende taggere). [21]

Oslo-Bergen taggeren ble evaluert på en treningssamling med omlag 100 000 ord, og fikk en recall på 99,2 prosent og en precision på 96,8 prosent. [22]

Oslo-Bergen taggeren er en viktig ressurs i AIDaS. Informasjonen fra Oslo-Bergen taggeren brukes for å klassifisere egennavnene. Taggeren har en algoritme for å slå sammen egennavnsekvenser og å håndtere komplekse egennavn. Det er derfor ikke nødvendig for AIDaS å inneholde denne typen funksjonalitet. Oslo-Bergen taggeren kan gi ut informasjon om egennavn, men AIDaS bruker ikke denne informasjonen under klassifiseringen. Grunnen til dette er at Oslo-Bergen taggeren bruker listeoppslag for å klassifisere egennavn. Disse listene er ikke fullstendige, det vil si at det er mange egennavn som ikke blir klassifisert.

For å bruke Oslo-Bergen taggeren har jeg fått tilgang til et SOAP-grensesnitt hvor all kommunikasjon skjer ved hjelp av xml-meldinger. Jeg har selv mulighet til å kontrollere forskjellige konfigurasjonsparametre, blant annet hvilke ordklasser eller liknende jeg ønsker at skal kunne tilordnes i taggingen.

Figur 7 viser et eksempel på hvordan et utdrag av en xml-fil fra Oslo-Bergen taggeren kan se ut. Det er viktig å merke seg at denne filen er resultatet etter at jeg har gjort operasjoner på meldingen fra Oslo-Bergen taggeren. Disse operasjonene forklares nærmere i kapitel 6. Som vi kan se av Figur 7 gir Oslo-Bergen taggeren ut mye informasjon om ordene i teksten. For eksempel har alle ord et lemma. Et lemma [23] kan ses på som en felles betegnelse på et sett av leksikalske former som har samme rot. På grunnlag av det kan man si at *politi* er lemma til *politiet*, når det gjelder verb er det ofte slik at det er infinitivsformen av verbet som er lemmaet. Informasjon om ordklassene gis i attributtet *features*. Som vi kan se inneholder variabelen også informasjon om bøyningen av substantiv, og i noen tilfeller hvilken rolle ordet har i setningen, som for eksempel ser vi at *Økokrim* er subjektet i setningen.

```
- <set>
    <word id="w270" lemma="Økokrim" features="subst prop @subj">Økokrim</word>
    <word id="w271" lemma="komme" features="verb">kom</word>
    <word id="w272" lemma="sammen" features="adv">sammen</word>
    <word id="w273" lemma="med" features="prep">med</word>
    <word id="w274" lemma="politi" features="subst be ent">politiet</word>
    <word id="w275" lemma="her" features="prep">her</word>
    <word id="w276" lemma="i" features="prep">i</word>
    <word id="w277" lemma="England" features="subst prop">England</word>
  </set>
```

Figur 7 Utdag av xml-fil fra Oslo-Bergen taggeren.

Som nevnt er Oslo-Bergen taggeren en viktig ressurs for AIDaS, og uten dette hjelpemiddelet ville det vært svært vanskelig å oppnå gode resultater. Dessverre har det vist seg at taggeren i flere tilfeller gir ut feil informasjon som fører til feilklassifisering i AIDaS. De feilene som gir mest utslag er feil på grunn av at sekvenser ikke grupperes rett og at ord får tildelt feil ordtagg. Her følger noen eksempler på slike feil:

1. *I **Kong Oscars gate** i Bergen ble fotgjengere i går tvunget ut i veibanen.*
   I denne sammenhengen burde *Kong Oscars gate* bli gruppert sammen til en sekvens, men dette er ikke tilfellet. Figur 8 viser hvordan Oslo-Bergen taggeren grupperte egennavnet. På grunn av denne feilen blir *Oscars* sett på som en person, i tillegg til at man mister et stedsnavn.

```
<word id="w150" lemma="kong" features="subst @tittel">Kong</word>
<word id="w151" lemma="Oscar" features="subst prop">Oscars</word>
<word id="w152" lemma="gate" features="subst fem ub ent">gate</word>
```

Figur 8 Eksempel på feil gruppering 1.

2. *Lensmannsbetjent Geir Stien ved **Fjell og Sund lensmannskontor** sier til Bergens Tidende at …*
   Fjell og Sund lensmannskontor er i dette tilfellet hele navnet. Som vi kan se av Figur 9 ble ikke *Fjell og* en del av *Sund lensmannskontor*. Dette fører til at *Fjell* blir klassifisert som et stedsnavn av AIDaS, selv om det egentlig refererer til en organisasjon.

```
<word id="w87" lemma="Fjell" features="subst prop">Fjell</word>
<word id="w88" lemma="og" features="">og</word>
<word id="w89" lemma="Sund lensmannskontor" features="subst prop @subj">Sund lensmannskontor</word>
```

Figur 9 Eksempel på feil gruppering 2.

3. *Informasjonsforbudet som rammer bankansatte **i** Bergen og Hordaland, er grunnen til at vi annonserte i BT.*
   Som vi kan se av Figur 10 blir *i* tagget som en tittel. Tittel er en god indikasjon på at egennavnet er et personnavn, derfor blir Bergen klassifisert som et personnavn, selv om det egentlig er et stedsnavn.

```
<word id="w403" lemma="i" features="subst mask ub ent @tittel">i</word>
<word id="w404" lemma="Bergen" features="subst prop @obj">Bergen</word>
```

Figur 10 Eksempel på feil ordtagg.

5.2.2 Artikkelsamlinger

I dette prosjektet er det nødvendig å ha to samlinger med artikler til disposisjon. Den ene samlingen brukes under utviklingen av løsningsmetoden. Den kan på en måte ses på som et treningssett, da reglene blir utvidet etter hvert som det påvises nye feil. Den andre samlingen brukes i evaluering av systemet. Det er denne samlingen som gir en indikasjon på hvor gode resultater AIDaS kan oppnå.

Samlingene består av artikler som er hentet fra Bergens Tidende (BT). Artiklene er i hovedsak skrevet i årene 1996 til 1998. BT har et vidt spekter av nyheter, alt fra lokale til utenriks saker. I utgangspunktet er ikke løsningsmetoden domeneavhengig, men jeg har valgt å legge fokus på et geografisk område, Hordaland. Det vil si at samlingen i hovedsak består av lokale artikler.

Artiklene som samlingene består av er valgt ut tilfeldig, det vil si at så lenge artikkelen er fra det geografiske området Hordaland ble den tatt med i samlingen. Siden hver artikkel er merket med en geografi tagg var det ikke nødvendig for meg å lese gjennom artiklene for å se hvilket område den omhandlet.

Testsamlingen består av 50 artikler, eller omtrent 18 800 ord. Av disse ordene er cirka 1120 egennavn. Artiklene er av varierende lengde, alt fra 72 ord til 1267 ord. Evalueringssamlingen består også av 50 artikler som inneholder cirka 18 750 ord. Av disse ordene er cirka 1140 egennavn.

Samlingene inneholder en del feil som må fjernes før AIDaS kan identifisere stedsnavnene. Midt i artiklene er det en del mellomrom i tillegg til at tilfeldige bokstaver eller lignende ofte står sammen med første ord i en setning. Det er også en del skrivefeil.

5.2.3 Lister med egennavn

I kapitel 4.6 konkluderte jeg med å bruke lister med egennavn under klassifiseringen. Jeg har valgt å bruke følgende lister:

1. Liste med stedsnavn, stedsnavnliste
2. Liste med organisasjoner, organisasjonsliste
3. Liste med fornavn, personnavnliste

Stedsnavnlisten er den viktigste listen og innholder alle offisielle stedsnavn i Hordaland. Listen er ekstrahert fra sentralt stedsnavnregister hos Statens Kartverk [24]. Organisasjonslisten skulle i utgangspunktet inneholde alle organisasjoner i Hordaland, men det viste seg at det var tidkrevende å få tak i denne informasjonen. Dette førte til at jeg laget listen selv. Denne listen inneholder derfor kun noen få organisasjoner og er dermed langt fra fullstendig. Den siste listen, personnavnlisten, har jeg også laget selv. Da kun en av de tre listene er fullstendig, må man ta dette i betraktning under evalueringen av systemet.

Som vi kan se av Figur 6 blir listene brukt i modul *C Klassifisering*. Listene brukes sammen med reglene som et ekstra hjelpemiddel. Dette forklares nærmere i kapittel 6.

5.2.4 Semantiske sett

For å gjenkjenne stedsnavn har jeg valgt å bruke samme fremgangsmåte som blir brukt i ARNER. Det vil si at jeg har tatt utgangspunkt i noen av de semantiske settene fra ARNER.

De semantiske settene er hjelpemiddel for å klassifisere egennavn i rett kategorier og brukes i modul *C Klassifisering*. For eksempel vil settet i Figur 11 hjelpe til med å klassifisere personnavn og organisasjonsnavn. Det er kun personer eller organisasjoner som kan gjøre en ytring, derfor er det naturlig å gruppere slike verb i en gruppe, sier-verb.

```
1  tilføye tenke svare sukke spørre skrive
2  si rope mumle mene hyle hviske fortsette
3  fortelle anta advare understreke
```

Figur 11 Eksempel på semantiske sett - sier-verb.

Her følger en liste over hvilke semantiske sett som brukes for å skille stedsnavn fra de andre egennavnene:

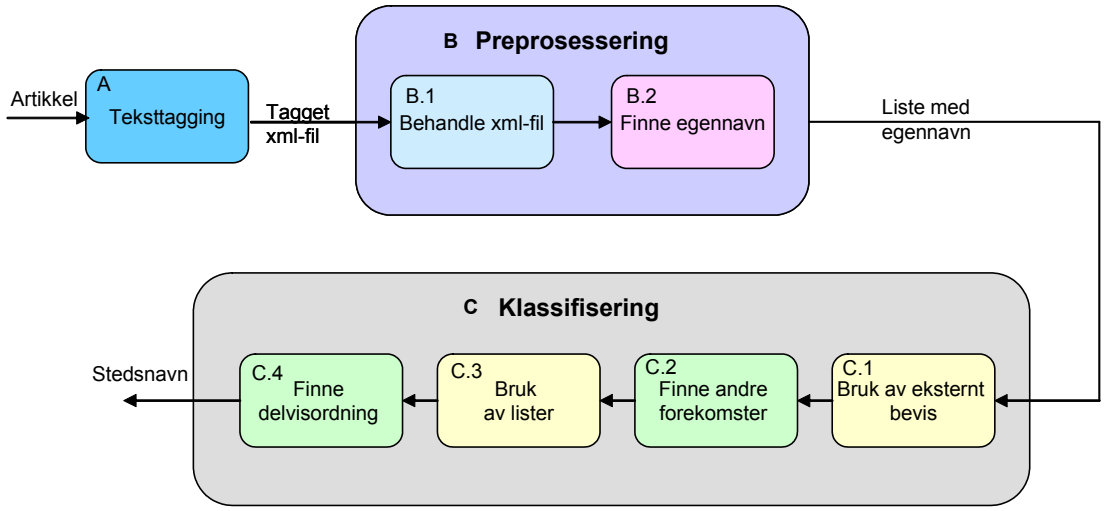
- Annetprefiks – typiske ord som er i begynnelsen av et egennavn som ikke er sted, for eksempel *TV* og *Høgskole*.
- Annetsuffiks – typiske ord som er i slutten av et egennavn som ikke er et sted, for eksempel *kontor* og *anlegg*.
- Person – ord som brukes om personer, for eksempel *tante* og *sønn*.
- Personyrke – forskjellige yrker, for eksempel *økonom* og *visesanger*.
- Sierverb – forklart ovenfor.
- StedSub – substantiv som står i forbindelse med sted, som for eksempel *hjemland* og *ås*.

5.3 Oppsummering

Dette kapittelet ga en overordnet beskrivelse av AIDaS med tilhørende ressurser. Oslo-Bergen taggeren er essensiell for at AIDaS skal fungere. Dette fordi ordklassetaggingen er et nødvendig hjelpemiddel for klassifisering av egennavn. Videre er de tre ressursene, artikkelsmalinger, lister med egennavn og de semantiske settene, viktige for at AIDaS skal gi best mulig resultater. Neste kapittel inneholder en detaljert beskrivelse av AIDaS. Det blir da lettere å se hvor viktig ressursene er for at AIDaS i sin helhet.

6 Detaljert beskrivelse av AIDaS

Dette kapittelet inneholder en detaljert beskrivelse av AIDaS. Figur 12 gir en oversikt over AIDaS. Modul *A Teksttagging* er den modulen som starter hele systemet og har kommunikasjonen med Oslo-Bergen taggeren, dette forklares i kapittel 6.1. Kapittel 6.2 og 6.3 beskriver modulene *B Preprosessering og C Klassifisering* med tilhørende faser. AIDaS bruker regler for å klassifisere egennavnene, kapittel 6.4 inneholder en beskrivelse av hvordan jeg gikk frem for å lage reglene samt en oversikt over de endelige reglene i AIDaS. Kapittel 6.5 inneholder et eksempel som illustrerer hva som skjer i hver modul i systemet.



Figur 12 Detaljert oversikt av AIDaS.

6.1 Modul A Teksttagging

Som vi kan av Figur 12 tar denne modulen inn artikkelteksten der AIDaS skal finne stednavnene. Før artikkelen kan sendes til Oslo-Bergen taggeren genererer denne fasen en xml-melding som inneholder artikkelteksten og noen konfigurasjons-parametre. Det er også denne modulen som oppretter SOAP-forbindelsen til Oslo-Bergen taggeren. Når Oslo-Bergen taggeren har tagget artikkelteksten mottar modulen en ny xml-melding som er av typen base64encoding. For å kunne bruke denne meldingen videre i systemet gjør modulen meldingen om til vanlig tekst. Til slutt gjøres noen små operasjoner som å sette inn taggene </set><set> i stedet for følgende streng:

<word id="w [0-9]+ " lemma="$." features="">.</word>

6.2 Modul B Preprosessering

Modulen inneholder to faser, *B.1 Behandle xml-fil* og B.*2 Finne egennavn*, slik som vist i Figur 12.

B.1 Behandle xml-fil tar inn xml-filen fra Oslo-Bergen taggeren. Fasen henter ut nyttig informasjon om hvert enkelt ord i teksten fra xml-filen og lagrer denne.

B.2 Finne egennavn er ganske lik modulen *Finne egennavn* i den gamle løsningsmetoden. En av forskjellene er imidlertid at denne fasen kun identifiserer ord som er skrevet med stor forbokstav. Den leter ikke etter sekvenser, da Oslo-Bergen taggeren har gjort dette i forkant. En annen forskjell er at fasen lager en liste med alle ord som starter en setning. For å kunne klassifisere egennavnene i neste modul, tar fasen vare på posisjon i setning og setningsnummer til alle egennavnene.

6.3 Modul C Klassifisering

Modulen for å klassifisere egennavne er delt inn i fire faser, henholdsvis *C.1 Bruk av eksternt bevis*, *C.2 Finne andre forekomster*, C.*3 Bruk av lister* og *C.4 Finne delvisordning*. I Figur 12 kan man se at C.*1* og *C.3* har samme farge, og at C.*2* og *C.4* har samme farge. Dette for å vise at de inneholder lignende funksjonaliteter. Rekkefølgen på fasene i denne modulen er svært viktig. Grunnen til dette er at man først må klassifisere med hensyn på kontekst før man baserer seg på listeoppslag. Det er kun på denne måten det er mulig å se om et egennavn brukes i en annen kontekst enn vanlig.

C.1 Bruk av eksternt bevis er relativt lik den første fasen i LTG systemet. For at egennavn skal klassifiseres må det komme klart frem av konteksten. Fasen tar ikke i bruk lister med egennavn. Reglene i denne fasen må se på konteksten rundt egennavnene, og klassifisere dem hvis det er mulig. Listen med første ord i setning blir ikke behandlet i denne fasen.

C.2 Finne andre forekomster tar inn egennavnene som ble klassifisert i forrige fase og søker etter andre forekomster av samme egennavn. Det vil si at man leter etter forekomster i både egennavnlisten og listen med første ord i setning. Forekomstene får tildelt samme type som det klassifiserte egennavnet. Man kan med stor grad av sikkerhet si at egennavnet er av samme type. Hvis det ikke hadde vært av samme type, hadde sannsynligvis forfatteren gitt informasjon om dette i konteksten, slik at den forrige fasen hadde fanget dette opp.

C.3 Bruk av lister er en fase som bruker regler for å klassifisere nye egennavn, men denne gangen kan reglene ta i bruk lister med egennavn. Det vil si at fasen ser på konteksten rundt egennavnene i tillegg til informasjon i lister. For eksempel vil det være mulig å klassifisere et egennavn som en person (*annet*) hvis det ser ut som et personnavn og at det første ordet i sekvensen finnes i en liste med fornavn.

C.4 Finne delvisordning er en utvidelse av den andre fasen. Den tar inn alle de egennavnene som er blitt klassifisert, både fra fase *C.1* og fase *C.3*. Hvis egennavnet er en sekvens, blir det delt inn i delvise ordninger. Oppdelingen er ikke helt lik den som blir gjort i LTG systemet. I denne fasen ser man også på sekvens, for eksempel vil egennavnet *Kong Oscars gate* få følgende delvise ordninger:

- Kong Oscars gate
- Kong Oscars
- Oscars gate
- Kong
- Oscars
- gate

Deretter leter man etter andre forekomster av ordningene og egennavn som ikke er sekvenser. Helt til slutt sjekkes det om ordene som er første ord i setning og ord som er skrevet med bare store bokstaver er et stednavn ved å gjøre oppslag i stedsnavn-listen. Når alle egennavn som kan klassifiseres har fått tildelt en type, enten *sted* eller *annet*, telles antall duplikater. Det vil si at den endelige listen med stedsnavn inneholder stedsnavn i teksten med antall forekomster.

6.4 Reglene i AIDaS

Det er vanskelig å finne litteratur på hvordan man bør gå frem for å lage regler for å identifisere stedsnavn i norske tekster. De fleste artiklene som tar opp liknende problemstillinger har en tendens til å vise trivielle eksempler på regler, uten å forklare hvordan de har gått frem. Når Jónsdóttir [5] utviklet regler for ARNER, begynte hun med noen enkle og generelle regler, for deretter å videreutvikle dem ved hjelp av en testsamling. Som nevnt tidligere ble noen av reglene i ARNER svært detaljerte og spesifikke. Dette er noe jeg har prøvd å unngå i AIDaS. I kapitlene 6.4.1 og 6.4.2 beskriver jeg hvordan jeg gikk frem for å lage reglene. Det er først i kapittel 6.4.3 de endelige reglene i AIDaS forklares.

6.4.1 De første reglene

For å lage de første reglene tok jeg utgangspunkt i en tagget samling fra Universitetet i Oslo. Ved å se på alle ord i forbindelse med stedsnavn, personnavn og organisasjonsnavn var det mulig å lage noen generelle mønster. Eksempel på mønster jeg kom frem til er som følger:

1. <prep> <sted>
   Det er ofte en preposisjon foran et stedsnavn.
2. <verb> <person/organisasjon>
   Det er sjelden et verb rett foran et sted, med unntak av verbet *forlate*.
3. <tittel> <person>
   Hvis en tittel kommer før et egennavn er det en person.
4. <sier-verb> <prep> <person/organisasjon>
   Hvis et sier-verb står foran en preposisjon er ikke egennavnet et sted.
5. <egennavn><,> <egennavn> … <konj> <egennavn>
   Hvis det er en oppramsing av egennavn vil alle egennavnene være av samme type som den første.

Som vi kan se er mønstrene svært generelle, noe som også var hensikten. Vi kan se at det er lettere å skille stedsnavn fra personnavn og organisasjonsnavn enn å lage regler for å skille personnavn fra organisasjonsnavn. Da denne oppgaven er rettet mot å identifisere stedsnavn, valgte jeg derfor å slå sammen organisasjonsnavn og personnavn til en felles kategori, *annet*. Det vil si at AIDaS gir ut to lister, en med stedsnavn og en som inneholder organisasjoner og personer samlet. I de tilfellene hvor alle egennavnene ikke blir klassifiserte, gis det også ut en liste som inneholder disse navnene.

6.4.2 Trening av AIDaS

Etter at jeg kom frem til de første reglene begynte jeg å trene systemet. Det vil si at jeg brukte testsamlingen min og kjørte et og et dokument gjennom systemet. Hvis det oppstod noen feilklassifiseringer prøvde jeg å utvide de allerede etablerte reglene,

eller å lage en ny regel. Det var også ved hjelp av denne testsamlingen jeg utvidet de semantiske settene og la til organisasjonsnavn og personnavn i listene.

I noen tilfeller var det vanskelig å lage en regel for å håndtere enkelte situasjoner, da tenker jeg spesielt på regelen <*prep*> <*sted*>. Dette er en regel som er svært generell. Regelen ble hele tiden utvidet, men i noen tilfeller var det vanskelig å lage en unntaksregel, slik som i følgende eksempel:

> ***Hos Konsum*** *i grensebyen Strømstad har butikksjef Kay Evertsson regnet ut at hver fjerde kunde er norsk.*

Problemet i denne setningen er *Hos Konsum*. Det er ikke naturlig å legge *Konsum* inn i listen med organisasjoner, da dette er en svensk organisasjon og ikke en fra Hordaland. *Konsum* blir heller ikke brukt andre steder i teksten slik at andre forekomster av egennavnet kan hjelpe til med å bestemme type. Dette fører altså til at AIDaS klassifiserer egennavnet som et sted. Jeg valgte imidlertid å bruke regelen i tredje fase som en av de siste reglene. Grunnen til dette er at regelen for det meste gir gode resultater når man skal identifisere stedsnavn som ikke er tilgjengelig i stedsnavnlisten, for eksempel i dette tilfellet:

> *Derfor må alle penger som samles inn* ***i Norge*** *til gatebarna, kanaliseres via vår stiftelse Europa i Fokus under kontrollerte forhold, mener misjonær Franz Johansen.*

I utgangspunktet er scenarioet relativt likt det ovenfor, men i dette tilfellet er egennavnet et stedsnavn. *Norge* er ikke et sted i Hordaland, derfor er det ikke med i listen med stedsnavn. Selv om artiklene i hovedsak er lokale artikler er det ofte slik at det brukes stedsnavn som ikke er i stedsnavnlisten. For eksempel hvis man sammenlikner Bergen med andre store byer som Oslo, Trondheim eller Stavanger. Jeg mener derfor at det er viktig å ha med denne regelen selv om den i noen tilfeller vil gi støy som vist i eksempelet over.

En annen ting man må være klar over når man bruker en prosedyre som denne for å lage regler er at systemet ikke kan være bedre enn de situasjonene som oppstår i testsamlingen. Jeg burde derfor hatt en større samling hvis systemet skulle vært mer robust, men dette var ikke mulig å gjennomføre innen tidsrammen til prosjektet. Dette fører derfor til at AIDaS ikke kan ta høyde for alle mulige situasjoner som kan oppstå.

6.4.3 De endelige reglene

Etter at jeg var ferdig med å trene AIDaS stod jeg igjen med to sett av regler, ett sett for fase *C.1Bruk av eksternt bevis* og ett for fase *C.3Bruk av lister*. Tabell 9 viser en oversikt over hvilke regler AIDaS bruker i de to fasene, som vi kan se går noen av reglene igjen i begge fasene. Jeg skal nå gi en presentasjon av de forskjellige reglene. I vedlegg A er det lagt ved kodeutklipp av reglene.

Tabell 9 Oversikt over regler i AIDaS.

Regler i C.1	**Regler i C.3**
R 1.0 StorBokstavRegel	*R 3.0* PersRegel
R 1.1 TittelRegel	*R 3.1* YrkeRegel
R 1.2 SierRegel	*R 3.2* SierRegel
R 1.3 StedSubRegel	*R 3.3* PrepRegelListe
R 1.4 SubRegel	*R 3.4* KommaRegel2
R 1.5 ForkRegel	*R 3.5* KonjRegel2
R 1.6 SammenMedRegel	*R 3.6* PrepRegel
R 1.7 ApposisjonsRegel	*R 3.7* ListeRegel
R 1.8 KommaRegel	
R 1.9 KonjRegel	
R 1.10 VerbRegel	

R 1.0 StorBokstavRegel er en regel som tar høyde for når forfattere skriver hele ord med store bokstaver. Dette brukes ofte i begynnelsen av et avsnitt, som i følgende eksempel:

> *SLUTT PÅ HJELPEN. Hvis ikke bergenserne er fornøyd etter brøyteinnspurten i ettermiddag, får de selv gå ut og måke, sier kommunen*.

Ingen av ordene i *SLUTT PÅ HJELPEN* er egennavn, men *HJELPEN* ville blitt klassifisert som et stedsnavn, på grunn av <*prep*> <*sted*> regelen, hvis man ikke håndterer disse ordene på en spesiell måte. Regelen fjerner derfor ord der hele ordet er skrevet med store bokstaver fra egennavnlisten og legger dem inn i en ny liste. Den nye listen blir behandlet i *C.4 Finne delvisordning*.

R 1.1 TittelRegel er en regel som klassifiserer et egennavn som *annet* hvis ordet foran er en tittel eller hvis de to foregående ordene er <*tittel*> <*prep*>, slik som i disse tilfellene:

Statsminister Kjell Magne Bondevik
Leder av NSB

R 1.2 SierRegel er en regel som sjekker om det brukes sier-verb i nærheten av egennavn. I slike tilfeller blir egennavnet klassifisert som *annet*. Denne regelen tar høyde for fire forskjellige scenario som vist nedenfor. Det er det egennavnet som er uthevet som skal klassifiseres:

1. sier *Hans*
2. sier til *NTB*
3. sier Hans til *NTB*
4. sier overingeniør Hosleif i *Vegdirektoratet*

Vi kan si at både det egennavnet som gjør en ytring og som mottar en ytring er av typen *annet* på grunn av at et sted ikke kan gjøre noen av delene. I det siste tilfellet er det også tatt høyde for at en person som gjør en ytring kan komme fra en organisasjon, siden dette ofte er tilfelle i nyhetstekster.

R 1.3 StedSubRegel og **R 1.4 SubRegel** er regler som er relativt like. *StedSubRegel* bruker en liste med substantiv som ofte brukes sammen med stedsnavn. Hvis et slikt substantiv står foran et egennavn, blir det klassifisert som et stedsnavn, <*stedsub*><*sted*>. Et egennavn vil også klassifiseres som et *sted* hvis det er en preposisjon mellom stedssubstantivet og egennavnet, <*stedsub*><*prep*><*sted*>.

I listen med stedssubstantiv er det også noen substantiv som ofte er suffiks i et sammensatt stedssubstantiv, for eksempel er *by* suffiks i *havneby* og *grenseby*. I stedet for å skrive inn alle mulige varianter hvor *by* er suffiks, er typiske stedssuffiks i listen markert med en *, slik som dette, **by*. Eksempel på tilfeller hvor *StedSubRegel* blir brukt er som følger:

*Hun kom fra **havnebyen Bergen**.*
*SOS Barnebyer har to **barnebyer i Europa**.*

SubRegel klassifiserer alle egennavn med et substantiv foran seg som *annet*. Det er viktig at *StedSubRegel* kommer før *SubRegel*, slik at substantiv som indikerer et sted ikke blir klassifisert som annet.

R 1.5 ForkRegel er en regel som klassifiserer forkortelser til egennavn. For at denne regelen skal slå til må forkortelsen komme direkte etter egennavnet og stå inne i parenteser, for eksempel *Bergens Tidende (BT)*. Det er viktig å merke seg at det fulle navnet, i dette tilfellet *Bergens Tindende* først må klassifiseres som *annet* før forkortelsen kan klassifiseres.

R 1.6 SammenMedRegel er en svært spesiell regel. Ordet *med* er en preposisjon, og hvis denne regelen ikke var tilgjengelig ville egennavnet i følgende setning blitt klassifisert som et stedsnavn.

> *Terje Sørensen ble rikskjendis sommeren 1992 da **han sammen med Frode Skogvold** ble ekskludert fra Fremskrittspartiet.*

SammenMedRegelen klassifiserer egennavn som har <*pronomen*> <*sammen med*> foran seg som annet, slik at vi kan unngå problemet ovenfor.

R 1.7 ApposisjonsRegel er en regel som klassifiserer apposisjoner. En apposisjon er et forklarende tillegg til et substantivisk ledd med samme innhold som det leddet det står til. Et eksempel på et tilfelle hvor denne regelen vil slå til er:

***Ingrid, søsteren** til Per, var svært dyktig.*

Regelen bruker en liste med typiske substantiv som refererer til en person, slik som *søster*, *bror*, *onkel* og lignende. Hvis det står et egennavn som er etterfølgt av et komma og et ord i den listen er det stor sannsynlighet for at egennavnet er en person, og dermed blir det klassifisert som *annet*.

R 1.8 KommaRegel og **R 1.9 KonjRegel** er regler som tar høyde for oppramsing av egennavn. Det er viktig å merke seg at det første ordet i oppramsingen må klassifiseres før neste ord kan klassifiseres. Her er et eksempel på en slik oppramsining:

*Alle de store byene **Oslo, Bergen, Trondheim og Stavanger** hadde fint vær 17. mai.*

Oslo klassifiseres først av *R 1.3*, *Bergen* blir så klassifisert som et sted på grunn av *KommaRegelen*, <*sted*> <*,*> <*sted*>. På samme måte blir Trondheim klassifisert som et sted. Til slutt blir Stavanger klassifisert av *KonjRegelen*, <*sted*> <*konj*> <*sted*>. Disse reglene tar inn et ukjent egennavn og ser på hvilken type egennavnet foran i oppramsingen ble klassifisert som og gir det den samme typen.

R 1.10 VerbRegel er en regel som klassifiserer alle egennavn som har et verb foran seg som *annet*, med unntak av verb som for eksempel *forlate*, som den ikke gjør noe med. Det er sjelden et verb står foran et stedsnavn, og i de tilfellene det skjer omtaler man ofte stedet som en organisasjon.

R 3.0 PersRegel er en regel som undersøker om et egennavn kan være et personnavn. Regelen ser på strukturen til navnet, hvis det inneholder mer enn to ord og første delen av navnet finnes i en liste over fornavn, blir hele egennavnet klassifisert som *annet*.

R 3.1 YrkeRegel er en regel som leter etter sekvensen <*yrke*> <*prep*> etterfølgt av et egennavn, for eksempel *markedssjefen i NSB*. Siden *markedssjef* er et yrke som er i en liste over forkjellige yrker blir *NSB* klassifisert som *annet*, og ikke et *sted*.

R 3.2 SierRegel er den samme regelen som *R 1.2*.

R 3.3 PrepRegelListe og **R 3.6 PrepRegel** er to regler som tar hånd om preposisjoner som kommer rett før et egennavn. Som nevnt tidligere gir *PrepRegelen* en god indikasjon på at et egennavn er et stedsnavn, men det kan dessverre komme en del støy. For å fjerne en del feilklassifiseringer har jeg laget en annen variant av regelen, *PrepRegelListe*. Denne regelen bruker lister med organisasjoner og stedsnavn og flere forskjellige semantiske sett. *PrepRegelListe* klassifiserer ikke et egennavn hvis det ikke finnes noe bevis på at det er av en bestemt type i en av listene eller settene.

R 3.4 KommaRegel2 og **R 3.5 KonjRegel2** er svært like regel *R 1.8* og *R 1.9*, eneste forskjell på reglene er at disse reglene har listeoppslag som et hjelpemiddel under klassifiseringen.

R 3.6 PrepRegel forklart sammen med *R 3.3*.

R 3.7 ListeRegel er den siste regelen i systemet. Regelen gjør oppslag i stedslister, organisasjonslister og lignende for å klassifisere eventuelle egennavn som ikke ble klassifisert av noen av de andre reglene. Det er viktig å merke seg at denne regelen ikke blir kjørt før alle egennavnene har vært gjennom alle reglene.

6.5 Eksempel – AIDaS

Dette kapittelet inneholder en demonstrasjon av AIDaS. Med utgangspunkt i en artikkel fra Bergens Tidende skal jeg vise hvordan systemet fungerer i praksis. Artikkelen som skal klassifiseres er som følger:

MC-fører skadd på Askøy
En 38 år gammel motorsyklist er fraktet til sykehus i ambulanse etter at han i morges ble funnet liggende i veibanen på Erdal. Mannen hadde skader i brystet da han ble funnet, men var ved bevissthet. Hvor alvorlig skadd han er, er fortsatt uklart.
Motorsyklisten behandles nå på Haukeland Universitetssykehus. Ifølge en pressemelding fra sykehuset er mannen lettere skadet, og tilstanden er stabil.
Ifølge politiet kom han kjørende på Askvegen da han kolliderte med en varebil i krysset ved det gamle postkontoret på Erdal. Bilen kom fra Grensedalen.
- Dette er et veldig kinkig kryss med dårlig sikt. Her bør noe gjøres, for det er fort gjort å krasje her, sier politibetjent Finn Martin Systad ved Askøy lensmannskontor til BT.
Politiet fikk melding om ulykken klokken 08.16. Ulykkesstedet ligger på riksvei 563, ved avkjørselen til gamleveien på Askøy.

Alle stedsnavn i artikkelen er markert med gul farge, mens organisasjoner og personer, som er av typen *annet* er markert med fiolett. Klassifiseringen ovenfor er ikke basert på resultatet fra Oslo-Bergen taggeren, men en manuell gjennomlesning. Jeg skal nå demonstrere hva som skjer i AIDaS steg for steg.

6.5.1 Modul A Teksttagging

Denne modulen tar inn artikkelteksten som er vist over, artikkelteksten er en ren tekstfil. Deretter lager modulen en xml-melding som inneholder artikkelteksten og konfigurasjonsparametrene. xml-meldingen ligger vedlagt i vedlegg B Figur 38. For å sende xml-meldingen til Oslo-Bergen taggeren oppretter modulen en SOAP-forbindelse. Det er via denne forbindelsen all kommunikasjon mellom AIDaS og Oslo-Bergen taggeren foregår. Når meldingen er sendt venter modulen på å få svar. Svaret modulen får fra Oslo-Bergen taggeren er vedlagt i vedlegg B Figur 39. Siden meldingen fra Oslo-Bergen taggeren er av typen base64encoding, gjør modulen derfor base64encoding om til vanlig tekst. Når det er gjort søker modulen i teksten etter følgende streng som den bytter ut med *</set> <set>*:

<word id="w [0-9]+ " lemma="$." features="">.</word>

Resultatet etter at den mottatte meldningen er konvertert til vanlig tekst og taggene *</set>* og *<set>* er satt inn er vist i vedlegg C.

6.5.2 Modul B Preprosessering

Når artikkelteksten er ferdig tagget er det klart for å preprosessere xml-filen.

B.1 Behandle xml-fil tar inn xml-filen slik den er vist i vedlegg C, og leser filen. Alle ordene i en setning er innenfor taggene *<set>* og *</set>*. Fasen henter ut ordene som er i hver enkelt av disse taggene og tar vare på informasjonen som er i lemma og feature variablene. Når alle setningene er behandlet legges de inn i en liste som sendes videre til neste fase.

B.2 Finne egennavn får inn en liste som inneholder alle setningene. Det er nå klart for å finne egennavn og første ord i setning. Hver enkelt setning blir behandlet etter

tur. Det første som skjer er at første ord i setning blir lagt til en liste som kun inneholder første ord i hver setning. Deretter undersøker fasen de resterende ordene i setningen, og ser om de er skrevet med stor forbokstav. Hvis dette er tilfellet, blir ordet lagt til en liste som inneholder egennavn. Hvert egennavn har en lenke til posisjonen i setningen og setningsnummer. Det er ikke nødvendig å ha slike lenker i listen med første ord i setning siden posisjonen er kjent og at posisjonen i tabellen tilsvarer setningsnummer. Etter at alle setningene er undersøkt har AIDaS generert en liste med første ord i setning og en med alle egennavn i teksten, slik som vist i henholdsvis Tabell 10 og Tabell 11. Når disse listene er generert er preprosessseringsmodulen ferdig, og det er klart for å begynne å klassifisere egennavnene. Det er viktig å merke seg at både setningsnummer og posisjonsnummer starter på 0.

Tabell 10 Liste med første ord i setning.

Første ord i setning
MC-fører
En
Mannen
Hvor
Motorsyklisten
Ifølge
Ifølge
Bilen
Dette
Her
Politiet
Ulykkesstedet

Tabell 11 Liste med egennavn.

Egennavn	**Posisjon**	**Setning**
Askøy	3	0
Erdal	20	1
Haukeland Universitetssykehus	4	4
Askvegen	6	6
Erdal	20	6
Grensedalen	3	7
Finn Martin Systad	16	9
Askøy lensmannskontor	18	9
BT	20	9
Askøy	11	11

6.5.3 Modul C Klassifisering

Denne modulen tar inn de to listene som ble generert i forrige modul og prøver å tilegne dem en type.

C.1 Bruk av eksternt bevis tar inn hvert enkelt egennavn i Tabell 11 og sjekker disse opp mot reglene. I dette tilfellet er det kun to egennavn som blir klassifisert, nemmelig *Finn Martin Systad* og *Askøy lensmannskontor*. *Finn Martin Systad* blir klassifisert som *annet* av *R 1.1 TittelRegel* på grunn av at *politibetjent* har fått variabelen *tittel* fra Oslo-Bergen taggeren.

Askøy lensmannskontor klassifiseres også som *annet*, på grunn av *R 1.2 SierRegel*. Når et egennavn blir klassifisert, fjernes det fra egennavnlisten og legges inn i en liste for den typen det er klassifisert som. Tabell 12 viser en foreløpig liste over klassifiseringen. Siden det ikke er funnet noen stedsnavn i denne fasen, er denne listen tom.

Tabell 12 Klassifiseringsresultatet etter *C.1*.

Stedsnavn	**Annet**
	Finn Martin Systad
	Askøy lensmannskonto

C.2 Finne andre forekomster leter etter andre forekomster av *Finn Martin Systad* og *Askøy lensmannskontor*, men det finnes ikke noen andre forekomster av dem verken i Tabell 10 eller Tabell 11. Dette medfører at det fremdeles er åtte egennavn som ikke er klassifisert etter de to første fasene.

C.3 Bruk av lister tar inn de resterende egennavnene som er igjen i egennavnlisten. Også her blir hvert egennavn sjekket opp mot regler, men i denne fasen er reglene litt annerledes. Grunnen til dette er at de kan bruke lister med stedsnavn, organisasjonsnavn og personnavn. Slike lister kan være svært nyttige, noe som vi kan se i eksempelet.

De resterende egennavnene blir klassifiser av regler som ser på preposisjoner foran ordet i tillegg til oppslag i lister, det vil si *R 3.3 PrepRegelListe*. Alle stedsnavnene i artikkelen finnes i stedsnavnlisten, det er derfor enkelt å klassifisere dem som *sted*.

Haukland Universitetssykehus er ikke med i en liste over organisasjoner, men siden egennavnet inneholder *sykehus* som et suffiks, blir det klassifisert som *annet*. *BT* blir også klassifisert som annet, siden *BT* er en organisasjon i organisasjonslisten.

Etter at denne fasen er ferdig er alle egennavnene i egennavnlisten klassifisert, se Tabell 13 for en oversikt.

Tabell 13 Klassifiseringsresultatet etter *C.3*.

Stedsnavn	**Annet**
Askøy	Finn Martin Systad
Erdal	Askøy lensmannskontor
Askvegen	Haukland Universitetssykehus
Erdal	BT
Grensedalen	
Askøy	

C.4 Finn delvisordning sjekker om det finnes noen treff på delvisordning, selv om alle navnene i egennavnlisten er klassifisert. Det er for eksempel nødvendig å sjekke om det finnes noen treff i listen med første ord i setning. Fasen sjekker også om noen av ordene i Tabell 10 finnes i listen over stedsnavn.

Til slutt gjenstår det å telle antall duplikater av egennavn i de to listene, se Tabell 14 for det endelige resultatet.

Tabell 14 Endelig klassifiseringsresultat og utdata til AIDaS.

Stedsnavn med forekomster		**Annet med forekomster**	
Askøy	2	Finn Martin Systad	1
Erdal	2	Askøy lensmannskontor	1
Askvegen	1	Haukland Universitetssykehus	1
Gensedalen	1	BT	1

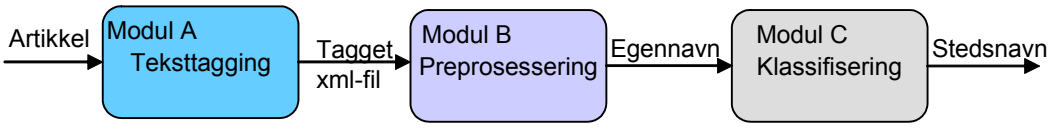
6.6 Oppsummering

I dette kapittelet har jeg forklart hvordan AIDaS fungerer. Hvis vi sammenligner AIDaS med den abstrakte løsningsmetoden fra høstprosjektet, ser vi at de er forskjellige men allikevel er det en del likheter. Den største forskjellen er at løsningsmetoden kun var en abstrakt løsning som ved implementering viste seg å inneholde en del svakheter, mens AIDaS er en løsning som er implementert og som fungerer. Jeg vil presisere at selv om AIDaS kun inneholder regler for i hovedsak å identifisere stedsnavn er det mulig å legge til eller endre reglene slik at AIDaS kan brukes i andre sammenhenger.

I kapittel 3.2 ble det nevnt at de største svakhetene til løsningsmetoden fra høstprosjektet var håndtering av første ord i setning og egennavnsekvenser. Disse problemene er ikke like dominerende i AIDaS, først og fremt på grunn av Oslo-Bergen taggeren.

7 Teknisk beskrivelse

Dette kapittelet inneholder en teknisk beskrivelse av AIDaS. Jeg har valgt å ta utgangspunkt i den overordnede oversikten i Figur 13 og klassediagram for å forklare hvordan AIDaS er implementert. (Figur 13 er en gjengivelse av Figur 5.)



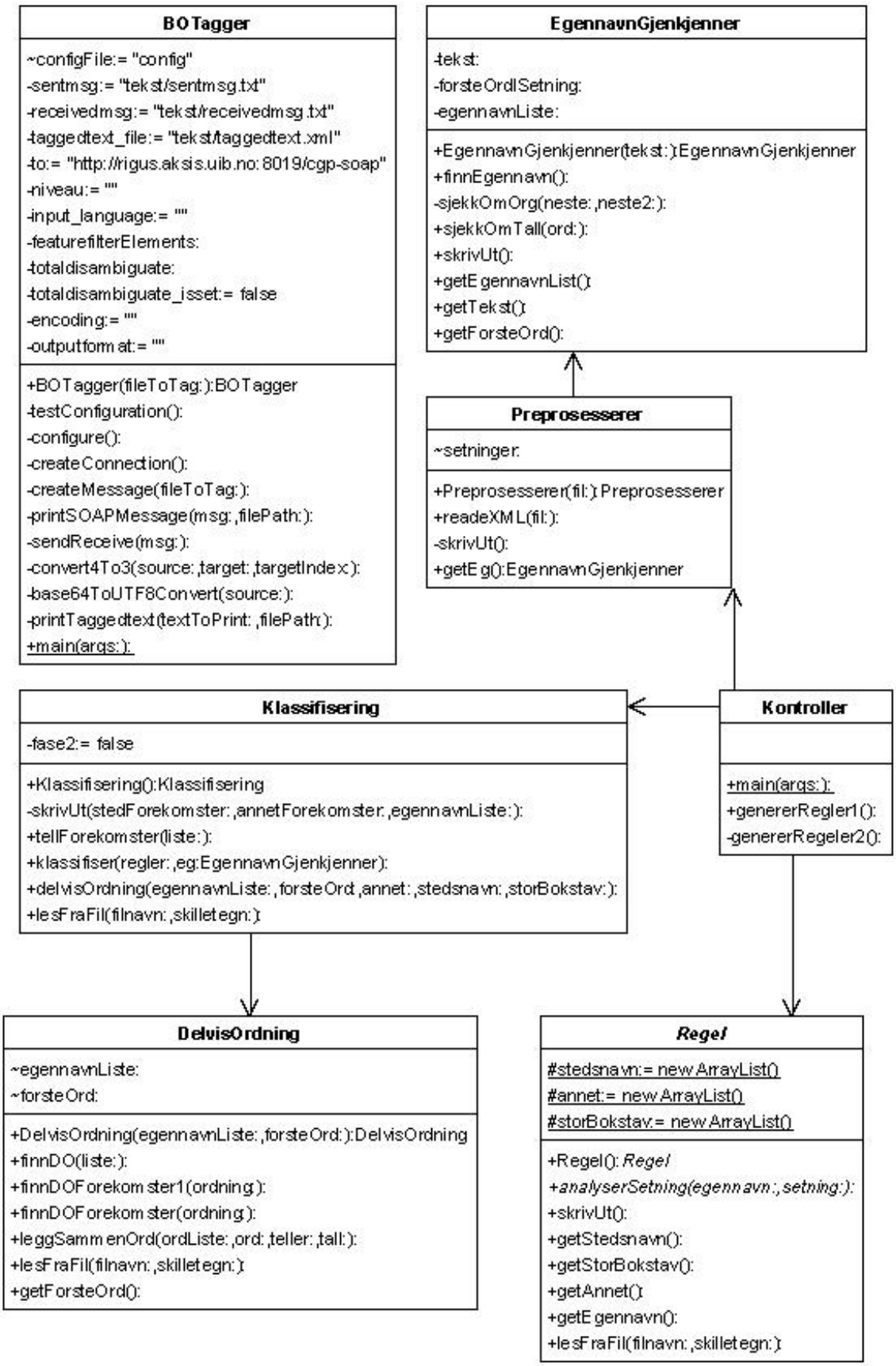
Figur 13 Overordnet oversikt over AIDaS.

7.1 Implementasjon

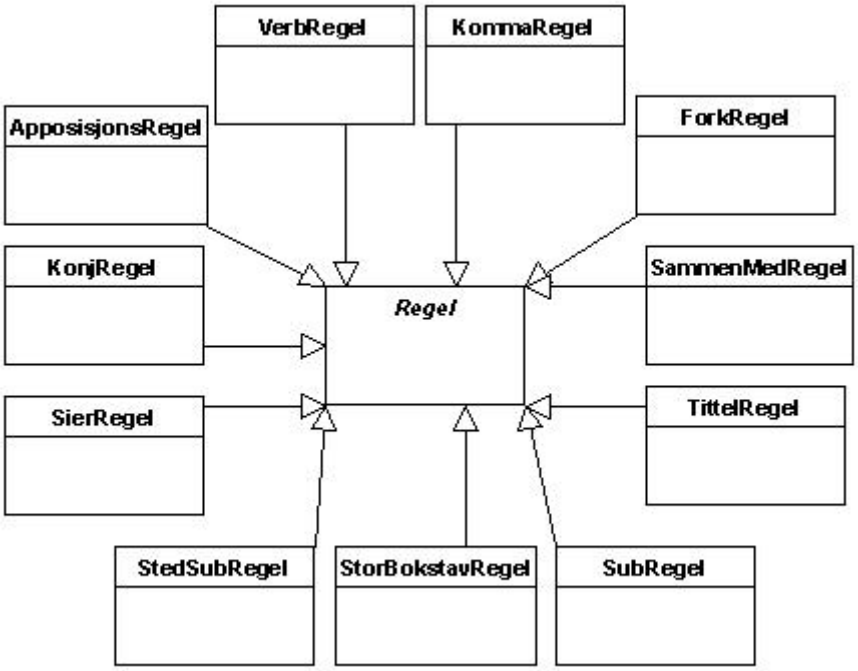
AIDaS er implementert i Java j2re1.4.2_06, i tillegg er følgende jar-filer importert: dom.jar, j2ee.jar, jdom.jar, mail.jar, saaj-api.jar, saaj-impl.jar og xercesImpl.jar.

Figur 13 til Figur 17 viser en oversikt over klassene i AIDaS. I utgangspunktet hører alle klassene sammen i et og samme diagram, men for å bedre lesbarheten har jeg valgt å dele dem opp. Figur 14 inneholder de viktigste klassene i AIDaS, med unntak av subklassene til *Regel*. Subklassene til *Regel* er i vist Figur 15 og Figur 16, hvor klassene i Figur 15 er reglene som brukes i fase *C.1 Bruk av eksternt bevis*, mens reglene i Figur 16 brukes i fase *C.3 Bruk av lister*. Det er viktig å merke seg at subklassen *SierRegel* er med i både Figur 15 og Figur 16, men klassen er kun implementert en gang. AIDaS inneholder også noen hjelpeklasser som flere av hovedklassene bruker. Disse klassene er vist i Figur 17. Jeg har selv implementert alle klassene, men klassen *BOTagger* har jeg implementert sammen med hjelpeveileder Øyvind Vestavik. En oversikt over hvordan klassene er implementert finnes i vedlegg D.

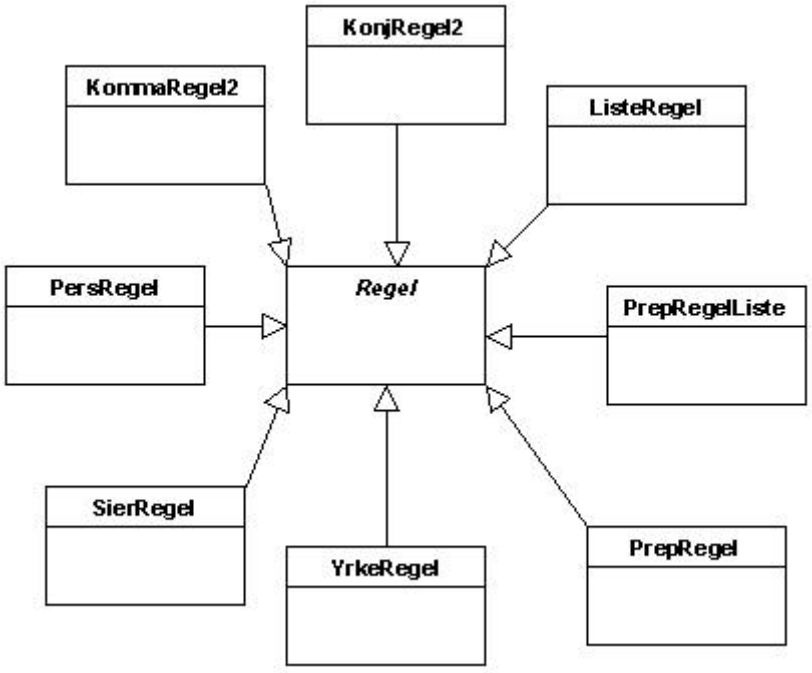
I de følgende kapitlene skal jeg forklare hvordan AIDaS fungerer teknisk sett. Dette gjøres ved å se på hver enkelt modul og å koble dem sammen med klassene.



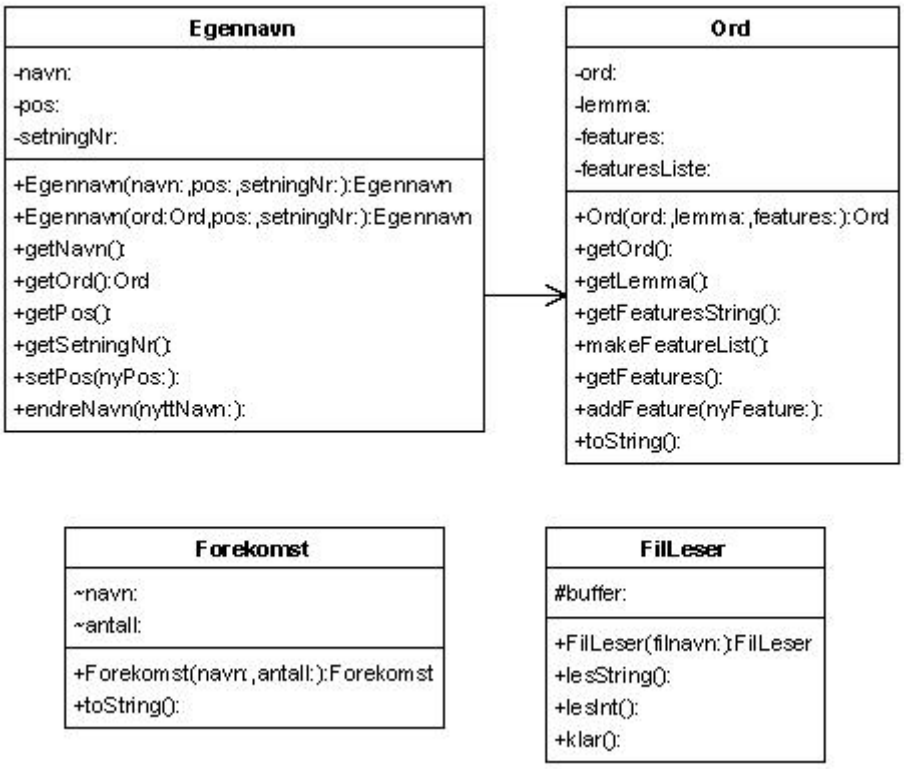
Figur 14 Klassediagram uten subklassene til Regel og hjelpeklasser.



Figur 15 Klassediagram over reglene i *C.1*.



Figur 16 Klassediagram over reglene i *C.2*.



Figur 17 Hjelpeklasser.

7.1.1 Modul A Teksttagging

All funksjonalitet i denne modulen ligger i klassen *BOTagger*.

BOTagger tar inn artikkelen som er inndataen til systemet. Før det er mulig å finne egennavnene i teksten og å klassifisere dem er det nødvendig å tagge den. Dette gjøres ved å sende den til Oslo-Bergen taggeren. Siden Oslo-Bergen taggeren er en ekstern ressurs sendes informasjonen via Internet. For å opprette en forbindelse med Oslo-Bergen taggeren bruker AIDaS en webservice basert på SOAP. SOAP er et grensesnitt som gjør det mulig for to applikasjoner som er knyttet til Internet å kommunisere med hverandre ved hjelp av xml-meldinger. [25]

xml-meldingen som AIDaS sender til Oslo-Bergen taggeren består av to deler, *SOAPEnvelope* (konvolutt) og *SOAPBody* (kropp). I konvolutten blir navnerommene som skal brukes lagt til. Kroppen inneholder flere element, deriblant hvor meldingen skal sendes, teksten som skal tagges og forskjellige konfigurasjonsparametre. Vedlegg B Figur 38 viser et eksempel på hvordan en slik melding ser ut.

Når meldingen er sendt venter *BOTaggeren* på å få svar. Svaret fra Oslo-Bergen taggeren inneholder, som den sendte meldingen, en konvolutt og en kropp. Kroppen inneholder kun et element, som er den taggede teksten i base64encoding. Vedlegg B Figur 39 viser hvordan en mottatt melding kan se ut. Siden den taggede teksten er i base64encoding må den gjøres om til vanlig tekst, utf8. Helt til slutt når teksten er gjort om til vanlig tekst søker *BOTagger* gjennom meldingen og erstatter alle

<word id="w [0-9]+ " lemma="$." features="">.</word>

med *</set> <set>* ved hjelp av regulære uttrykk. Dette gjøres for at det skal være lettere å finne hvor en setning begynner og slutter når xml-filen skal leses i neste modul.

7.1.2 Modul B Preprosessering

Denne modulen inneholder to klasser som utgjør funksjonaliteten til fasene *B.1 Behandle xml-fil* og *B.2 Finne egennavn*. De to klassene er henholdsvis *Preprosesserer* og *EgennavnGjenkjenner*. Det er klassen *Kontroller* som starter modulen. I tillegg brukes hjelpeklassene *Ord* og *Egennavn*.

Kontroller starter denne modulen ved å opprette en instans av klassen *Preprosesserer*.

Preprosesserer leser xml-filen som ble endret i *BOTagger*. xml-filen er bygget opp som en trestruktur hvor *<set>*-elementet ligger rett under rot-elementet. Det er derfor lett å hente ut relevant informasjon ved hjelp av klasser og metoder i pakken *jdom.jar*. På denne måten kan *Preprosesserer* hente ut informasjon om hvert enkelt ord i teksten. Denne informasjonen lagres i et *Ord-objekt*. Alle ord som er i en setning, legges til en liste som representerer setningen. Disse listene blir igjen lagt til en ny liste som inneholder alle setningene i hele teksten. Klassen inneholder med andre ord funksjonaliteten som trengs for å utføre fase *B.1 Behandle xml-fil*.

EgennavnGjenkjenner får tilsendt listen som inneholder alle setningene i teksten fra *Preprosesserer*. Ved å gå systematisk gjennom listene finner den alle ord som starter en setning og alle ord som er skrevet med stor forbokstav. Ord som er skrevet med stor forbokstav blir sett på som egennavnkandidater, det vil si at klassen oppretter et *Egennavn-objekt* for hver kandidat. Når klassen er ferdig har vi en liste med egennavn og en liste med første ord i en setning. Denne klassen tilsvarer fase *B.2 Finne egennavn*.

Ord er en hjelpeklasse som lagrer nyttig informasjon om et ord i et objekt. Informasjonen som lagres er:

- Selve ordet
- Lemma til ordet
- Liste med features

Denne informasjonen brukes under klassifisering av egennavnene.

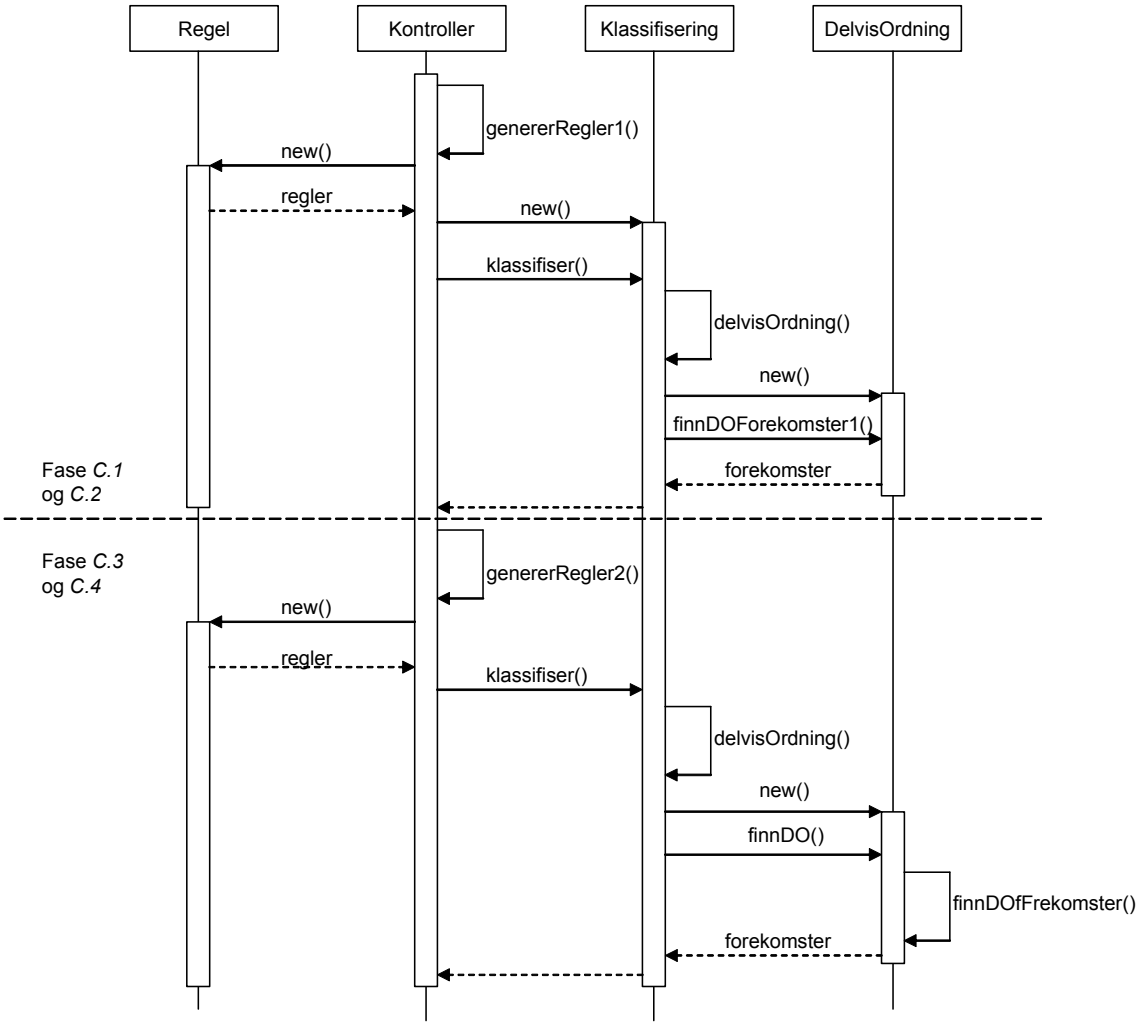
Egennavn er også en hjelpeklasse som lagrer informasjon som brukes under klassifisering. Alle egennavn som *EgennavnGjenkjenner* finner blir lagret som et *Egennavn-objekt*. Et *Egennavn-objekt* består av følgende:

- *Ord-objekt*
- Setningsnummer
- Posisjon i setning

7.1.3 Modul C Klassifisering

Funksjonaliteten for å utføre klassifiseringen finnes i klassene *Klassifisering*, *DelvisOrdning* og *Regel* med tilhørende subklasser. Klassen som starter selve

klassifiseringen er *Kontroller*. I tillegg brukes klassene *FilLeser* og *Forekomst*. For at det skal være lettere å forstå hva som skjer i denne modulen har jeg laget et sekvensdiagram, se Figur 18. Jeg har ikke tatt med argumentene til metodene i figuren eller i teksten.



Figur 18 Sekvensdiagram for modul *C*.

Kontroller starter også denne modulen, men før den kaller metoder for å utføre klassifiseringen oppretter den reglene som skal brukes i fasene. Som vi kan se av Figur 18 kaller den metoden *genererRegler1()* for å lage reglene til fase *C.1 Bruk av eksternt bevis* og *genererRegler2()* for å lage reglene i fase *C.3 Bruk av lister*. Når reglene er generert kaller den metoden *klassifiser()*.

Klassifisering inneholder metoder for å utføre deler av klassifiseringen. Når *Kontroller* kaller metoden *klassifiser()* begynner første fase *C.1 Bruk av eksternt bevis*. Hvert egennavn sjekkes opp mot reglene som tilhører denne fasen, det vil si reglene i Figur 15. Hvis en regel slår til blir egennavnet tatt ut av egennavnlisten og lagt til en liste for den typen det er klassifisert som, det vil si enten *sted* eller *annet*. I de tilfellene det ikke er mulig å bestemme hvilken type det er blir egennavnet værende igjen i listen. Når alle egennavnene er sjekket opp mot reglene startes *C.2 Finne andre forekomster* ved å først kalle metoden *delvisOrdning()*. Metoden inneholder ikke

funksjonaliteten som trengs for å utføre *C.2 Finne andre forekomster*, den oppretter kun en instans av *DelvisOrdning* og kaller metoden *finnDOForekomst1()*.

Når alle forekomstene av de klassifiserte egennavnene fra *C.1 Bruk av eksternt bevis* er funnet er det klart til å se om det er mulig å klassifisere flere egennavn ved hjelp av reglene som tilhører *C.3 Bruk av lister*. Som i fase *C.1* kaller *Kontroller* metoden *klassifiser()*, se Figur 18. Igjen, når alle egennavnene er sjekket opp mot reglene kalles metoden *delvisOrdning()* for å starte den siste fasen, *C.4 Finne delvis ordning*.

Denne klassen inneholder store deler av funksjonaliteten til fasene *C.1* og *C.3*. I tillegg inneholder klassen en del av funksjonaliteten til fasen *C.4*, blant annet sjekkes det om noen av ordene i første ord i setning er et stedsnavn.

DelvisOrdning inneholder metoder for å utføre fase *C.2 Finne andre forekomster* og *C.4 Finne delvise ordninger*. Metodene *finnDOForekomster1()* og *finnDOForekomster()* leter etter andre forekomster av egennavnet og de delvise ordningene. Metoden *finnDO()* som kalles i C.4 deler de klassifiserte egennavnene opp i delvise ordninger.

Regel er en abstrakt superklasse som alle reglene arver fra. Det er ikke mulig å lage instanser av en abstrakt klasse, klassen definerer kun fellesegenskaper til alle reglene, det vil si subklasssene. Klassen inneholder en abstrakt metode, *analyserSetning()*, som alle subklasser av *Regel* må implementere. Det er denne metoden som inneholder logikken til hver regel. Det vil si at koden som er vist i vedlegg A ligger inne i *analyserSetning()-metoden* til hver regel.

En av grunnene til at jeg har organisert reglene på denne måten er for at det skal være enkelt å legge til nye eller endre eksisterende regler. Det er også mulig å bruke polymorfi. Polymorfi vil si at programmeringen blir enklere fordi jeg slipper å ha orden på alt i programmet. Det er objektet selv som har kontroll på hvilken type det er. I metodene *genererRegler1/2()* opprettes det regler slik som vist i Tabell 15. Disse reglene blir så lagt inn i en liste. Når reglene brukes vet hvert objekt hvilken type det er, for eksempel vet *tittelRegel* at den er av regeltypen *TittelRegle*, det samme gjør de andre reglene. De forskjellige reglene ble forklart i kapittel 6.4.3, og i vedlegg A finnes det en oversikt over hvordan reglene er implementert.

Tabell 15 Liste med regler.

Regel tittelRegel = new TittelRegel();
Regel sierRegel = new SierRegel();
Regel stedSubRegel = new StedSubRegel();

I Figur 18 vises kun et *Regel-objekt*, dette er gjort for å lette leseligheten. I utgangspunktet skulle alle reglene i Figur 15 og Figur 16 blitt representert med et objekt hver.

FilLeser inneholder funksjonalitet for å lese fra en fil som ligger lagret på disk. Listene med egennavn og de semantiske settene ligger alle lagret på disk som rene tekstfiler. Disse må derfor leses inn for å kunne brukes i AIDaS.

Forekomst er en hjelpeklasse. Et *Forekomst-objekt* er et klassifisert egennavn med antall ganger det egennavnet forekommer i teksten.

DEL III: Evaluering og diskusjon

8 Evaluering

Dette kapittelet innholder en oversikt over evalueringen av AIDaS, hvordan den ble utført og hvilke resultater AIDaS fikk.

8.1 Fremgangsmåte

Evalueringen utføres som nevnt ved hjelp av en samling med 50 artikler fra Bergens Tidende. Før evalueringen kan begynne er det viktig å først lese gjennom samlingen og klassifisere egennavnene manuelt. Grunnen til at dette gjøres i forkant av selve evalueringen av systemet er for å være mest mulig objektiv under lesingen. Det vil si at jeg ikke skal være påvirket av resultatet AIDaS får når jeg selv klassifiserer egennavnene.

Etter at egennavnene er klassifisert manuelt, kjøres alle artiklene gjennom AIDaS. Når AIDaS har identifisert stedsnavnene i en artikkel, sammenlignes resultatet med den manuelle klassifiseringen. Dette gjøres manuelt. Det er viktig å merke seg at jeg kun ser på stedsnavnene under evalueringen, det vil si følgende tilfeller er relevante:

1. Stedsnavn blir klassifisert som *sted*
2. Stedsnavn blir klassifisert som *annet*
3. Stedsnavn blir ikke klassifisert
4. Annet blir klassifisert som *sted*

Med andre ord, hvis et personnavn blir klassifisert som *annet* eller ikke klassifisert i det hele tatt, vil ikke det ha noe å si på resultatet. Imidlertid, vil det ha innvirkning på resultatet hvis et personnavn blir klassifisert som et sted.

I det optimale system vil kun stedsnavn bli klassifisert som *sted*, men dette er dessverre vanskelig å oppnå. Det vil helt klart oppstå feil, og da er det viktig å finne ut hva som førte til feilen. For å lære mest mulig av evalueringen er det viktig å finne ut hvor feilen oppstod og hvilke endringer som kan gjøres for å rette den.

Etter at en artikkel er ferdig evaluert, regnes precision, recall og F-mål verdi ut for kun den artikkelen. Disse evalueringsmålene beskrives i kapittel 8.2. På denne måten kan man se om det finnes et mønster i hvilke typer artikler som gir gode resultat, og hvilke som ikke gir fult så gode resultat. Når alle artiklene er evaluert regnes det også ut en samlet precision, recall og F-mål verdi for hele samlingen. Som det ble nevnt i kapittel 5.2.1, gir Oslo-Bergen taggeren ut noen feil som igjen fører til feil i AIDaS, det vil derfor også være interessant å se bort fra disse feilene og regne ut en korrigert verdi for precision, recall og F-mål.

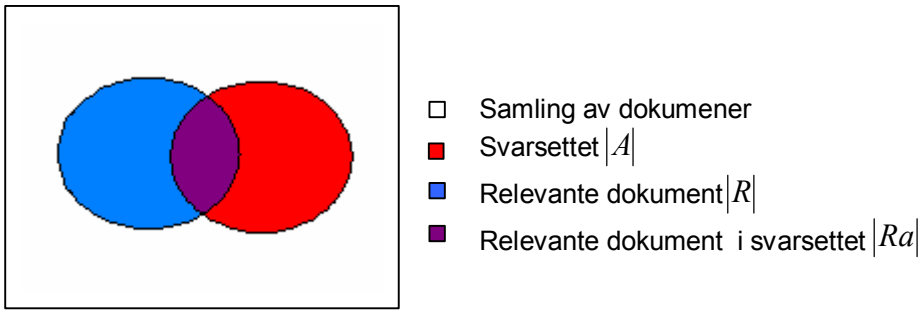
8.2 Evalueringsmål

Evalueringsmålene [5], [26] som ble brukt i evalueringen av AIDaS er precision, recall og F-mål.

Figur 19 viser definisjonen av evalueringsmålene, precison og recall. Den hvite firkanten er evalueringssamlingen, den blå sirkelen er alle stedsnavn i dokumentsamlingen. Den røde sirkelen er resultatet AIDaS gir av spørringen, det vil

si de egennavnene som er klassifisert som sted. Snittet mellom disse sirklene, det lilla feltet, er egennavnene i resultatet som faktisk er stedsnavn. Ut i fra dette kan man regne ut både precision og recall. Precision er andelen av funnet egennavn som er sted. Recall er andelen av stedsnavn som er funnet fra evalueringssamlingen.

$$Precision = \frac{|Ra|}{|A|} \qquad Recall = \frac{|Ra|}{|R|}$$



Figur 19 Precision og recall [26]

Det kan også være interessant å se en sammenheng mellom precision og recall under evalueringen av et system. I noen tilfeller er det slik at når man oppnår en høy recall så er det på bekostningen av precision og omvendt. Ved å kombinere precision og recall i et mål, kan man dermed se hvor godt systemet er med hensyn til både recall og precision. Et eksempel på et slikt mål er F-mål:

$$F - mål = \frac{1}{a\frac{1}{Precision} + (1-a)\frac{1}{Recall}}$$

a er en valgfri verdi mellom 0 og 1 som bestemmer vekten av precision og recall. I evalueringen av AIDaS er det ønskelig å oppnå både høy precision og recall, derfor er *a* satt til 0,5.

8.3 Resultater

I dette kapittelet presenteres resultatene AIDaS fikk etter evalueringen. Jeg skal først presentere det generelle resultatet AIDaS fikk. Deretter diskuterer jeg resultatene mer detaljert ved å analysere resultatene som er oppnådd i hver artikkel. Til slutt ser vi nærmer på hvilke feil som oppstod, og hvilke endringer som bør gjøres for å rette disse feilene. Det er viktig å merke seg at alle resultatene som presenteres er resultatene AIDaS fikk fra evalueringssamlingen.

8.3.1 Generelt resultat

Tabell 16 viser en oversikt over det generelle resultatet AIDaS fikk etter evalueringen. Jeg vil igjen presisere at det kun er identifisering av stedsnavn som er grunnlag for evalueringen.

Tabell 16 Generelt evalueringsresultat til AIDaS.

Precison i %	Recall i %	F-mål i %
80,7	86,7	83,6

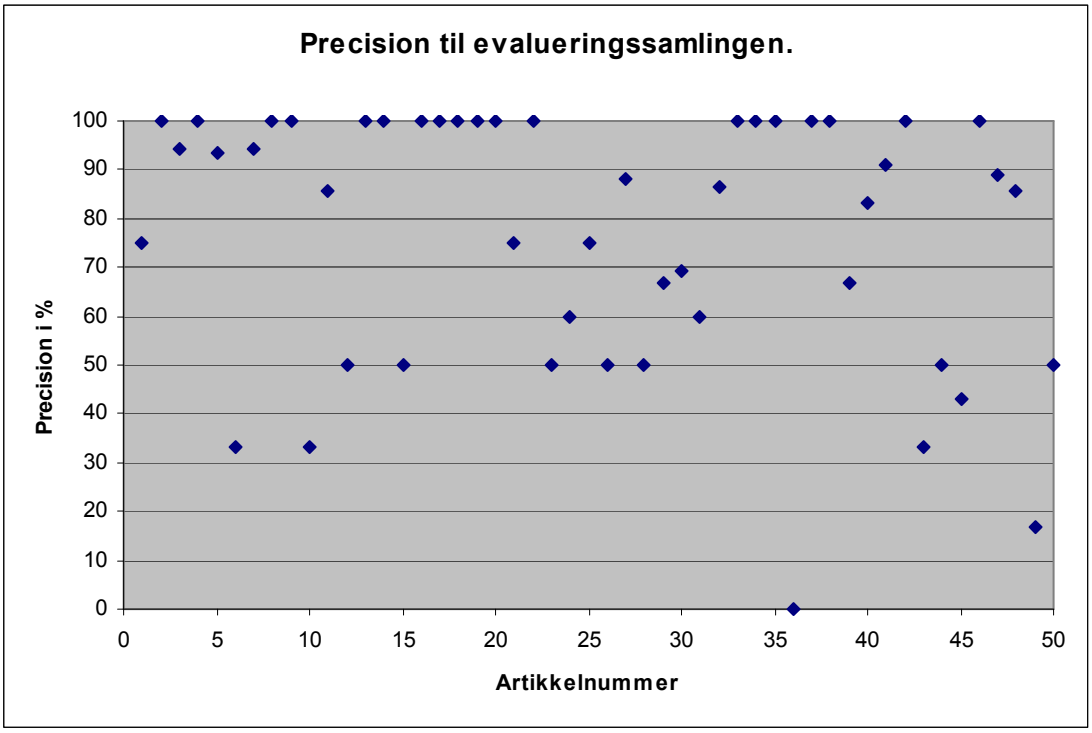
Alt i alt gjør AIDaS det svært bra. Det kan være flere grunner for dette. Artiklene i samlingen har vært relativt like da alle er nyhetsartikler fra Bergens Tidende. I tillegg er artiklene avgrenset innen et begrenset geografisk område. I nyhetsartikler er det svært vanlig at personer blir presentert med titler eller lignende. Dette gjør det lett å lage regler for å identifisere dem. Resultatet hadde ikke blitt like bra hvis evalueringssamlingen bestod av skjønnlitterære tekster, men dette er utenfor rammen til dette prosjektet.

Et annet element som kan ha gitt utslag på resultatet er at AIDaS slår sammen personnavn og organisasjonsnavn til en kategori, *annet*. De systemene jeg har sett på både i løpet av dette prosjektet og i høstprosjektet har ikke gjort denne forenklingen.

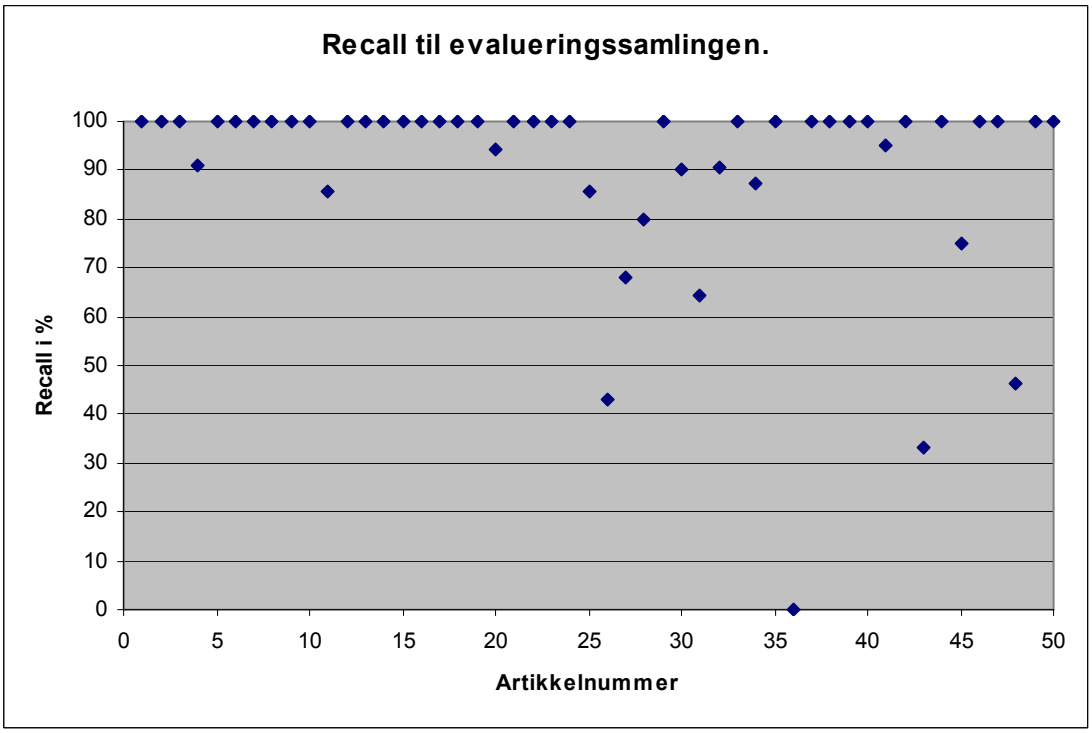
Jeg vil også presisere at listen med stedsnavn som jeg har brukt har vært svært god, og dette er en av grunnene til at jeg har fått høy recall. Listene med personnavn og organisasjonsnavn har jeg laget selv. Dette gjorde jeg ved å legge til navn etter hvert som de kom i artiklene i treningssamlingen. Jeg la også til noen organisasjoner før jeg startet selve evalueringen, siden jeg egentlig skulle ha en liste over organisasjoner i Hordaland.

8.3.2 Resultatet fordelt på artikler

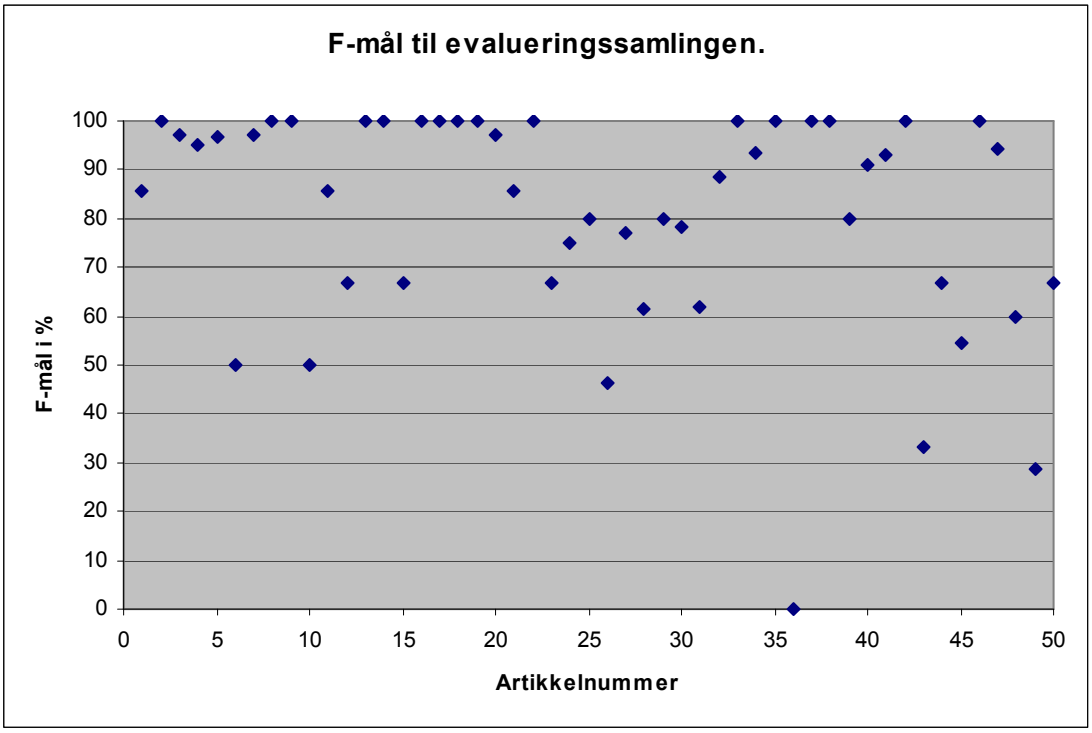
Figur 20 til Figur 22 viser grafiske fremstillinger av hvilke resultater hver enkelt artikkel har fått. Som vi kan se av Figur 20 er precisionsresultatet til artiklene varierende. Resultatet varierer faktisk fra 100 prosent til 0 prosent, hvor 19 av artiklene fikk en precision på 100 prosent, mens 29 artikler har bedre enn 80 prosent. Recallresultatene, er vist i Figur 21. Her ser vi at hele 34 av artiklene fikk en recall på 100 prosent, mens 43 har en recall på mer enn 80 prosent. F-mål er en kombinasjon av både precision og recall og dette kommer tydelig frem av Figur 22. F-mål viser hvor god balanse det er mellom precision og recall, og som vi kan se av Figur 22 blir resultatene litt mer spredt, men allikevel er det 32 artikler som har et F-mål på over 80 prosent. Grunnen til dette er at de gode recallresultatene kompenserer for de precisionsresultatene som ikke var fult så gode.



Figur 20 Precision til evalueringssamlingen.



Figur 21 Recall til evalueringssamlingen.



Figur 22 F-mål til evalueringssamlingen.

Det er interessant å se om det er noen sammenheng i de artiklene som har oppnådd gode resultater, og de som ikke har fullt så gode. For å gjøre dette deles artiklene inn i følgende kategorier:

1. Svært godt, 100 – 80 prosent F-mål
2. Godt, 80 – 60 prosent F-mål
3. Lite godt, 60 – 0 prosent F-mål

I den første gruppen, er det 32 artikler, det vil si over halve samlingen. Det som er typisk for disse artiklene er at det er ”seriøse” saker som tas opp, for eksempel ulykker, politikk og samfunnsrelaterte saker. I slike saker blir personer som intervjues eller som har tilknytning til saken presentert med en eller annen form for tittel, som for eksempel *lensmannsbetjent Einar Bystøl* eller *økonomisjef Thor Audun Fonnes*. Hvis personene kommer fra en organisasjon er det vanlig å skrive hvilken organisasjon de kommer fra etter personnavnet, *lensmannsbetjent Einar Bystøl ved Voss lensmannskontor*. Hvis en person ikke er presentert med en tittel, men kun uttaler seg om en sak er det vanlig å skrive en eller annen forekomst av *sier-verb*, for eksempel *sier Dan Johansson*. Det er ikke like vanlig å presentere stedsnavn i samme grad i slike artikler, da stedsnavnene som blir omtalt ofte er kjente for leseren. Men siden personnavn og organisasjonsnavn blir godt presentert er det lett å skille dem fra stedsnavnforekomstene. Grunnen til at AIDaS får så gode resultater i disse artiklene er med andre ord at journalisten gir mye informasjon i konteksten, slik at det blir lettere å skille mellom typen *sted* og *annet*.

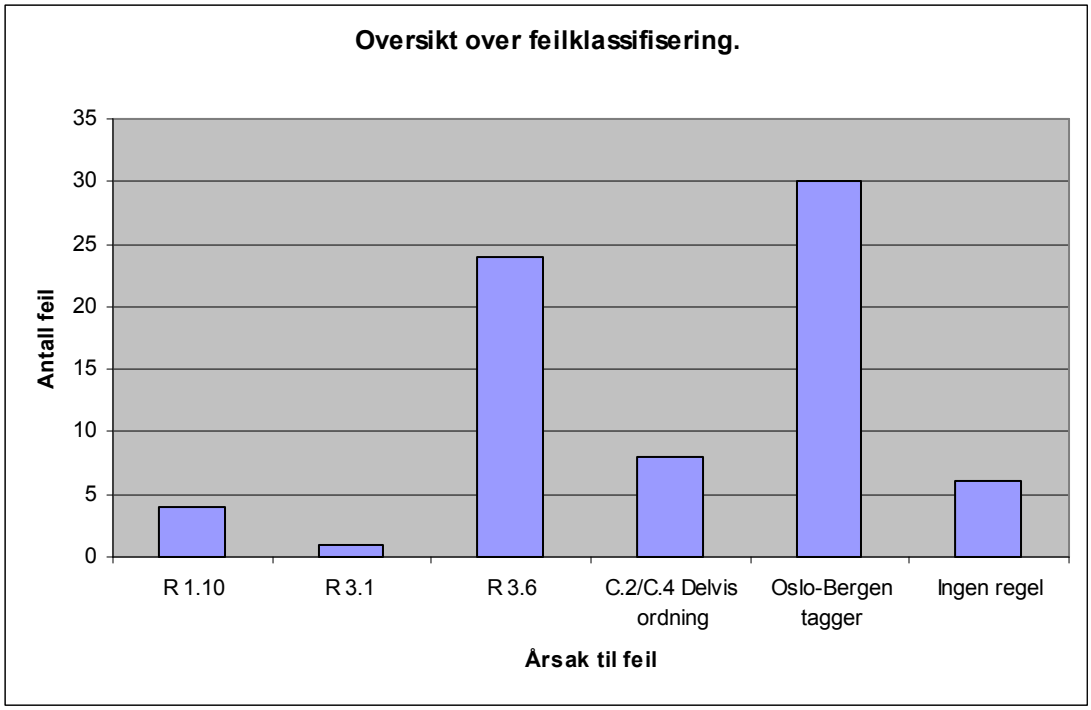
Den andre gruppen inneholder 11 artikler. Disse artiklene har fått dårligere resultat av en eller flere av følgende grunner:

- Artikkelen inneholder få egennavn. En liten feil gjør store utslag på resultatet.
- Utenlandske personnavn og organisasjonsnavn som ikke er presentert med titler eller lignende.
- Sportsartikler. Det kan være vanskelig å skille mellom når man snakker om et lag og når man snakker om et sted. I for eksempel artikler om fotball brukes sted og navn på lag om hverandre i en og samme artikkel. Det er lett for personer å forstå hvilken type det er snakk om, men ikke en datamaskin.
- Noen ganger er det vanskelig selv for en leser å bestemme om et egennavn er et sted eller organisasjon. For eksempel kan Bergen kommune i et tilfelle være sted, mens i et annet en organisasjon, i en og samme artikkel.

Den siste gruppen inneholder de resterende 7 artiklene. Det som skiller denne gruppen mest ut fra de andre er at seks av artiklene, inneholder kun fra et til tre stedsnavn, mens de inneholder en del organisasjonsnavn. Ellers er det flere artikler hvor *Rattsø-utvalget* blir diskutert. AIDaS er ikke like godt rustet til å håndtere egennavn som verken er person-, organisasjons- eller stedsnavn. Jeg vil spesielt kommentere den artikkelen som fikk 0 prosent i alle evalueringsmålene. Grunnen til dette er at det kun finnes et stedsnavn, *Molde*, i teksten, og det er plassert som første ord i en setning. Et egennavn som er første ord i setning blir kun klassifisert hvis det finnes andre forekomster av det i andre deler av teksten, eller hvis ordet står i listen over stedsnavn. Siden *Molde* ikke er et sted i Hordaland, er det ikke representert i listen over stedsnavn. På bakgrunn av dette er det ikke mulig for AIDaS å identifisere *Molde* som et stedsnavn.

8.3.3 Hvorfor klassifiserte AIDaS feil?

I dette kapittelet skal jeg se nærmere på hva som forårsaket at AIDaS klassifiserte feil under evalueringen.
Figur 23 gir en oversikt over hva som førte til feilklassifisering og hvor mange ganger den regelen eller ressursen ga ut feil resultat.



Figur 23 Årsak til feilklassifisering.

R 1.10 VerbRegel fungerer i de fleste tilfeller. Men i denne samlingen var det fire ganger den slo feil. To av gangene denne regelen ga feil type var med verbet *komme***,** det kunne derfor vært en mulighet å legge *komme* som et unntaksverb slik at et egennavn etter verbet ikke blir klassifisert av verbregelen. Et eksempel på når verbet *komme* brukes før et stedsnavn er som følger:

*I hennes sammenligning kommer **Bergen** desidert dårligst ut…*

De to andre verbene som førte til feil klassifisering er *gi* og *miste*. Når man ser nærmere på disse to situasjonene kan man diskutere hvorvidt de er stedsnavn eller organisasjonsnavn:

*… norsk teleindustri, som kan gi **Norge** en levevei uavhengig av naturinteresser.*
*Blir forslaget til Rattsø gjennomført, mister **Modalen** 5,6 millioner i inntekter.*

Man kan på en måte si at både *Norge* og *Modalen* i dette tilfellet er organisasjoner, selv om de er blitt klassifiser som stedsnavn i den manuelle klassifiseringen. Dette er typiske eksempler på at det kan være vanskelig å klassifisere egennavn rett selv for mennesker.

R 3.1 YrkeRegel gir en feil i følgende tilfelle:

*…mener det fortsatt er god plass for flere meglere i **Norges** nest største by.*

I dette tilfellet blir *Norges* klassifisert som annet på grunn av yrkeregelen. *Megler* er et yrke og er derfor i en liste over yrker, i tilfeller hvor et yrke står foran en preposisjon og et egennavn er det stor sannsynlighet for at egennavnet er en

organisasjon. Siden regelen kun fikk en feil, er det ingen grunn til å gjøre endringer på regelen.

R 3.6 PrepRegel gir som forventet flest feil av alle reglene. Regelen er imidlertid svært viktig for å klassifisere stedsnavn som ikke er i Hordaland. De fleste feilene regelen gir er mulig å luke bort ved å utvide annetlistene. Annetlistene innholder kun ord og uttrykk som jeg fant i testingen av AIDaS, og de er dermed ikke fullstendige. Hvis systemet hadde hatt litt mer trening og mer utfyllende lister ville antall feil blitt betraktelig mindre. Eksempel på ord og uttrykk som kunne vært lagt til annetlisten er som følger:

- *Rattsø-utvalget*
  *...sin lit til **Rattsø-utvalget**...*
  Rattsø-utvalget ble diskutert i mange artikler og bidrog til flere feil som kunne vært unngått ved enten å legge inn hele egennavnet, eller å legge utvalget til som et annet suffiks.
- Suffiksene *søknaden, kantine, bygg* osv
  *...en del av utvalgets standpunkt til **Ambassadeur-søknaden**.*
  *... tillate røyking i **Merinokantinen**.*
  *I kantinen i **Merinobygget**...*

C.2/C.4 Delvis ordning gir feil i åtte artikler. Hvis et egennavn er klassifisert feil og det finnes andre forekomster av disse egennavnene vil forekomstene også bli klassifisert feil. Vi kan på en måte si at delvis ordning gir følgefeil.

Oslo-Bergen taggeren er årsaken til de fleste feilene i AIDaS. Grunnen til dette er som forklart tidligere at den i noen tilfeller grupperer og klassifiserer ord og uttrykk feil. For eksempel ble ikke egennavnet Olaf Ryes vei gruppert sammen i dette tilfellet:

*...som ligger i **Olaf Ryes vei** 17...*

Denne feilklassifiseringen førte til at fem stedsnavn ikke ble funnet i AIDaS. Et annet tilfelle hvor Oslo-Bergen taggeren gjorde feil er som følger:

*...i **Os skule** torsdag kveld...*

Os blir ikke slått sammen med *skule*, og siden *Os* er et sted som står i stedsnavnlisten blir det klassifisert som et sted.

Ingen regel vil si at det ikke fantes noen regel som kunne klassifisere egennavnet. I de fleste tilfellene var det et adjektiv eller adverb før stedsnavnet. Det vil være enkelt å ta høyde for dette ved å utvide reglene til også å håndtere slike tilleggsopplysninger.

8.4 Diskusjon

Dette kapittelet har presentert resultatene som AIDaS fikk etter evaluering. Resultatene i Tabell 16 gir ikke mye informasjon hvis man ikke ser hvilke resultater andre liknende systemer har oppnådd. Jeg har derfor valgt å sammenligne resultatet til AIDaS med resultatene som de norske systemene i Nomen Nescio Project og ARNER fikk. Under sammenlikningen er det viktig å være klar over at det kan ligge

forskjellige kriterier til grunn for resultatene og at samlingene kan være forskjellige. Man kan imidlertid få en pekepinne på hvordan systemene er i forhold til hverandre.

Tabell 17 Evalueringsresultatene til liknende norske systemer i forhold til AIDaS [5] [17].

	Precision i %	**Recall i %**	**F-mål i %**
Norsk MBL system	17	80	28
Norsk ME system	14	80	23,8
Norsk CG system	10	60	17,1
ARNER	28,3	99,56	44
AIDaS	80,7	86,7	83,6

Tabell 17 gir en oversikt over resultatene de norske systemene, MBL, ME og CG i Nomen Nescio Project og ARNER fikk etter evaluering i forhold til AIDaS. Resultatene i Tabell 17 er kun for typen sted. Som vi kan se har AIDaS fått bedre precison og F-mål enn samtlige av de andre systemene. Når det gjelder recall har AIDaS fått bedre resultat enn systemene i Nomen Nescio Project, men dårligere enn ARNER.

Det er viktig å ta høyde for at AIDaS kun er beregnet på nyhetstekster innefor et geografisk domene og at de andre systemene også har klassifisert egennavn inn i andre typer. Samlingene som ble brukt for å trene og evaluere AIDaS var ikke store, og det er helt klart at det finnes situasjoner AIDaS ikke tar høyde for og som dermed vil føre til feil. I tillegg er det også viktig å huske at AIDaS er laget for å identifisere stedsnavn i nyhetstekster og derfor ikke vil gi like gode resultater i samlinger med skjønnlitterære tekster.

Som nevnt flere ganger er Oslo-Bergen taggeren en viktig ressurs for AIDaS, men dessverre gir den også ut flere feil, og representerer den største feilkilden. Hvis vi ser bort fra feilene som Oslo-Bergen taggeren gir får AIDaS bedre resultater. Tabell 18 gir en oversikt over resultatene til AIDaS hvis vi ser bort fra feilene fra Oslo-Bergen taggeren. Som vi kan se blir resultatene en god del bedre, spesielt precision.

Tabell 18 Resultatene til AIDaS uten feil fra Oslo-Bergen taggeren.

Precison i %	**Recall i %**	**F-mål i %**
84,6	88,8	86,6

9 Konklusjon

Formålet med oppgaven var å videreutvikle den abstrakte løsningsmetoden fra høstprosjektet. Det vil si å designe og implementere en prototyp for å automatisk identifisere stedsnavn i norske nyhetstekster.

For å videreutvikle løsningsmetoden var det viktig å finne ut hvilke svakheter den hadde. Det var derfor nødvendig å implementere den og se hvordan den fungerte i praksis. Selv om løsningsmetoden fungerte godt på et forhåndsdefinert scenario, viste det seg at den inneholdt en del feil i forbindelse med første ord i setning og håndtering av sekvenser. Ved å ta utgangspunkt i svakhetene til løsningsmetoden og et oppdatert litteraturstudium, ble det utviklet en ny løsning. Denne løsningen dannet grunnlaget for AIDaS.

AIDaS oppnådde svært gode resultater under evalueringen med et F-mål på hele 83,6 prosent. Resultatet til AIDaS er bedre enn ARNER og de norske systemene i Nomen Nescio Project. En av grunnene til at AIDaS fikk så gode resultater i forhold til de andre systemene, er at AIDaS har et begrenset bruksområde ved at den er laget kun for å identifisere stedsnavn i nyhetstekster. Nyhetstekster inneholder en viss struktur og personer og organisasjoner blir ofte godt presentert. Denne strukturen kan brukes for å skille personnavn og organisasjonsnavn fra stedsnavn.

Test- og evalueringssamlingen som ble brukt for å utvikle og evaluere AIDaS inneholdt kun tekster fra Bergens Tidende. Konteksten til artiklene var lokale tekster fra Hordaland. Som hjelpemiddel under klassifiseringen ble det derfor brukt en liste over alle offisielle stedsnavn i fylket. Grunnen til at en slik liste ble tatt i bruk er at lokale stedsnavn ofte er kjent blant folk i området, og det er derfor ikke mye informasjon rundt dem i nyhetsartikler. Selv om AIDaS bruker lister kan det fremdeles identifisere andre stedsnavn, men da kun med hjelp av eksternt bevis eller preposisjonsregelen. En annen ressurs AIDaS har stor nytte av er Oslo-Bergen taggeren. Uten denne hadde resultatet ikke blitt på langt nær så godt som det ble, dette på tross av at Oslo-Bergen taggeren er den størst feilkilden AIDaS har.

AIDaS bruker regler for å identifisere stedsnavn. Det norske skriftspråket har stor grad av valgfrihet og gir dermed flere måter å skrive samme tekst på. Videre er språket i kontinuerlig utvikling og det vil dermed alltid finnes situasjoner som man ikke kan fange opp i reglene. De fleste reglene i AIDaS er derfor laget så generelle som mulig, slik at man i utgangspunktet skal kunne dekke de fleste scenarioene.

AIDaS er utviklet for å identifisere stedsnavn, men det er lagt stor vekt på at systemet skal være utvidbart. Dette er en av grunnene til at systemet er delt inn i moduler og faser. Det skal være relativt enkelt å endre funksjonaliteten til noen av fasene hvis det er ønskelig. Det vil si at det vil være trivielt å klassifisere egennavn i flere typer, for eksempel personnavn, organisasjonsnavn eller lignende ved å kun legge til nye regler i fasene *C.1 Bruk av eksternt bevis* og *C.3 Bruk av lister*. AIDaS er med andre ord et rammeverk som ikke nødvendigvis må brukes i et geografisk informasjonsgjenfinningssystem, men gjerne også i tradisjonelle informasjons-gjenfinningssystem eller maskinoversettingssystem. Med bakgrunn i dette vil jeg til slutt konkludere med at AIDaS er et system som er godt egnet for å identifisere stedsnavn i norske nyhetstekster.

10 Videre arbeid

Selv om AIDaS fikk et svært godt resultat under evalueringen er det flere elementer som det er mulig å jobbe videre med. For eksempel er det mulig å utvide systemet slik at det ikke er domenespesifikt. Vi kan på en måte si at AIDaS er begrenset innen to områder:

1. Geografisk
2. Type inndata

AIDaS bør utvides til å omfatte hele Norge, siden systemet også er utviklet til å kunne håndtere andre stedsnavn. En av forandringene som må gjøres ved en utvidelse av det geografiske området er å bytte ut listen over stedsnavn i Hordaland med en liste over byer, fylker eller lignende i Norge. Det ville da også vært hensiktsmessig å endre på listen med organisasjonsnavn til å inneholde de mest kjente organisasjonene i Norge.

Reglene i AIDaS er laget med utgangspunkt i avisartikler. Men dokumenter i et geografisk informasjonsgjenfinningssystem kan også være av andre typer, for eksempel skjønnlitterære tekster som omhandler stedsnavn. For å dekke andre sjangere kan man utvide reglene i AIDaS til å også kunne identifisere stedsnavn i for eksempel skjønnlitterære tekster.

AIDaS kan også utvides til å identifisere andre typer egennavn, som for eksempel organisasjon, person, verk eller liknende. I den forbindelse kan man legge til en modul som kan identifisere komplekse egennavn, da organisasjoner ofte inneholder konjunksjonsuttrykk.

11 Referanser

[1] Røyneberg E.:
Tekstanalyse for Geografisk informasjonsgjenfinning
Høstprosjekt i DT4710 Informasjonsforvaltning fordypningsemne, høsten 2004
http://www.idi.ntnu.no/grupper/if/publikasjoner/ProsjektOppgave.Ellen.pdf

[2] Ravin Y., Wacholder N.:
IBM Research Report. Extracting Names from Natural-Language Text, 1997
http://citeseer.ist.psu.edu/cache/papers/cs/2282/http:zSzzSzwww.research.ibm.comzSzpeoplezSzrzSzravinzSz20338.pdf/ravin97extracting.pdf

[3] McDonald D.D.:
Internal and External Evidence in the Identification and Semantic Categorization of Proper Names
Corpus Processing for Lexical Acquisition, kap 2 side 21-39, MIT press, Cambridge , MA 1996
http://citeseer.ist.psu.edu/cache/papers/cs/27836/http:zSzzSzacl.ldc.upenn.eduzSzWzSzW93zSzW93-0104.pdf/mcdonald96internal.pdf

[4] Norsk språkråd
http://www.sprakrad.no/
http://www.sprakrad.no/templates/Page.aspx?id=7660 (sist lest 10.05.2005)

[5] Jónsdóttir A.B:
ARNER, what kind of name is that?
Hovedfagsoppgave, Institutt for Lingvistikk, Universitet i Oslo, 2003

[6] Norsk lovdata
http://www.lovdata.no/all/tl-19900518-011-0.html#1 (sist lest 23.05.2005)

[7] Statens namnekonsulentar for norske kvenske og samiske stadnamn
http://www.stadnamn.org/
http://www.stadnamn.org/?id=stedsnavn (sist lest 21.02.2005)

[8] Krupka G.R. og Hausman K.:
Description of the NetOwl Extractor System as Used for MUC-7
Proceedings of 7th Message Understanding Conference. 19 april - 1 mai, 1998
http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/muc_7_proceedings/isoquest.pdf

[9] Mikheev A., Moens M og Grover C.:
Named Entity recognition without gazetteers
Proceedings of the Ninth Conference of the European Chapter of the Association for Computational Linguistics (EACL'99), pp1-8. June 8-12, 1999
http://www.ltg.ed.ac.uk/papers/99mikheev_eacl.pdf

[10] Ravin Y., Wacholder N., og Choi M.:
Disambiguation of Proper Names in Text
Proceedings of the fifth conference on Applied natural language processing, side

202-208
Morgan Kaufmann Publishers Inc. San Francisco. CA, USA, 1997
http://delivery.acm.org/10.1145/980000/974587/p202-wacholder.pdf?key1=974587&key2=6842821011&coll=GUIDE&dl=GUIDE&CFID=29944818&CFTOKEN=10135216

[11] Fukumoto J., Masui F. Shimohata M. og Sasaki M.:
Oki Electric Industry: Description of the Oki System as Used for MUC-7
Proceedings of 7th Message Understanding Conference. 19 april - 1 mai, 1998
http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/muc_7_proceedings/oki_muc7.pdf

[12] Fukumoto J., Masui F. Shimohata M. og Sasaki M.:
Oki Electric Industry: Description of the Oki System as Used for MET-2
Proceedings of 7th Message Understanding Conference. 19 april - 1 mai, 1998
http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/muc_7_proceedings/oki_met2.pdf

[13] Maynard D., Tablan V., Bontcheva K., Cunningham H. og Wilks Y:
MUSE: a Multi-Source Entity recognition system
Submitted to Computers and the Humanities, 2003
http://gate.ac.uk/sale/muse/muse.pdf

[14] Maynard D., Cunningham H., Bontcheva K. og Dimitrov M.:
Adapting A Robust Multi-Genre NE System for Automatic Content Extraction
Proceedings of the Tenth International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA 2002)
http://gate.ac.uk/sale/aimsa/aimsa.pdf

[15] Maynard D.:
Mottatt e-post fra Diana Maynard 01.10.2004

[16] En automatisk navnegjenkjenner for norsk, svensk og dansk
http://spraakdata.gu.se/nn/NorFa-00.html (sist lest 24.11.2004)

[17] Johannessen J. B., Hagen K., Haaland Å., Kokkinakis D., Meurer P., Bick E., Haltrup D., Jónsdottir A, B. og Nøklestad A.:
Named Entity Recognition for the Mainland Scandinavian Languages

[18] Hagen K.:
Mottatt e-post fra Kristin Hagen 04.10.2004

[19] Named Entity Scores – English:
Proceedings of 7th Message Understanding Conference. 19 april - 1 mai, 1998
http://ww.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/ne_english_score_report.html (sist lest 27.04.2005)

[20] Mikheev A., Grover C. og Moens M.:
Description of the LTG System used for MUC-7
Proceedings of 7th Message Understanding Conference. 19 april - 1 mai, 1998

http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/muc_7_proceedings/ltg_muc7.pdf

[21] Johannesen J. B.:
En grammatisk tagger for norsk (bokmål), Tekstlaboratoriet, 1998

[22] Johannesen J. B.:
The Oslo-Bergen-Tagger and The Nomen Nescio Prosject
http://www.nada.kth.se/~hercules/scandsum/Janne.pdf (sist lest 24.04)

[23] Gulla J.A.:
Text Operations, Foiler i faget TDT 4215
http://www.idi.ntnu.no/emner/tdt4215/slides/TDT4215%20TextOperations%20V2005.pdf (sist lest 24.04)

[24] Sentralt stedsnavnregister:
http://www.statkart.no/virksomh/forvaltning/navnlov/ssr.html (siste lest 27.05.2005)

[25] Web Services
http://www.w3.org/2002/ws/ (sist lest 31.05.2005)

[26] Baeza-Yates R. og Ribeiro-Neto B.:
Modern Information Retrieval
Addison-Wesley, 1999
ISBN 0-201-39829-X

DEL IV: Vedlegg

A. Kodeutklipp av reglene i AIDaS

Dette vedlegget inneholder utklipp av de fleste reglene i AIDaS i Figur 24 til Figur 37. Reglene som ikke er representert er like andre regler. For en fullstendig oversikt over alle reglene henviser jeg til vedlegg D hvor hele kildekoden er gjengitt.

For at det skal være enklere å lese disse utklippene av reglene er det viktig å merke seg følgende:

- Variabelen *egennavn* er et objekt som representerer det egennavnet som skal klassifiseres. Et egennavnobjekt inneholder all informasjon om et egennavn, som navn, posisjon og setningsnummer.
- Variabelen *setning* er en *ArrayList* som inneholder alle ordene som er i samme setning som det egennavnet som skal klassifiseres. Ordene i listen er en instans av klassen Ord. Det vil si at alle ordene i listen inneholder informasjon som lemma og features.

Jeg vil også henvise til forklaringen av reglene i kapittel 6.4.3 og den tekniske beskrivelsen av systemet i kapittel 7.

```
int lengde = 0;
boolean liten = false;
//sjekker om alle boktavene er store
while (!liten&& lengde< egennavn.getNavn().length()) {
    if(Character.isLowerCase(egennavn.getNavn().charAt(lengde))||
            Character.isDigit(egennavn.getNavn().charAt(lengde))){
        liten = true;
    }
    lengde++;
}
//skiller ut de som er skrevet med stor bokstav
if(!liten){
    storBokstav.add(egennavn);
    return true;
}
return false;
```

Figur 24 StorBokstavRegel.

```
// denne regelen ser om det er en tittel
// rett før egennavnet => annet
// f.eks statsminister Bondevik
Ord tittel = (Ord) setning.get(egennavn.getPos() - 1);
for (int i = 0; i < (tittel.getFeatures()).size(); i++) {
    if (tittel.getFeatures().get(i).equals("@tittel")) {
        annet.add(egennavn);
        egennavn.getOrd().addFeature("ANNET");
        return true;
    }
}

// denne regelen ser om det er en tittel
// før en preposisjon før et egennavn => annet
// f.eks leder av NSB, men ikke lederen av NSB
Ord ord = (Ord) setning.get(egennavn.getPos() - 2);
Ord tittel = (Ord) setning.get(egennavn.getPos() - 1);
for (int i = 0; i < (ord.getFeatures()).size(); i++) {
    for (int j = 0; j < (tittel.getFeatures()).size(); j++) {
        if (tittel.getFeatures().get(j).equals("prep")
                && ord.getFeatures().get(i).equals("@tittel")) {
            annet.add(egennavn);
            egennavn.getOrd().addFeature("ANNET");
            return true;
        }
    }
}
```

Figur 25 TittelRegel.

```
// sier NOEN/NOE til egennavnet
Ord sier = (Ord) setning.get(egennavn.getPos() - 3);
Ord pers = (Ord) setning.get(egennavn.getPos() - 2);
Ord prep = (Ord) setning.get(egennavn.getPos() - 1);
for (int i = 0; i < verb.size(); i++) {
    if (sier.getLemma().equals((String) (verb.get(i)))) {
        for (int j = 0; j < (prep.getFeatures()).size(); j++) {
            if (prep.getFeatures().get(j).equals("prep")) {
                for (int l = 0; l < (pers.getFeatures()).size(); l++) {
                    if (pers.getFeatures().get(l).equals("pron")
                            ||pers.getFeatures().get(l).equals("ANNET")){
                        annet.add(egennavn);
                        egennavn.getOrd().addFeature("ANNET");
                        return true;
                    }
                }
            }
        }
    }
}
```

Figur 26 SierRegel nr 3.

```
// egennavnet har et stedssubstantiv foran seg
Ord stedSub = (Ord) setning.get(egennavn.getPos() - 1);
Ord prep = (Ord) setning.get(egennavn.getPos() - 2);
for (int j = 0; j < (prep.getFeatures()).size(); j++) {
    if (prep.getFeatures().get(j).equals("prep")) {
        ArrayList liste = lesFraFil("tekst/stedSub.txt", "");
        for (int i = 0; i < liste.size(); i++) {
            if (stedSub.getLemma().equals((String) liste.get(i))) {
                stedsnavn.add(egennavn);
                egennavn.getOrd().addFeature("STED");
                return true;
            }
            if(((String) liste.get(i)).startsWith("*")){
                String suffix = ((String) liste.get(i)).
                substring(1,((String) liste.get(i)).length());
                if (stedSub.getLemma().endsWith(suffix)) {
                    stedsnavn.add(egennavn);
                    egennavn.getOrd().addFeature("STED");
                    return true;
                }
            }
        }
    }

}
```

Figur 27 StedSubRegel.

```
// selve forkortelsen blir klassifisert
Ord fulltNavn = (Ord) setning.get(egennavn.getPos() - 2);
Ord vPar = (Ord) setning.get(egennavn.getPos() - 1);
Ord hPar = (Ord) setning.get(egennavn.getPos() + 1);
if (vPar.getOrd().equals("(") && hPar.getOrd().equals(")")) {
    for (int i = 0; i < fulltNavn.getFeatures().size(); i++) {
        if ((fulltNavn.getFeatures()).get(i).equals("ANNET")) {
            annet.add(egennavn);
            egennavn.getOrd().addFeature("ANNET");
            return true;
        }
    }
}
```

Figur 28 ForkRegel.

```
//han sammen med Frode Skogvold
Ord pers = (Ord)(setning.get(egennavn.getPos()-3));
Ord sammen = (Ord)(setning.get(egennavn.getPos()-2));
Ord med = (Ord)(setning.get(egennavn.getPos()-1));
if(sammen.getOrd().equals("sammen")&& med.getOrd().equals("med")){
    for (int l = 0; l < (pers.getFeatures()).size(); l++) {
        if (pers.getFeatures().get(l).equals("pron")) {
            annet.add(egennavn);
            egennavn.getOrd().addFeature("ANNET");
            return true;
        }
    }

}
```

Figur 29 SammenMedRegel.

```
//Ingrid, søsteren
Ord komma = (Ord) setning.get(egennavn.getPos() + 1);
Ord pers = (Ord) setning.get(egennavn.getPos() + 2);
if(komma.getOrd().equals(",")){
    ArrayList liste = lesFraFil("tekst/person.txt", "#");
    for (int i = 0; i < liste.size(); i++) {
        if(pers.getLemma().equals( (String)liste.get(i))){
            annet.add(egennavn);
            egennavn.getOrd().addFeature("ANNET");
            return true;
        }
    }
}
```

Figur 30 Apposisjonsregel.

```
// egennavnet etter en konjunksjon vil vare av samme
// type som det før konjunksjonen
Ord og = (Ord) setning.get(egennavn.getPos() - 1);
Ord navn = (Ord) setning.get(egennavn.getPos() - 2);
if (og.getLemma().equals("og")||og.getLemma().equals("samt") ) {
    for (int i = 0; i < (navn.getFeatures()).size(); i++) {
        if (navn.getFeatures().get(i).equals("ANNET")) {
            annet.add(egennavn);
            egennavn.getOrd().addFeature("ANNET");
            return true;
        }
        if (navn.getFeatures().get(i).equals("STED")) {
            stedsnavn.add(egennavn);
            egennavn.getOrd().addFeature("STED");
            eg = egennavn;
            return true;
        }
    }
}
```

Figur 31 KonjRegel.

```
// verb før et egennavn => annet, med unntak
// av forlate og slutte og være
Ord verb = (Ord) setning.get(egennavn.getPos() - 1);
if (verb.getLemma().equals("forlate")) {
    return false;
}
if (verb.getLemma().equals("slutte")) {
    return false;
}
if (verb.getLemma().equals("være")) {
    return false;
}

for (int i = 0; i < (verb.getFeatures()).size(); i++) {
    if (verb.getFeatures().get(i).equals("verb")) {
        annet.add(egennavn);
        egennavn.getOrd().addFeature("ANNET");
        return true;
    }
}
```

Figur 32 VerbRegel.

```
// egennavnet er en person
for (int i = 0; i < fornavn.size(); i++) {
    if(person.size()>= 2 && ((String)person.get(0))
            .equals((String)fornavn.get(i))){
        annet.add(egennavn);
        egennavn.getOrd().addFeature("ANNET");
        return true;
    }
}
```

Figur 33 PersRegel.

```
// lederen i Venstre => Venstre blir annet
Ord yrke = (Ord) setning.get(egennavn.getPos() - 2);
Ord prep = (Ord) setning.get(egennavn.getPos() - 1);
for (int j = 0; j < (prep.getFeatures()).size(); j++) {
    if(prep.getFeatures().get(j).equals("prep")){
        ArrayList liste = lesFraFil("tekst/personYrke.txt","");
        for (int i = 0; i < liste.size(); i++) {
            if(yrke.getLemma().equals((String)liste.get(i))){
                annet.add(egennavn);
                egennavn.getOrd().addFeature("ANNET");
                return true;
            }
        }
    }
}
```

Figur 34 YrkeRegl.

```
Ord prep = (Ord) setning.get(egennavn.getPos() - 1);
for (int i = 0; i < (prep.getFeatures()).size(); i++) {
    if (prep.getFeatures().get(i).equals("prep")) {
        stedsnavn.add(egennavn);
        egennavn.getOrd().addFeature("STED");
        return true;
    }
}
```

Figur 35 PrepRegel.

```
//Skansentunnelen og EØS-avtalen
ArrayList annetSuffix = lesFraFil("tekst/annetSuffix.txt", "#");
for (int j = 0; j < annetSuffix.size(); j++) {
    if(egennavn.getNavn().endsWith( (String) annetSuffix.get(j))){
        annet.add(egennavn);
        egennavn.getOrd().addFeature("ANNET");
        return true;
    }
}
```

Figur 36 PrepRegelListe.

```
ArrayList sted = lesFraFil("tekst/sted.txt", "#");
for (int i = 0; i < sted.size(); i++) {
    if(egennavn.getNavn().equals((String)sted.get(i))){
        stedsnavn.add(egennavn);
        egennavn.getOrd().addFeature("STED");
        return true;
    }
}
```

Figur 37 ListeRegel.

B. Meldinger mellom AIDaS og Oslo-Bergen taggeren

Dette vedlegget inneholder eksempel på hvordan meldingene som sendes mellom AIDaS og Oslo-Bergen taggeren ser ut. Figur 38 er et eksempel på forespørselen fra AIDaS. Som vi kan se inneholder den teksten som skal tagges i tillegg til en del konfigurasjonsparametre. Figur 39 viser hvordan deler av en mottatt melding ser ut. Siden meldingen er av typen base64encoding, er den ikke forståelig for mennesker.

```
- <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" SOAP-
    ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
    ENC="http://schemas.xmlsoap.org/soap/encoding" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  - <SOAP-ENV:Body>
    - <cgps:tagTextBase64 xmlns:cgps="http://www.aksis.uib.no/cgp">
        <language xsi:type="xsd:string">nbo</language>
        <niveau xsi:type="xsd:string">SND</niveau>
        <text xsi:type="xsd:string">MC-fører skadd på Askøy . En 38 år gammel motorsyklist er fraktet til sykehus i
          ambulanse etter at han i morges ble funnet liggende i veibanen på Erdal. Mannen hadde skader i brystet da
          han ble funnet, men var ved bevissthet. Hvor alvorlig skadd han er, er fortsatt uklart. Motorsyklisten
          behandles nå på Haukeland Universitetssykehus. Ifølge en pressemelding fra sykehuset er mannen lettere
          skadet, og tilstanden er stabil. Ifølge politiet kom han kjørende på Askvegen da han kolliderte med en
          varebil i krysset ved det gamle postkontoret på Erdal. Bilen kom fra Grensedalen. Dette er et veldig kinkig
          kryss med dårlig sikt. Her bør noe gjøres, for det er fort gjort å krasje her, sier politibetjent Finn Martin
          Systad ved Askøy lensmannskontor til BT. Politiet fikk melding om ulykken klokken 08.16. Ulykkesstedet
          ligger på riksvei 563, ved avkjørselen til gamleveien på Askøy.</text>
        <encoding xsi:type="xsd:string">utf-8</encoding>
        <output-format xsi:type="xsd:string">WORD-LEMMA-TAGS-XML</output-format>
      - <feature-filter>
          <f xsi:type="xsd:string">SUBST</f>
          <f xsi:type="xsd:string">ADJ</f>
          <f xsi:type="xsd:string">VERB</f>
          <f xsi:type="xsd:string">ADV</f>
          <f xsi:type="xsd:string">PREP</f>
          <f xsi:type="xsd:string">INTERJ</f>
          <f xsi:type="xsd:string">DET</f>
          <f xsi:type="xsd:string">KVANT</f>
          <f xsi:type="xsd:string">DEM</f>
          <f xsi:type="xsd:string">PRON</f>
          <f xsi:type="xsd:string">POSS</f>
          <f xsi:type="xsd:string">PERS</f>
          <f xsi:type="xsd:string">REFL</f>
          <f xsi:type="xsd:string">INF-MERKE</f>
          <f xsi:type="xsd:string">SBU</f>
          <f xsi:type="xsd:string">FORK</f>
          <f xsi:type="xsd:string">FOREIGN</f>
          <f xsi:type="xsd:string">UKJENT</f>
          <f xsi:type="xsd:string">ENT</f>
          <f xsi:type="xsd:string">FL</f>
          <f xsi:type="xsd:string">BE</f>
          <f xsi:type="xsd:string">UB</f>
          <f xsi:type="xsd:string">MASK</f>
          <f xsi:type="xsd:string">FEM</f>
          <f xsi:type="xsd:string">NT</f>
          <f xsi:type="xsd:string">PROP</f>
          <f xsi:type="xsd:string">@SUBJ</f>
          <f xsi:type="xsd:string">@I-OBJ</f>
          <f xsi:type="xsd:string">@OBJ</f>
          <f xsi:type="xsd:string">@TITTEL</f>
          <f xsi:type="xsd:string">&lt;OVERSKRIFT&gt;</f>
        </feature-filter>
        <total-disambiguate xsi:type="xsd:boolean" />
      </cgps:tagTextBase64>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figur 38 Eksempel på sentmsg.xml.

```
<?xml version="1.0" ?>
- <SOAP-ENV:Envelope xmlns:cgps="http://www.aksis.uib.no/cgp" xmlns:SOAP-
  ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
  ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" SOAP-
  ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  - <SOAP-ENV:Body>
      <cgps:sResponseBase64 xsi:type="SOAP-
        ENC:base64">PG5ld2xpbmUvPjx3b3JkIGlkPSJ3MCIgbGVtbWE9Ik1DLWbDuHJlciIgZmVhdHVyZXM9InN1YnN
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```


Figur 39 Eksempel på receivedmsg.xml.

C. xml-fil etter dekryptering

Dette vedlegget inneholder et eksempel på et resultat fra Oslo-Bergen taggeren etter dekryptering. Det er denne xml-filen som blir brukt i AIDaS for å finne egennavn og å klassifisere dem.

```
- <newline>
  - <set>
      <word id="w0" lemma="MC-fører" features="subst mask ub ent @tittel">MC-fører</word>
      <word id="w1" lemma="skade" features="verb">skadd</word>
      <word id="w2" lemma="på" features="prep">på</word>
      <word id="w3" lemma="Askøy" features="subst prop">Askøy</word>
    </set>
  - <set>
      <word id="w5" lemma="en" features="det mask ent kvant">En</word>
      <word id="w6" lemma="38" features="det fl kvant">38</word>
      <word id="w7" lemma="år" features="subst ub fl">år</word>
      <word id="w8" lemma="gammel" features="adj ub ent">gammel</word>
      <word id="w9" lemma="motorsyklist" features="subst mask ub ent @subj">motorsyklist</word>
      <word id="w10" lemma="være" features="verb">er</word>
      <word id="w11" lemma="frakte" features="verb">fraktet</word>
      <word id="w12" lemma="til" features="prep">til</word>
      <word id="w13" lemma="sykehus" features="subst ub ent">sykehus</word>
      <word id="w14" lemma="i" features="prep">i</word>
      <word id="w15" lemma="ambulanse" features="subst mask ub ent">ambulanse</word>
      <word id="w16" lemma="etter at" features="sbu">etter at</word>
      <word id="w17" lemma="han" features="pron mask ent pers @subj">han</word>
      <word id="w18" lemma="i morges" features="adv">i morges</word>
      <word id="w19" lemma="bli" features="verb">ble</word>
      <word id="w20" lemma="finne" features="verb">funnet</word>
      <word id="w21" lemma="ligge" features="adj @obj">liggende</word>
      <word id="w22" lemma="i" features="prep">i</word>
      <word id="w23" lemma="veibane" features="subst mask be ent">veibanen</word>
      <word id="w24" lemma="på" features="prep">på</word>
      <word id="w25" lemma="Erdal" features="subst prop">Erdal</word>
    </set>
  - <set>
      <word id="w27" lemma="mann" features="subst mask be ent @subj">Mannen</word>
      <word id="w28" lemma="ha" features="verb">hadde</word>
      <word id="w29" lemma="skade" features="subst mask ub fl @obj">skader</word>
      <word id="w30" lemma="i" features="prep">i</word>
      <word id="w31" lemma="bryst" features="subst be ent">brystet</word>
      <word id="w32" lemma="da" features="sbu">da</word>
      <word id="w33" lemma="han" features="pron mask ent pers @subj">han</word>
      <word id="w34" lemma="bli" features="verb">ble</word>
      <word id="w35" lemma="finne" features="verb">funnet</word>
      <word id="w36" lemma="$," features="">,</word>
      <word id="w37" lemma="men" features="">men</word>
      <word id="w38" lemma="være" features="verb">var</word>
      <word id="w39" lemma="ved" features="prep">ved</word>
      <word id="w40" lemma="bevissthet" features="subst fem ub ent">bevissthet</word>
    </set>
  - <set>
      <word id="w42" lemma="hvor" features="adv">Hvor</word>
      <word id="w43" lemma="alvorlig" features="adj ub ent">alvorlig</word>
      <word id="w44" lemma="skade" features="adj ub ent">skadd</word>
      <word id="w45" lemma="han" features="pron mask ent pers @subj">han</word>
      <word id="w46" lemma="være" features="verb">er</word>
      <word id="w47" lemma="$," features="">,</word>
      <word id="w48" lemma="være" features="verb">er</word>
      <word id="w49" lemma="fortsatt" features="adv">fortsatt</word>
      <word id="w50" lemma="uklar" features="adj ub ent">uklart</word>
    </set>
  - <set>
      <word id="w52" lemma="motorsyklist" features="subst mask be ent @subj">Motorsyklisten</word>
      <word id="w53" lemma="behandle" features="verb">behandles</word>
      <word id="w54" lemma="nå" features="adv">nå</word>
      <word id="w55" lemma="på" features="prep">på</word>
      <word id="w56" lemma="Haukeland Universitetssykehus" features="subst prop">
                        Haukeland Universitetssykehus</word>
    </set>
```

```
- <set>
    <word id="w58" lemma="ifølge" features="prep">Ifølge</word>
    <word id="w59" lemma="en" features="det mask ent kvant">en</word>
    <word id="w60" lemma="pressemelding" features="subst mask ub ent">pressemelding</word>
    <word id="w61" lemma="fra" features="prep">fra</word>
    <word id="w62" lemma="sykehus" features="subst be ent">sykehuset</word>
    <word id="w63" lemma="være" features="verb">er</word>
    <word id="w64" lemma="mann" features="subst mask be ent @subj">mannen</word>
    <word id="w65" lemma="lett" features="adj">lettere</word>
    <word id="w66" lemma="skade" features="verb">skadet</word>
    <word id="w67" lemma="$," features="">,</word>
    <word id="w68" lemma="og" features="">og</word>
    <word id="w69" lemma="tilstand" features="subst mask be ent @subj">tilstanden</word>
    <word id="w70" lemma="være" features="verb">er</word>
    <word id="w71" lemma="stabil" features="adj ub ent">stabil</word>
  </set>
- <set>
    <word id="w73" lemma="ifølge" features="prep">Ifølge</word>
    <word id="w74" lemma="politi" features="subst be ent">politiet</word>
    <word id="w75" lemma="komme" features="verb">kom</word>
    <word id="w76" lemma="han" features="pron mask ent pers @subj">han</word>
    <word id="w77" lemma="kjøre" features="adj @obj @subj">kjørende</word>
    <word id="w78" lemma="på" features="prep">på</word>
    <word id="w79" lemma="Askvegen" features="subst prop">Askvegen</word>
    <word id="w80" lemma="da" features="sbu">da</word>
    <word id="w81" lemma="han" features="pron mask ent pers @subj">han</word>
    <word id="w82" lemma="kollidere" features="verb">kolliderte</word>
    <word id="w83" lemma="med" features="prep">med</word>
    <word id="w84" lemma="en" features="det mask ent kvant">en</word>
    <word id="w85" lemma="varebil" features="subst mask ub ent">varebil</word>
    <word id="w86" lemma="i" features="prep">i</word>
    <word id="w87" lemma="kryss" features="subst be ent">krysset</word>
    <word id="w88" lemma="ved" features="prep">ved</word>
    <word id="w89" lemma="det" features="det dem ent">det</word>
    <word id="w90" lemma="gammel" features="adj be ent">gamle</word>
    <word id="w91" lemma="postkontor" features="subst be ent">postkontoret</word>
    <word id="w92" lemma="på" features="prep">på</word>
    <word id="w93" lemma="Erdal" features="subst prop">Erdal</word>
  </set>
- <set>
    <word id="w95" lemma="bil" features="subst mask be ent @subj">Bilen</word>
    <word id="w96" lemma="komme" features="verb">kom</word>
    <word id="w97" lemma="fra" features="prep">fra</word>
    <word id="w98" lemma="Grensedalen" features="subst prop">Grensedalen</word>
  </set>
- <set>
    <word id="w100" lemma="dette" features="pron ent pers @subj">Dette</word>
    <word id="w101" lemma="være" features="verb">er</word>
    <word id="w102" lemma="en" features="det ent kvant">et</word>
    <word id="w103" lemma="veldig" features="adj ub ent">veldig</word>
    <word id="w104" lemma="kinkig" features="adj ub ent">kinkig</word>
    <word id="w105" lemma="kryss" features="subst ub ent">kryss</word>
    <word id="w106" lemma="med" features="prep">med</word>
    <word id="w107" lemma="dårlig" features="adj ub ent">dårlig</word>
    <word id="w108" lemma="sikt" features="subst mask ub ent">sikt</word>
  </set>
```

```
- <set>
    <word id="w110" lemma="her" features="prep">Her</word>
    <word id="w111" lemma="burde" features="verb">bør</word>
    <word id="w112" lemma="noe" features="pron ent pers @subj">noe</word>
    <word id="w113" lemma="gjøre" features="verb">gjøres</word>
    <word id="w114" lemma="$," features="">,</word>
    <word id="w115" lemma="for" features="">for</word>
    <word id="w116" lemma="det" features="pron ent pers @subj">det</word>
    <word id="w117" lemma="være" features="verb">er</word>
    <word id="w118" lemma="fort" features="adj">fort</word>
    <word id="w119" lemma="gjøre" features="verb">gjort</word>
    <word id="w120" lemma="å" features="inf-merke @obj">å</word>
    <word id="w121" lemma="krasje" features="verb">krasje</word>
    <word id="w122" lemma="her" features="prep">her</word>
    <word id="w123" lemma="$," features="">,</word>
    <word id="w124" lemma="si" features="verb">sier</word>
    <word id="w125" lemma="politibetjent" features="subst mask ub ent @tittel">politibetjent</word>
    <word id="w126" lemma="Finn Martin Systad" features="subst prop @subj">Finn Martin Systad</word>
    <word id="w127" lemma="ved" features="prep">ved</word>
    <word id="w128" lemma="Askøy lensmannskontor" features="subst prop">Askøy lensmannskontor</word>
    <word id="w129" lemma="til" features="prep">til</word>
    <word id="w130" lemma="BT" features="subst prop">BT</word>
  </set>
- <set>
    <word id="w132" lemma="politi" features="subst be ent @subj">Politiet</word>
    <word id="w133" lemma="få" features="verb">fikk</word>
    <word id="w134" lemma="melding" features="subst fem ub ent @obj">melding</word>
    <word id="w135" lemma="om" features="prep">om</word>
    <word id="w136" lemma="ulykke" features="subst mask be ent">ulykken</word>
    <word id="w137" lemma="klokke" features="subst mask be ent">klokken</word>
    <word id="w138" lemma="08.16" features="subst">08.16</word>
  </set>
- <set>
    <word id="w140" lemma="ulykkessted" features="subst be ent @obj @subj">Ulykkesstedet</word>
    <word id="w141" lemma="ligge" features="verb">ligger</word>
    <word id="w142" lemma="på" features="prep">på</word>
    <word id="w143" lemma="riksvei" features="subst mask ub ent">riksvei</word>
    <word id="w144" lemma="563" features="det fl kvant @obj @subj">563</word>
    <word id="w145" lemma="$," features="">,</word>
    <word id="w146" lemma="ved" features="prep">ved</word>
    <word id="w147" lemma="avkjørsel" features="subst mask be ent">avkjørselen</word>
    <word id="w148" lemma="til" features="prep">til</word>
    <word id="w149" lemma="gamlevei" features="subst mask be ent">gamleveien</word>
    <word id="w150" lemma="på" features="prep">på</word>
    <word id="w151" lemma="Askøy" features="subst prop">Askøy</word>
  </set>
</newline>
```

D. Kildekode AIDaS

Dette vedlegget inneholder all kildekode som brukes for å kjøre AIDaS. Kildekoden er delt inn i modulene slik som i kapittel 7.

Kildekode til modul A Teksttagging

BOTagger

```
/**
 * @author Øyvind Vestavik og ellenroy Kommunikasjon med Olso-Bergen taggeren
 */

import java.net.*;
import java.io.*;
import java.util.*;

import javax.xml.soap.*;

public class BOTagger {
    String configFile = "config";

    //her angir dere hvor dere vil ha soap request og respons skrevet ut til
    // for inspeksjon.
    //Nå også den taggede teksten.
    private String sentmsg = "tekst/sentmsg.xml";
    private String receivedmsg = "tekst/receivedmsg.txt";
    private String taggedtext_file = "tekst/taggedtext.xml";
    private String to = "http://rigus.aksis.uib.no:8019/cgp-soap";
    private String niveau = "";
    private String input_language = "";
    private String[] featurefilterElements; //feature-filter
    private boolean totaldisambiguate; //total-disambiguate
    private boolean totaldisambiguate_isset = false; // must be changed when the
                                                     // above variable is set.
    private String encoding = "";
    private String outputformat = ""; //output-format
    private SOAPConnection con; //  Connection to send messages.
    public BOTagger(String fileToTag) {
        SOAPMessage reply = null;
        String base64String;
        configure();
        testConfiguration();
        createConnection();
        SOAPMessage request = createMessage(fileToTag); //msg er null hvis noe
                                                        // går galt....
        if (!request.equals(null)) {
            reply = sendReceive(request);
        } else {
            System.err
                    .println("Vi fikk ikke laget en request. Avslutter uten videre arbeid");
            System.exit(0);
        }
        //hvis applikasjonen fortsatt kjører har vi nå en response som er
        // skrevet til receivedmessage.txt
        //Dette er litt kode for å gjøre om fra base64encoding til vanlig
        // tekst. Skriver ut den taggede
        //teksten som en vanlig string i fila tagged text.
```

```
        try {
            //elementnavn som det skal hentes verdi for: sResponseBase64 (item
            // 0 i body)
            String base64content = reply.getSOAPBody().getChildNodes().item(0)
                    .getFirstChild().getNodeValue();
            String utf8content = base64ToUTF8Convert(base64content);
            utf8content = utf8content.replaceFirst("<newline/>",
                    "<newline> <set>");
            utf8content = utf8content.replaceAll(
                            "<word id=\"w[0-9]+\"+ lemma=\"[^w]\\.\" features=\"\">.</word>",
                            "</set> <set>");
            utf8content = utf8content.replaceAll(
                            "<word id=\"w[0-9]+\"+ lemma=\"[^w]:\" features=\"\">:</word>",
                            "</set> <set>");
            utf8content = utf8content.concat("</set></newline>");
            utf8content = utf8content.replaceAll("<set></set></newline>","</newline>");
            System.out.println(utf8content);
            //skriver den taggede teksten til fila angitt i global variabel
            // taggedtext_file
            printTaggedtext(utf8content, taggedtext_file);
        } catch (SOAPException e) {
            e.printStackTrace();
        }

    }

    /**
     * Metode som sjekker at alle nødvendige attributter er lagt til i
     * konfigurasjonsfila ved å sjekke at alle viktige globale verdier er satt,
     * dvs at hvis konfigurasjonsfila utvides med flere attributter må det
     * opprettes nye globale variabler til å hplde verdiene samt at denne
     * metoden må oppdateres I så tilfelle må også metoden configure oppdateres
     *
     */
    private void testConfiguration() {

        if (niveau.equals("")) {
            System.err.println("niveau er ikke satt i konfigurasjonsfila "
                    + configFile);
            System.exit(0);
        }
        if (featurefilterElements == null) {
            System.err.println("featurefilter er ikke satt i konfigurasjonsfila "
                            + configFile);
            System.exit(0);
        }
        if (input_language.equals("")) {
            System.err.println("language er ikke satt i konfigurasjonsfila "
                    + configFile);
            System.exit(0);
        }
        if (!totaldisambiguate_isset) {
            System.err.println("total-disambiguate er ikke satt i konfigurasjonsfila "
                            + configFile);
            System.exit(0);
        }
        if (encoding.equals("")) {
            System.err.println("encoding er ikke satt i konfigurasjonsfila "
                    + configFile);
            System.exit(0);
```

```
    }
    if (outputformat.equals("")) {
        System.err.println("output-format er ikke satt i konfigurasjonsfila "
                + configFile);
        System.exit(0);
    }

    System.out.println("configuration seems ok....");
}

/**
 * Leser konfigurasjonsfila og legger inn verdiene i globale variabler i
 * denne klassen. Hvis flere opsjoner legges til i konfigurasjonsfila må
 * metoden oppdateres for å lese de. Metoden testConfiguration må da også
 * oppdateres.
 *
 */
private void configure() {
    try {
        //setter opp det som trengs for å lese konfigurasjonsfila linje for
        // linj og splitte attributtverdipar.
        BufferedReader d = new BufferedReader(new FileReader(configFile));
        String input;
        String[] parts;

        //leser linje for linje i konfigurasjonsfila. Behandler bare linjer
        // som ikke starter
        //med # og som inneholder tekst. overfører data til globale
        // variabler og
        //skriver ut en feilmelding dersom finner en udefinert variabel i
        // konfigurasjonsfila
        while ((input = d.readLine()) != null) {
            if (!(input.equals("") || input.startsWith("#"))) {
                parts = input.split("=");

                if (parts[0].trim().equals("niveau")) {
                    niveau = parts[1].trim();
                }
                else {
                    if (parts[0].trim().equals("language")) {
                        input_language = parts[1].trim();
                    } else {
                        if (parts[0].trim().equals("feature-filter")) {
                            //Splitter verdiene basert på komma som
                            // delimiter.
                            StringTokenizer tokenizer = new StringTokenizer(
                                    parts[1], ",");
                            int tokens = tokenizer.countTokens();
                            featurefilterElements = new String[tokens];
                            int tokennr = 0;
                            while (tokenizer.hasMoreElements()) {
                                featurefilterElements[tokennr] = tokenizer
                                        .nextToken();
                                tokennr++;
                            }
                        } else {
                            if (parts[0].trim()
                                    .equals("total-disambiguate")) {
                                if (parts[1].trim().equals("true"))
                                    totaldisambiguate = true;
```

```
                            else
                                totaldisambiguate = false;
                            //denne er viktig" Brukes av
                            // testConfiguration()
                            totaldisambiguate_isset = true;
                        } else {
                            if (parts[0].trim().equals("encoding")) {
                                encoding = parts[1].trim();
                            } else {
                                if (parts[0].trim().equals(
                                                "output-format")) {
                                        outputformat = parts[1].trim();
                                }
                            }
                        }
                    }
                }
            }
        }
    } catch (Exception e) {
        System.out
                .println("Kunne ikke konfigurere applikasjonen basert på konfigurasjonsfila "
                        + configFile);
        e.printStackTrace();
    }
}

private void createConnection() {
    try {
        SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
        con = scf.createConnection();
        System.out.println("SOAP connection opened");
    } catch (Exception e) {
        System.err.println("Unable to open a SOAPConnection" + e);
        System.exit(0);
    }
}

private SOAPMessage createMessage(String fileToTag) {

    System.out.println("Lager en SOAP request");
    SOAPMessage msg = null;

    //Leser inn tekst fra fila som skal tagges til et String-objekt
    String input, textToTag;
    FilLeser fl = new FilLeser(fileToTag);
    input = "";

    //personverb
    while (fl.klar()) {

        String linje = fl.lesString() + " ";
        if (linje.startsWith("-")) {
            linje = linje.replaceFirst("-", "");
        }

        if (!linje.endsWith(". ")) {
            if (!linje.endsWith(": ")) {
                linje = linje.concat(". ");
```

```
            }
        }
        input += linje;
    }
    input = input.replaceAll("\"", "");
    String untagged = input;
    try {
        // Create a message factory.
        MessageFactory mf = MessageFactory.newInstance();

        // Create a message from the message factory.
        msg = mf.createMessage();

        /*
         * Her er oppdateringen fra Paul Meurer satt inn. Setter http-header
         * i message
         */

        MimeHeaders hd = msg.getMimeHeaders();
        hd.addHeader("SOAPAction", "ACLSOAP");

        // Message creation takes care of creating the SOAPPart - a
        // required part of the message as per the SOAP 1.1
        // specification.
        SOAPPart sp = msg.getSOAPPart();

        // Retrieve the envelope from the soap part to start buildin
        // the soap message.
        SOAPEnvelope envelope = sp.getEnvelope();

        //setting encoding style and namespaces for the envelope
        envelope
                .setEncodingStyle("http://schemas.xmlsoap.org/soap/encoding/");
        envelope.addNamespaceDeclaration("xsd",
                "http://www.w3.org/2001/XMLSchema");
        envelope.addNamespaceDeclaration("xsi",
                "http://www.w3.org/2001/XMLSchema-instance");
        envelope.addNamespaceDeclaration("SOAP-ENC",
                "http://schemas.xmlsoap.org/soap/encoding");

        //Fjerner header siden header ikke var del av eksempelet vi fikk
        // Create a soap header from the envelope.
        //SOAPHeader hdr = envelope.getHeader();

        envelope.removeChild(envelope.getHeader());

        // Create a soap body from the envelope.
        SOAPBody bdy = envelope.getBody();

        //lager element-strukturen for body og legegr til body av SOAP
        // elementet.
        SOAPBodyElement tagTextBase64 = bdy.addBodyElement(envelope
                .createName("tagTextBase64", "cgps",
                        "http://www.aksis.uib.no/cgp"));

        SOAPElement language = tagTextBase64.addChildElement(envelope
                .createName("language"));
        language.setAttribute("xsi:type", "xsd:string");
        language.addTextNode(input_language);
```

```
        SOAPElement nivaa = tagTextBase64.addChildElement(envelope
                .createName("niveau"));
        nivaa.setAttribute("xsi:type", "xsd:string");
        nivaa.addTextNode(niveau);

        SOAPElement tekst = tagTextBase64.addChildElement(envelope
                .createName("text"));
        tekst.setAttribute("xsi:type", "xsd:string");
        tekst.addTextNode(untagged);

        // Dette var det jeg la til for å få brukt norske bokstaver
        SOAPElement tekstformat = tagTextBase64.addChildElement(envelope
                .createName("encoding"));
        tekstformat.setAttribute("xsi:type", "xsd:string");
        tekstformat.addTextNode(encoding);

        SOAPElement outformat = tagTextBase64.addChildElement(envelope
                .createName("output-format"));
        outformat.setAttribute("xsi:type", "xsd:string");
        outformat.addTextNode(outputformat);

        SOAPElement featurefilter = tagTextBase64.addChildElement(envelope
                .createName("feature-filter"));

        for (int i = 0; i < featurefilterElements.length; i++) {
            SOAPElement f = featurefilter.addChildElement(envelope
                    .createName("f"));
            f.setAttribute("xsi:type", "xsd:string");
            f.addTextNode(featurefilterElements[i]);
        }

        SOAPElement disambiguation = tagTextBase64.addChildElement(envelope
                .createName("total-disambiguate"));
        disambiguation.setAttribute("xsi:type", "xsd:boolean");
        String booleanValue = "";
        if (totaldisambiguate) {
            booleanValue = "true";
        } else {
            booleanValue = "false";
            disambiguation.addTextNode(booleanValue);
        }
        FileOutputStream sentFile = new FileOutputStream(sentmsg);
        msg.writeTo(sentFile);
        sentFile.close();

    }

    catch (Throwable e) {
        System.err.println("Error in constructing message "
                + e.getMessage());
    }
    return msg;
}

/**
 *
 * @param msg
 *          Meldingen som skal skrives ut. Dette kan være request som
 *          sendt til Bergen eller responsen som mottatt fra bergen..
 * @param filePath
```

```
     *          Fila det skal skrives til
     */
    private void printSOAPMessage(SOAPMessage msg, String filePath) {
        System.out.println("Skriver kopi av request til " + filePath);
        try {
            FileOutputStream file = new FileOutputStream(filePath);
            msg.writeTo(file);
            file.close();
        } catch (Exception e) {
            System.err.println("Kunne ikke skrive SOAP request til fila"
                    + sentmsg);
            e.printStackTrace();
        }
    }

    private SOAPMessage sendReceive(SOAPMessage msg) {

        URL urlEndpoint = null;
        SOAPMessage reply = null;

        try {
            urlEndpoint = new URL(to);
        } catch (MalformedURLException e) {
            System.out.println("Ubrukelig URL. Avslutter.");
            e.printStackTrace();
            System.exit(0);
        }
        //Send the message to the provider using the connection.
        System.out.println("Sender request til Bergen");
        try {
            reply = con.call(msg, urlEndpoint);
        } catch (SOAPException e) {
            System.out
                    .println("Noe gikk galt med forespørselen til Bergen. Se under for detaljer. Avslutter
her...");
            e.printStackTrace();
            System.exit(0);
        }

        if (reply != null) {
            System.out.println("Svar mottatt fra Bergen. Skriver svaret til "
                    + receivedmsg);
            printSOAPMessage(reply, receivedmsg);

        } else {
            System.err.println("No reply fra Bergen. Avslutter");
        }

        return reply;

    }

    /**
     * Utility method for private String base64ToUTF8Convert(String source)
     *
     * @param source
     * @param target
     * @param targetIndex
     */
    private void convert4To3(byte[] source, byte[] target, int targetIndex) {
```

```
        target[targetIndex] = (byte) ((source[0] << 2) | (source[1] >>> 4));
        target[targetIndex + 1] = (byte) (((source[1] & 0x0f) << 4) | (source[2] >>> 2));
        target[targetIndex + 2] = (byte) (((source[2] & 0x03) << 6) | (source[3]));
    }

    /**
     *
     * Denne koden er hentet fra nettet.
     * http://www.koders.com/java/fid2B76F86B8F8FF984EFE3C04AC6DE198E40D8EF41.aspx
     *
     * @param source
     *            Stringen som skal konverteres fra base64 til utf-8
     * @return Input convertert til UTF-8
     */
    private String base64ToUTF8Convert(String source) {
        String encodingChar =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
        if (source.length() % 4 != 0)
            throw new RuntimeException(
                    "valid Base64 codes have a multiple of 4 characters");
        int numGroups = source.length() / 4;
        int numExtraBytes = source.endsWith("==") ? 2
                : (source.endsWith("=") ? 1 : 0);
        byte[] targetBytes = new byte[3 * numGroups];
        byte[] sourceBytes = new byte[4];
        for (int group = 0; group < numGroups; group++) {
            for (int i = 0; i < sourceBytes.length; i++) {
                sourceBytes[i] = (byte) Math.max(0, encodingChar.indexOf(source
                        .charAt(4 * group + i)));
            }
            convert4To3(sourceBytes, targetBytes, group * 3);
        }
        return new String(targetBytes, 0, targetBytes.length - numExtraBytes);
    }

    private void printTaggedtext(String textToPrint, String filePath) {
        System.out.println("Skriver kopi av tagget tekst til " + filePath);
        try {
            FileWriter out = new FileWriter(filePath);
            for (int i = 0; i < textToPrint.length(); i++) {
                out.write(textToPrint, i, 1);
            }
            out.close();
        } catch (Exception e) {
            System.err.println("Kunne ikke skrive SOAP request til fila"
                    + sentmsg);
            e.printStackTrace();
        }
    }
    public static void main(String[] args) {
        if (args.length != 1) {
            System.out.println("USAGE: java BOTagger fileToTag");
        } else {
            BOTagger tagger = new BOTagger(args[0]);
        }
    }
}
```

Kildekode til modul B Preprossesering

Kontroller

```
import java.io.File;
import java.util.ArrayList;


/**
 * @author ellenroy
 *
 * Denne klassen inneholder main metoden og starter klassifiseringen
 */
public class Kontroller {

    public static void main(String[] args) {
        Kontroller k = new Kontroller();
        //preprosesserer filen
        Preprosesserer pp = new Preprosesserer(new File("xml/eval2.xml"));
        //henter egennavnGjenkjenner objektet
        EgennavnGjenkjenner eg = pp.getEg();
        //genererer reglene til B.1
        ArrayList regler = k.genererRegler1();
        //første fase, B.1
        Klassifisering kl = new Klassifisering();
        kl.klassifiser(regler, eg);
        //tredje fase, B.3
        System.out.println("ANDRE GJENNOMKJØRING: B.3");
        System.out.println("");
        //genererer reglene til B.3
        ArrayList regler2 = k.genererRegeler2();
        kl.klassifiser(regler2, eg);
    }

    /**
     * Lager reglene til den første fasen, B.1
     * @return listen med regler
     */
    public ArrayList genererRegler1() {
        ArrayList regler = new ArrayList();

        Regel storB = new StorBokstavRegel();
        regler.add(storB);

        Regel tittelRegel = new TittelRegel();
        regler.add(tittelRegel);

        Regel sierRegel = new SierRegel();
        regler.add(sierRegel);

        Regel stedSubRegel = new StedSubRegel();
        regler.add(stedSubRegel);

        Regel subRegel = new SubRegel();
        regler.add(subRegel);

        Regel forkRegel = new ForkRegel();
        regler.add(forkRegel);
```

```
        Regel sammenMed = new SammenMedRegel();
        regler.add(sammenMed);

        Regel appRegel = new ApposisjonsRegel();
        regler.add(appRegel);

        Regel kommaRegel = new KommaRegel();
        regler.add(kommaRegel);

        Regel konjRegel = new KonjRegel();
        regler.add(konjRegel);

        Regel verbRegel = new VerbRegel();
        regler.add(verbRegel);

        return regler;

    }
    /**
     * Lager reglene til den tredje fasen, B.3
     * @return
     */
    private ArrayList genererRegeler2() {
        ArrayList regler = new ArrayList();

        Regel persRegel = new PersRegel();
        regler.add(persRegel);

        Regel yrkeRegel = new YrkeRegel();
        regler.add(yrkeRegel);

        Regel sierRegel = new SierRegel();
        regler.add(sierRegel);

        Regel prepRegel = new PrepRegelListe();
        regler.add(prepRegel);

        Regel kommaRegel = new KommaRegel2();
        regler.add(kommaRegel);

        Regel konjRegel = new KonjRegel2();
        regler.add(konjRegel);

        Regel preposisjon = new PrepRegel();
        regler.add(preposisjon);

        //Liste regel blir ikke lagt til her,
        //kjøres helt på slutten
        return regler;
    }
}
```

Preprosseserer

```
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.jdom.Document;
```

```
import org.jdom.Element;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;

/**
 * @author ellenroy
 *
 * Denne klassen tar inn den taggede teksten som en xml fil og lese innholdet
 * Alle ord blir lagret som et Ord objekt og lagt inn i en ArrayList som inneholder alle ordene i en
setning
 * Deretter blir disse ArrayListene lagt inn i en ny ArrayList. På denne måten tar man vare på den
relevante
 * infoen som man fikk fra Oslo Bergen taggeren.
 *
 */
public class Preprosesserer{
    ArrayList setninger;
    EgennavnGjenkjenner eg;

    /**
     * Konstruktor
     * @param fil
     */
    public Preprosesserer(File fil){
        readeXML(fil);
        //skrivUt();
        eg = new EgennavnGjenkjenner(setninger);
    }

    /**
     * tar inn XML-fila og leser innholdet i den.
     * @param fil
     */
    public void readeXML(File fil) {
        SAXBuilder myBuilder = new SAXBuilder();
        Document myDocument = null;
        setninger = new ArrayList();
        try {
            myDocument = myBuilder.build(fil);
        } catch (JDOMException jde) {
            jde.printStackTrace();

        } catch (IOException e) {

            e.printStackTrace();
        }
        Element rot = myDocument.getRootElement();
        List setListe = rot.getChildren("set");
        for (Iterator i =  setListe.iterator(); i.hasNext();) {
            ArrayList ordISetning = new ArrayList();
            List ordListe = ((Element)i.next()).getChildren("word");
            for (Iterator iter = ordListe.iterator(); iter.hasNext();) {
                Element ordElement = (Element) iter.next();
                Ord nyttOrd = new Ord(ordElement.getText(),ordElement.getAttributeValue("lemma"),
                        ordElement.getAttributeValue("features"));
                ordISetning.add(nyttOrd);
            }

            setninger.add(ordISetning);
        }
```

```
    }
    /**
     * Skriver ut ordene i teksten.
     */
    private void skrivUt() {
        for (int i = 0; i < setninger.size(); i++) {
            ArrayList temp = (ArrayList)setninger.get(i);
            System.out.println("Ny setning: ");
            for (int j = 0; j <temp.size(); j++) {
                System.out.print(((Ord)temp.get(j)).getOrd() + " ");
            }
            System.out.println("");
        }

    }

    public EgennavnGjenkjenner getEg(){
        return eg;
    }
}
```

EgennavnGjenkjenner

```
import java.util.ArrayList;

/**
 * @author ellenroy Klassen finner egennavnene i inputteskten
 *
 */
public class EgennavnGjenkjenner {

    private ArrayList tekst;

    private ArrayList forsteOrdISetning;

    private ArrayList egennavnListe;

    /**
     * Kontruktoren
     *
     * @param tekst
     */
    public EgennavnGjenkjenner(ArrayList tekst) {
        this.tekst = tekst;
        forsteOrdISetning = new ArrayList();
        egennavnListe = new ArrayList();
        finnEgennavn();
        skrivUt();
    }

    /**
     * finner egennavnene
     */
    public void finnEgennavn() {
        for (int i = 0; i < tekst.size(); i++) {
            // første listen som inneholder lister med setninger
            ArrayList setning = (ArrayList) tekst.get(i);
            if (setning.size() == 1) {
                forsteOrdISetning.add((setning.get(0)));
```

```
} else {
    for (int j = 0; j < setning.size(); j++) {
        //liste med ord i setninger
        String ord = ((Ord) (setning.get(j))).getOrd();
        char temp = ord.charAt(0);
        // slår ikke sammen sekvenser her
        if (j == 0 && Character.isUpperCase(temp)) {
            forsteOrdISetning.add((setning.get(j)));
        } else if (Character.isUpperCase(temp)) {
            //alle andre ord med stor forbokstav
            // Oslo-Bergen taggeren har lagt sammen sekvenser
            //men må ta høyde for tall
            String neste = "";
            String neste2 = "";
            if (j < setning.size() - 1) {
                neste = ((Ord) (setning.get(j + 1))).getOrd();
            }
            if (j < setning.size() - 2) {
                neste2 = ((Ord) (setning.get(j + 2))).getOrd();
            }
            if (sjekkOmTall(neste)) {
                ord = ord.concat(" " + neste);
                String lemma0 = ((Ord) (setning.get(j))).getLemma();
                String lemma1 = ((Ord) (setning.get(j + 1)))
                        .getLemma();
                String lemma = lemma0.concat(" " + lemma1);
                String feature0 = ((Ord) (setning.get(j)))
                        .getFeaturesString();
                String feature1 = ((Ord) (setning.get(j + 1)))
                        .getFeaturesString();
                String feature = feature0.concat(" " + feature1);
                Ord nyttOrd = new Ord(ord, lemma, feature);
                egennavnListe.add(new Egennavn(nyttOrd, j, i));
                setning.add(j, nyttOrd);
                setning.remove(j + 1);
                setning.remove(j + 1);
                tekst.add(i, setning);
                tekst.remove(i + 1);
            }
            //må sjekke om det finnes en org som ikke er tatt opp
            // av Oslo-Bergen taggeren
            else if (sjekkOmOrg(neste, neste2)) {
                ord = ord.concat(" " + neste + " " + neste2);
                String lemma0 = ((Ord) (setning.get(j))).getLemma();
                String lemma1 = ((Ord) (setning.get(j + 1)))
                        .getLemma();
                String lemma2 = ((Ord) (setning.get(j + 2)))
                        .getLemma();
                String lemma = lemma0.concat(" " + lemma1 + " "
                        + lemma2);
                String feature0 = ((Ord) (setning.get(j)))
                        .getFeaturesString();
                String feature1 = ((Ord) (setning.get(j + 1)))
                        .getFeaturesString();
                String feature2 = ((Ord) (setning.get(j + 2)))
                        .getFeaturesString();
                String feature = feature0.concat(" " + feature1
                        + " " + feature2);
                Ord nyttOrd = new Ord(ord, lemma, feature);
                egennavnListe.add(new Egennavn(nyttOrd, j, i));
```

```
                    setning.add(j, nyttOrd);
                    setning.remove(j + 1);
                    setning.remove(j + 1);
                    setning.remove(j + 1);
                    tekst.add(i, setning);
                    tekst.remove(i + 1);
                } else {
                    egennavnListe.add(new Egennavn(((Ord) setning
                            .get(j)), j, i));
                }
            }
        }
    }

}

/**
 * @param neste
 * @param neste2
 * @return
 */
private boolean sjekkOmOrg(String neste, String neste2) {
    char temp;
    if (neste2.length() > 0) {
        temp = neste2.charAt(0);
        if (neste.equals("for") && Character.isUpperCase(temp)) {
            return true;
        }
    }

    return false;
}

/**
 * sjekker om neste ord er et tall
 *
 * @param ord
 * @return
 */
public boolean sjekkOmTall(String ord) {
    try {
        Integer i = Integer.valueOf(ord);
        return true;
    } catch (NumberFormatException e) {
        return false;
    }
}

/**
 *
 */
public void skrivUt() {
    System.out.println("Dette er først ord i setning");
    for (int i = 0; i < forsteOrdISetning.size(); i++) {
        System.out.println(((Ord) (forsteOrdISetning.get(i))).getOrd());
    }
    System.out.println("Dette er egennavn");
    for (int i = 0; i < egennavnListe.size(); i++) {
        System.out.print((((Egennavn) (egennavnListe.get(i))).getOrd())
```

```
                .getOrd());
            System.out
                .print(" " + ((Egennavn) (egennavnListe.get(i))).getPos());
            System.out.println(" "
                + ((Egennavn) (egennavnListe.get(i))).getSetningNr());
        }
        System.out.println("");
    }

    /**
     * @return
     */
    public ArrayList getEgennavnList() {
        return egennavnListe;
    }

    /**
     * @return
     */
    public ArrayList getTekst() {
        return tekst;
    }

    /**
     * @return
     */
    public ArrayList getForsteOrd() {
        return forsteOrdISetning;
    }
}
```

Ord

```
import java.util.ArrayList;
import java.util.StringTokenizer;

/**
 * @author ellenroy
 *
 * Hjelpeklasse
 */
public class Ord {

    private String ord;
    private String lemma;
    private String features;
    private ArrayList featuresListe;

    public Ord(String ord, String lemma, String features) {
        this.ord = ord;
        this.lemma = lemma;
        this.features = features;
        makeFeatureList();

    }
    /**
     * aksessor metode for å hente ordet
     * @return
     */
    public String getOrd(){
```

```
        return ord;
    }
    /**
     * akessor metode for å hente lemma
     * @return
     */
    public String getLemma(){
        return lemma;
    }

    /**
     * aksessor metode for å hente streng med feature
     * @return
     */
    public String getFeaturesString(){
        return features;
    }

    /**
     * metode som deler opp strengen med features og
     * legger hver feature som et element i en liste
     *
     */
    public void makeFeatureList(){
        featuresListe = new ArrayList();
        StringTokenizer st = new StringTokenizer(features);
         while (st.hasMoreTokens()) {
            featuresListe.add(st.nextToken());
         }
    }

    /**
     * aksessormetode for å hente liste med feature
     * @return
     */
    public ArrayList getFeatures(){
        return featuresListe;
    }

    /**
     * metode som kalles når et egennavn har fått tildelt en type.
     * hvis et egennavn er av typen sted, skal featuren skrives med store forbokstaver,(STED eller
ANNET)
     * dette for å skille mellom mine features og oslo-bergen taggeren sine
     * @param nyFeature
     */
    public void addFeature(String nyFeature){
        featuresListe.add(nyFeature);
    }

    public String toString(){
        return "Ord " + ord +  " Lemma " + lemma + " features " + features;
    }
}
```

Egennavn

```
/**
 * @author ellenroy
 *
 * Hjelpeklasse for å ta var på informasjon om egennavn
 */
public class Egennavn {

    private String navn;
    private int pos;
    private int setningNr;
    private Ord ord;

    public Egennavn(String navn, int pos, int setningNr) {
        this.navn = navn;
        this.pos = pos;
        this.setningNr = setningNr;

    }

    public Egennavn(Ord ord, int pos, int setningNr) {
        this.ord = ord;
        this.pos = pos;
        this.navn = ord.getOrd();
        this.setningNr = setningNr;

    }

    public String getNavn() {
        return navn;
    }

    public Ord getOrd(){
        return ord;
    }

    public int getPos() {
        return pos;
    }

    public int getSetningNr() {
        return setningNr;
    }

    public void setPos(int nyPos) {
        pos = nyPos;
    }

    public void endreNavn (String nyttNavn){
        navn = nyttNavn;
    }

}
```

Kildekode til modul C Klassifisering

Klassifisering

```
import java.util.ArrayList;
import java.util.StringTokenizer;


/**
 * @author ellenroy
 * Klassen styrer klassifiseringen.
 */
public class Klassifisering {
    private boolean fase2 = false;

    public Klassifisering() {
    }
    /**
     * Skriver ut lister med steder, annet med forekomster
     * @param stedForekomster
     * @param annetForekomster
     * @param egennavnListe
     */
    private void skrivUt(ArrayList stedForekomster, ArrayList annetForekomster, ArrayList
egennavnListe) {
        System.out.println("Dette er stedslisten med antall forekomster: ");
        for (int l = 0; l < stedForekomster.size(); l++) {
            System.out.println(((Forekomst) stedForekomster.get(l)).toString());
        }
        System.out.println("");

        System.out.println("Dette er annetlisten: ");
        for (int l = 0; l < annetForekomster.size(); l++) {
            System.out.println(((Forekomst) annetForekomster.get(l)).toString());
        }
        System.out.println("");
        System.out.println("Dette er egennavnlisten: ");
        for (int l = 0; l < egennavnListe.size(); l++) {
            System.out.println(((Egennavn)egennavnListe.get(l)).getNavn()+
((Egennavn)egennavnListe.get(l)).getSetningNr());
        }
        System.out.println("");

    }

    /**
     * metoden teller antall forekomster av et navn i en liste
     * @param liste
     * @return
     */
    public ArrayList tellForekomster(ArrayList liste) {
        ArrayList navneListe = new ArrayList();
        for (int j = 0; j < liste.size(); j++) {
            navneListe.add(liste.get(j));
        }
        ArrayList forekomster = new ArrayList();
        System.out.println("");
        for (int l = 0; l < liste.size(); l++) {
            String navn = ((Egennavn) liste.get(l)).getNavn();
```

```
            int antall = 0;
            int m = 0;
            int teller1 = navneListe.size();
            while (teller1>0 && m<navneListe.size()) {
                String navn2 = ((Egennavn) navneListe.get(m)).getNavn();
                if (navn.equals(navn2)) {
                    antall++;
                    navneListe.remove(m);
                }
                else {
                    m++;
                }
                teller1 --;
            }
            if (antall > 0) {
                forekomster.add(new Forekomst(navn, antall));
            }
        }
        return forekomster;

    }

    /**
     * Denne metoden starter selve klassifiseringen, kaller også delvisordning
     * @param regler
     * @param eg
     */
    public void klassifiser(ArrayList regler, EgennavnGjenkjenner eg){
        // henter listene som inneholder egennavn og hele teksten
        ArrayList egennavnListe = eg.getEgennavnList();
        ArrayList tekst = eg.getTekst();
        ArrayList forsteOrd = eg.getForsteOrd();
        ArrayList stedsnavn = new ArrayList();
        ArrayList annet = new ArrayList();
        ArrayList fase = new ArrayList();
        ArrayList storBokstav = new ArrayList();

        int teller = egennavnListe.size();
        System.out.println("TELLER ER "+ teller );
        int i=0;
        while(!egennavnListe.isEmpty() && teller > 0){

            System.out.println("Egennavn " +((Egennavn)egennavnListe.get(i)).getNavn());
            boolean ferdig = false;
            int j = 0;
            Egennavn navn = null;
            while(ferdig == false && j<regler.size()){
                int setnr = ((Egennavn)(egennavnListe.get(i))).getSetningNr();
                ferdig
=((Regel)regler.get(j)).analyserSetning((Egennavn)egennavnListe.get(i),(ArrayList)tekst.get(setnr));
                stedsnavn = ((Regel)regler.get(j)).getStedsnavn();
                annet = ((Regel)regler.get(j)).getAnnet();
                storBokstav = ((Regel)regler.get(j)).getStorBokstav();
                j++;
            }
            if(ferdig){
                //fjerner egennavnet
                System.out.println("Egennavnet ble klassifisert "+(j-1));
                egennavnListe.remove(i);
            }
```

```
            else{
                i++;
            }
            teller--;
        }
        if(fase2 == true){
            Regel listeRegel = new ListeRegel();
            boolean ok = false;
            int j = 0;
            int teller2 = egennavnListe.size();
            while (!egennavnListe.isEmpty()&& teller2>0) {
                ok = listeRegel.analyserSetning((Egennavn)egennavnListe.get(j), null);
                if(ok){
                    egennavnListe.remove(j);
                }
                else{
                    j ++;
                }
                teller2 --;

            }
        }
        delvisOrdning(egennavnListe, forsteOrd, annet, stedsnavn, storBokstav);
    }

    /**
     * Metoden starter fase C.2 og C.4, dvs finn forekomster og delvisordning
     * @param egennavnListe
     * @param forsteOrd
     * @param annet
     * @param stedsnavn
     */
    public void delvisOrdning( ArrayList egennavnListe, ArrayList forsteOrd, ArrayList annet,
ArrayList stedsnavn, ArrayList storBokstav){

        DelvisOrdning dO = new DelvisOrdning(egennavnListe, forsteOrd);
        if(!fase2){
            ArrayList nyeAnnet = dO.finnDOForekomster1(annet);
            for (int l = 0; l < nyeAnnet.size(); l++) {
                ((Egennavn)nyeAnnet.get(l)).getOrd().addFeature("ANNET");
                annet.add(nyeAnnet.get(l));
            }
            ArrayList nyeSteder = dO.finnDOForekomster1(stedsnavn);
            for (int l = 0; l < nyeSteder.size(); l++) {
                ((Egennavn)nyeSteder.get(l)).getOrd().addFeature("STED");
                stedsnavn.add(nyeSteder.get(l));
            }
        }
        if(fase2){
            ArrayList nyeAnnet = dO.finnDO(annet);
            for (int l = 0; l < nyeAnnet.size(); l++) {
                ((Egennavn)nyeAnnet.get(l)).getOrd().addFeature("ANNET");
                annet.add(nyeAnnet.get(l));
            }
            ArrayList nyeSteder = dO.finnDO(stedsnavn);
            for (int l = 0; l < nyeSteder.size(); l++) {
                ((Egennavn)nyeSteder.get(l)).getOrd().addFeature("STED");
                stedsnavn.add(nyeSteder.get(l));
            }
```

```
            ArrayList navn = new ArrayList();
            ArrayList sted = lesFraFil("tekst/sted.txt", "#");
            ArrayList ord = dO.getForsteOrd();
            for (int j = 0; j < ord.size(); j++) {
               //sjekker om første ord er et stedsnavn
               for (int i = 0; i < sted.size(); i++) {
                  if(((Ord)ord.get(j)).getOrd().equals((String)sted.get(i))){
                     stedsnavn.add(new Egennavn(((Ord)ord.get(j)),0,j));
                  }
               }
            }

            //håndterer ord som er skrevet med store bokstaver
            if(!storBokstav.isEmpty()){
               for (int i = 0; i < storBokstav.size(); i++) {
                  String del = ((Egennavn)storBokstav.get(i)).getNavn().substring(1);
                  del = del.toLowerCase();
                  String nyttnavn = (((Egennavn)storBokstav.get(i)).getNavn()).substring(0,1).concat(del);
                  boolean funnet = false;
                  for (int j = 0; j < stedsnavn.size(); j++) {
                     if(nyttnavn.equals(((Egennavn)stedsnavn.get(j)).getNavn())){
                        stedsnavn.add(storBokstav.get(i));
                        j = stedsnavn.size();
                     }

                  }
               }
            }
            ArrayList stedForekomster = tellForekomster(stedsnavn);
            ArrayList annetForekomster = tellForekomster(annet);
            skrivUt(stedForekomster, annetForekomster, egennavnListe);

         }
         fase2 = true;

      }

      public ArrayList lesFraFil(String filnavn, String skilletegn) {
         ArrayList ord = new ArrayList();
         FilLeser  fl = new FilLeser(filnavn);
         String tekst ="";
         while(fl.klar()){
            tekst += fl.lesString() + " ";
         }
         StringTokenizer st = new StringTokenizer(tekst,skilletegn);
         while (st.hasMoreTokens()) {
            ord.add(st.nextToken());
         }
         return ord;
      }
}
```

DelvisOrdning

```
import java.util.ArrayList;
import java.util.StringTokenizer;

/**
 * @author ellenroy
```

```
 * Denne klasse gjør store deler av operasjonen til fase C.2 og C.4
 */
public class DelvisOrdning {

    ArrayList egennavnListe, forsteOrd;

    /**
     * Konstruktoren
     * @param egennavnListe
     * @param forsteOrd
     */
    public DelvisOrdning(ArrayList egennavnListe, ArrayList forsteOrd) {
        this.egennavnListe = egennavnListe;
        this.forsteOrd = forsteOrd;
    }

    /**
     * Denne metoden skal finne alle partisjoner av et egennavn
     * @param liste
     */
    public ArrayList finnDO(ArrayList liste){
        ArrayList ordning = new ArrayList();

        //deler opp navnet og legger det i listen ordninger
        for (int i = 0; i < liste.size(); i++) {
            ArrayList ordninger = new ArrayList();
            String navn = ((Egennavn) liste.get(i)).getNavn();
            StringTokenizer st = new StringTokenizer(navn);
            while (st.hasMoreTokens()) {
                String neste = st.nextToken();
                ordninger.add(neste);
            }

            int tall = 0;
            for (int j = 0; j < ordninger.size(); j++) {
                int antall = 0;
                String ord = "";
                for (int k = tall; k < ordninger.size(); k++) {
                    ord = leggSammenOrd(ordninger, ord, antall, tall);
                    ordning.add(ord);
                    antall++;
                    ord = "";
                }
                tall++;
                antall++;
            }
        }
        ArrayList retur = finnDOForekomster(ordning);

        return retur;
    }

    /**
     * Sjekker om det finnes forekomster av egennavnene, dvs fase C.2
     * @param ordning
     * @return
     */
    public ArrayList finnDOForekomster1(ArrayList ordning) {
        ArrayList retur = new ArrayList();
```

```
        for (int i = 0; i < ordning.size(); i++) {
            boolean ok = false;
            int l = 0;
            int teller = egennavnListe.size();
            while (!egennavnListe.isEmpty()&& teller>0) {

    if(((Egennavn)ordning.get(i)).getNavn().equals(((Egennavn)egennavnListe.get(l)).getNavn()) ||

    (((Egennavn)ordning.get(i)).getNavn()+"s").equals(((Egennavn)egennavnListe.get(l)).getNavn())){
                        //tar høyde for genetiv
                    System.out.println("de er like" +((Egennavn)ordning.get(i)).getNavn()+ " " +
((Egennavn)egennavnListe.get(l)).getNavn() );
                    retur.add(((Egennavn)egennavnListe.remove(l)));
                }
                else{
                    l ++;
                }
                teller --;

            }


            for (int j = 0; j < forsteOrd.size(); j++) {
                if(ordning.get(i).equals(((Ord)forsteOrd.get(j)).getOrd())){
                    retur.add(new Egennavn(((Ord)forsteOrd.remove(j)),0,j));
                    forsteOrd.add(j,new Ord("", "", ""));
                }
            }
        }
        return retur;
    }
    /**
     * Sjekker om det finnes forekomster av delvisordning dvs deler av fase C.4
     * @param ordning
     * @return
     */
    public ArrayList finnDOForekomster(ArrayList ordning) {
        ArrayList retur = new ArrayList();

        for (int i = 0; i < ordning.size(); i++) {
            boolean ok = false;
            int l = 0;
            int teller = egennavnListe.size();
            while (!egennavnListe.isEmpty()&& teller>0) {
                if(ordning.get(i).equals(((Egennavn)egennavnListe.get(l)).getNavn()) ||
                        (ordning.get(i)+"s").equals(((Egennavn)egennavnListe.get(l)).getNavn())){
                        //tar høyde for genetiv
                    System.out.println("de er like" +ordning.get(i)+ " " +
((Egennavn)egennavnListe.get(l)).getNavn() );
                    retur.add(((Egennavn)egennavnListe.remove(l)));
                }
                else{
                    l ++;
                }
                teller --;

            }
            for (int j = 0; j < forsteOrd.size(); j++) {
                if(ordning.get(i).equals(((Ord)forsteOrd.get(j)).getOrd())){
                    retur.add(new Egennavn(((Ord)forsteOrd.remove(j)),0,j));
```

```
                forsteOrd.add(j,new Ord("", "", ""));
            }
        }
    }
    return retur;
}


/**
 * en liten hjelpemetode som legger fjerner ett og et ord fra første ord i
 * setning
 * @param ordListe
 * @param ord
 * @param teller
 * @param tall
 * @return
 */
public String leggSammenOrd(ArrayList ordListe, String ord, int teller, int tall){
    for (int j = tall; j < ordListe.size()-teller; j++) {
        if(j == ordListe.size()-teller-1){
            ord = ord.concat(((String)ordListe.get(j)));
        }
        else{
        ord = ord.concat(((String)ordListe.get(j))+ " ");
        }
    }
    return ord;
}

/**
 * Leser fra fil
 * @param filnavn
 * @param skilletegn
 * @return
 */
public ArrayList lesFraFil(String filnavn, String skilletegn) {
    ArrayList ord = new ArrayList();
    FilLeser  fl = new FilLeser(filnavn);
    String tekst ="";
    while(fl.klar()){
        tekst += fl.lesString() + " ";
    }
    StringTokenizer st = null;
    if(skilletegn.equals("#")){
        st = new StringTokenizer(tekst,skilletegn);
    }
    else{
        st = new StringTokenizer(tekst);
    }
    while (st.hasMoreTokens()) {
    ord.add(st.nextToken());
    }
    return ord;
}

public ArrayList getForsteOrd(){
    return forsteOrd;
}
}
```

Regel

```
import java.util.ArrayList;
import java.util.StringTokenizer;


/**
 * @author ellenroy
 * Superklassen til alle reglene. Den er abstrakt slik at det er mulig å bruke polimorfi
 */
public abstract class Regel {

    protected static ArrayList stedsnavn = new ArrayList();
    protected static ArrayList annet = new ArrayList();
    protected static ArrayList storBokstav = new ArrayList();
    protected Egennavn eg;

    public Regel(){
    }

    /**
     * Alle subklasser av denne klassen må ha en metode som ser på egennavnet
     * og bestemme hvilken type ordet er
     */
    public abstract boolean analyserSetning(Egennavn egennavn, ArrayList setning);

    /**
     * Metode som skriver ut listene med navn
     */
    public void skrivUt() {
        System.out.println("Dette er stedslisten: ");
        for (int i = 0; i < stedsnavn.size(); i++) {
            System.out.println(((Egennavn)stedsnavn.get(i)).getNavn());
        }

        System.out.println("Dette er annetlisten: ");
        for (int i = 0; i < annet.size(); i++) {
            System.out.println(((Egennavn)annet.get(i)).getNavn());
        }

    }

    public ArrayList getStedsnavn() {
        return stedsnavn;

    }

    public ArrayList getStorBokstav() {
        return storBokstav;

    }

    public ArrayList getAnnet() {
        return annet;
    }

    public Egennavn getEgennavn(){
            return eg;
    }
    /**
```

```
     * Metode for å lese fra fil
     * @param filnavn
     * @param skilletegn
     * @return
     */
    public ArrayList lesFraFil(String filnavn, String skilletegn) {
        ArrayList ord = new ArrayList();
        FilLeser  fl = new FilLeser(filnavn);
        String tekst ="";
        while(fl.klar()){
            tekst += fl.lesString() + " ";
        }
        StringTokenizer st = null;
        if(skilletegn.equals("#")){
            st = new StringTokenizer(tekst,skilletegn);
        }
        else{
            st = new StringTokenizer(tekst);
        }
        while (st.hasMoreTokens()) {
        ord.add(st.nextToken());
        }
        return ord;
    }

}
```

FilLeser

```
import java.io.*;
/**
 * @author ellenroy
 * Klassen inneholder metoder for å lese fra fil
 */
public class FilLeser {
    protected BufferedReader buffer;

    /**
     * Konstruktor.
     * Oppretter et nytt FilLeserobjekt, med et
     * filnavn den skal lese fra.
     *
     *@param filnavn Filen FilLeseren skal lese fra
     */
    public FilLeser(String filnavn){
        try{
            buffer = new BufferedReader(new FileReader(filnavn));
        }
        catch(FileNotFoundException e){
            System.out.println("Filnavn ikke gyldig");
            System.out.println(e.getMessage());
        }
    }

    /**
     * Metode som leser en setning fra en fil.
     *@return Setningen som er lest,eller bare en tom streng dersom lesingen ikke var vellykket.
     */
    public String lesString(){
        try{
```

```
            return buffer.readLine();
        }
        catch(IOException e){
            System.out.println("Kunne ikke lese linje fra fil");
            System.out.println(e.getMessage());
            return "";
        }

    }

    /**
     * Metode som leser et tall fra en fil.
     * Metoden leser først en setning, og forsøker å
     * parse setningen til et tall. Dersom det ikke er
     * mulig, bl.a. hvis setningen ikke er et tall,
     * returneres tallet null.
     *
     *@return Tallet som er resultatet av parsingen
     */
    public int lesInt(){
        try{
            return Integer.parseInt(lesString());
        }
        catch(NumberFormatException e){
            System.out.println("Feil i å parse fra string til int,");
            System.out.println(e.getMessage() );
            return 0;
        }
    }

    public boolean klar(){
        try{
            return buffer.ready();
        }
        catch(IOException io){
            System.out.println("Buffer ikke klar");
            return false;
        }
    }

}
```

Forekomst

```
/**
 * @author ellenroy
 * Hjelpeklasse
 */
public class Forekomst {

    String navn;
    int antall;

    public Forekomst(String navn, int antall) {
        this.navn = navn;
        this.antall = antall;
    }

    public String toString(){
        return (navn +" " + antall);
```

```
    }

}
```

Kildekode til reglene

StorBokstavRegel

```
import java.util.ArrayList;

/**
 * @author ellenroy
 * Regel
 */
public class StorBokstavRegel extends Regel {

    public StorBokstavRegel() {
        super();
    }

    /* (non-Javadoc)
     * @see Regel#analyserSetning(Egennavn, java.util.ArrayList)
     */
    public boolean analyserSetning(Egennavn egennavn, ArrayList setning) {
        if(egennavn.getNavn().equals("BT")){
            return false;
        }
        int lengde = 0;
        boolean liten = false;
        //sjekker om alle boktavene er store
        while (!liten&& lengde< egennavn.getNavn().length()) {
            if(Character.isLowerCase(egennavn.getNavn().charAt(lengde))||
                    Character.isDigit(egennavn.getNavn().charAt(lengde))){
                liten = true;
            }
            lengde++;
        }
        //skiller ut de som er skrevet med stor bokstav
        if(!liten){
            storBokstav.add(egennavn);
            return true;
        }
        return false;
    }
}
```

TittelRegel

```
import java.util.ArrayList;


/**
 * @author ellenroy
 * Regel
 */
public class TittelRegel extends Regel {


    public TittelRegel() {
```

```
        super();
    }

    /*
     * @see Regel#analyserSetning(Egennavn, java.util.ArrayList)
     */
    public boolean analyserSetning(Egennavn egennavn, ArrayList setning) {
        if (egennavn.getPos() - 1 >= 0) {
            // denne regelen ser om det er en tittel
            // rett før egennavnet => annet
            // f.eks statsminister Bondevik
            Ord tittel = (Ord) setning.get(egennavn.getPos() - 1);
            for (int i = 0; i < (tittel.getFeatures()).size(); i++) {
                if (tittel.getFeatures().get(i).equals("@tittel")) {
                    annet.add(egennavn);
                    egennavn.getOrd().addFeature("ANNET");
                    return true;
                }
            }
        }

        if (egennavn.getPos() - 2 >= 0) {
            // denne regelen ser om det er en tittel
            // før en preposisjon før et egennavn => annet
            // f.eks leder av NSB, men ikke lederen av NSB
            Ord ord = (Ord) setning.get(egennavn.getPos() - 2);
            Ord tittel = (Ord) setning.get(egennavn.getPos() - 1);
            for (int i = 0; i < (ord.getFeatures()).size(); i++) {
                for (int j = 0; j < (tittel.getFeatures()).size(); j++) {
                    if (tittel.getFeatures().get(j).equals("prep")
                            && ord.getFeatures().get(i).equals("@tittel")) {
                        annet.add(egennavn);
                        egennavn.getOrd().addFeature("ANNET");
                        return true;
                    }
                }
            }
        }
        return false;
    }

}
```

SierRegel

```
import java.util.ArrayList;

/**
 * @author ellenroy
 *
 */
public class SierRegel extends Regel {

    public SierRegel() {
        super();
    }

    /* (non-Javadoc)
     * @see Regel#analyserSetning(Egennavn, java.util.ArrayList)
     */
```

```
public boolean analyserSetning(Egennavn egennavn, ArrayList setning) {
    ArrayList verb = lesFraFil("tekst/sierVerb.txt", "");
    if (egennavn.getPos() - 3 >= 0) {

        // sier NOEN/NOE til egennavnet
        Ord sier = (Ord) setning.get(egennavn.getPos() - 3);
        Ord pers = (Ord) setning.get(egennavn.getPos() - 2);
        Ord prep = (Ord) setning.get(egennavn.getPos() - 1);
        for (int i = 0; i < verb.size(); i++) {
            if (sier.getLemma().equals((String) (verb.get(i)))) {
                for (int j = 0; j < (prep.getFeatures()).size(); j++) {
                    if (prep.getFeatures().get(j).equals("prep")) {
                        for (int l = 0; l < (pers.getFeatures()).size(); l++) {
                            if (pers.getFeatures().get(l).equals("pron")
                                    ||pers.getFeatures().get(l).equals("ANNET")){
                                annet.add(egennavn);
                                egennavn.getOrd().addFeature("ANNET");
                                return true;
                            }
                        }
                    }
                }
            }
        }
    }

    if(egennavn.getPos()+ 2 < setning.size()){
        // sier Hans
        Ord sier = (Ord) setning.get(egennavn.getPos() +1);
        for (int i = 0; i < verb.size(); i++) {
            if (sier.getLemma().equals((String) (verb.get(i)))) {
                annet.add(egennavn);
                egennavn.getOrd().addFeature("ANNET");
                return true;
            }
        }

    }

    if(egennavn.getPos() - 2 >= 0){
        //sier til Hans
        Ord sier = (Ord) setning.get(egennavn.getPos() -2);
        Ord prep = (Ord) setning.get(egennavn.getPos() -1);
        for (int j = 0; j < (prep.getFeatures()).size(); j++) {
            if (prep.getFeatures().get(j).equals("prep")) {
                for (int i = 0; i < verb.size(); i++) {
                    if (sier.getLemma().equals((String) (verb.get(i)))) {
                        annet.add(egennavn);
                        egennavn.getOrd().addFeature("ANNET");
                        return true;
                    }
                }
            }
        }
    }
    //sier overingeniør Hosleif i Vegdirektoratet
    if (egennavn.getPos() - 4 >= 0) {
        Ord sier = (Ord) setning.get(egennavn.getPos() - 4);
        Ord tittel = (Ord) setning.get(egennavn.getPos() - 3);
        Ord pers = (Ord) setning.get(egennavn.getPos() - 2);
```

```
            Ord prep = (Ord) setning.get(egennavn.getPos() - 1);

            for (int j = 0; j < (prep.getFeatures()).size(); j++) {
                if (prep.getFeatures().get(j).equals("prep")) {
                    for (int i = 0; i < verb.size(); i++) {
                        if(sier.getLemma().equals((String) (verb.get(i)))){
                            for (int k = 0; k < tittel.getFeatures().size(); k++) {
                                if(((String)(tittel.getFeatures().get(k))).equals("@tittel")){
                                    for (int index = 0; index < pers.getFeatures().size(); index++) {
                                        if(((String)(pers.getFeatures().get(index))).equals("ANNET")){
                                            annet.add(egennavn);
                                            egennavn.getOrd().addFeature("ANNET");
                                            return true;
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }

            return false;
        }
    }
```

StedSubRegel

```
import java.util.ArrayList;

/**
 * @author ellenroy
 * Regel
 */
public class StedSubRegel extends Regel {

    public StedSubRegel() {
        super();
    }

    /* (non-Javadoc)
     * @see Regel#analyserSetning(Egennavn, java.util.ArrayList)
     */
    public boolean analyserSetning(Egennavn egennavn, ArrayList setning) {

        if (egennavn.getPos() - 2 >= 0) {
            // egennavnet har et stedssubstantiv foran seg
            Ord stedSub = (Ord) setning.get(egennavn.getPos() - 1);
            Ord prep = (Ord) setning.get(egennavn.getPos() - 2);
            for (int j = 0; j < (prep.getFeatures()).size(); j++) {
                if (prep.getFeatures().get(j).equals("prep")) {
                    ArrayList liste = lesFraFil("tekst/stedSub.txt", "");
                    for (int i = 0; i < liste.size(); i++) {
                        if (stedSub.getLemma().equals((String) liste.get(i))) {
                            stedsnavn.add(egennavn);
                            egennavn.getOrd().addFeature("STED");
                            return true;
                        }
```

```
                    if(((String) liste.get(i)).startsWith("*")){
                        String suffix = ((String) liste.get(i)).
                        substring(1,((String) liste.get(i)).length());
                        if (stedSub.getLemma().endsWith(suffix)) {
                            stedsnavn.add(egennavn);
                            egennavn.getOrd().addFeature("STED");
                            return true;
                        }
                    }
                }
            }

        }
    }
    if (egennavn.getPos() - 1 >= 0) {
        Ord stedSub = (Ord) setning.get(egennavn.getPos() - 1);
        ArrayList liste = lesFraFil("tekst/stedSub.txt", "");
        for (int i = 0; i < liste.size(); i++) {
            if (stedSub.getLemma().equals((String) liste.get(i))) {
                stedsnavn.add(egennavn);
                egennavn.getOrd().addFeature("STED");
                return true;
            }
            if(((String) liste.get(i)).startsWith("*")){
                String suffix = ((String) liste.get(i)).substring(1,((String) liste.get(i)).length());
                if (stedSub.getLemma().endsWith(suffix)) {
                    stedsnavn.add(egennavn);
                    egennavn.getOrd().addFeature("STED");
                    return true;
                }
            }
        }
    }

    return false;
}

}
```

SubRegel

```
import java.util.ArrayList;

/**
 * @author ellenroy
 * Regel
 */
public class SubRegel extends Regel {
    public SubRegel() {
        super();
    }

    /*(non-Javadoc)
     * @see Regel#analyserSetning(Egennavn, java.util.ArrayList)
     */
    public boolean analyserSetning(Egennavn egennavn, ArrayList setning) {
        Ord sub = (Ord)setning.get(egennavn.getPos()-1);
        for(int i = 0; i<(sub.getFeatures()).size(); i++){
            if(sub.getFeatures().get(i).equals("subst")){
                annet.add(egennavn);
```

```
                egennavn.getOrd().addFeature("ANNET");
                return true;
            }
        }
        return false;
    }

}
```

ForkRegel

```
import java.util.ArrayList;

/**
 * @author ellenroy
 * Regel
 */
public class ForkRegel extends Regel {

    public ForkRegel() {
        super();
    }

    /* (non-Javadoc)
     * @see Regel#analyserSetning(Egennavn, java.util.ArrayList)
     */
    public boolean analyserSetning(Egennavn egennavn, ArrayList setning) {
        if (setning.size() > egennavn.getPos() + 4){
            Ord vPar = (Ord) setning.get(egennavn.getPos() + 1);
            Ord hPar = (Ord) setning.get(egennavn.getPos() + 3);
            String fork = ((Ord) setning.get(egennavn.getPos() + 2)).getOrd();
            char temp = fork.charAt(0);
            //egennnavnet forann forkortelsen blir klassifisert
            if (vPar.getOrd().equals("(") && hPar.getOrd().equals(")")
                    && Character.isUpperCase(temp)) {
                annet.add(egennavn);
                egennavn.getOrd().addFeature("ANNET");
                eg = egennavn;
                return true;
            }
        }
        if (egennavn.getPos() - 2 >= 0 && setning.size() > egennavn.getPos() + 2) { //start - 2 >= 0

            // selve forkortelsen blir klassifisert
            Ord fulltNavn = (Ord) setning.get(egennavn.getPos() - 2);
            Ord vPar = (Ord) setning.get(egennavn.getPos() - 1);
            Ord hPar = (Ord) setning.get(egennavn.getPos() + 1);
            if (vPar.getOrd().equals("(") && hPar.getOrd().equals(")")) {
                for (int i = 0; i < fulltNavn.getFeatures().size(); i++) {
                    if ((fulltNavn.getFeatures()).get(i).equals("ANNET")) {
                        annet.add(egennavn);
                        egennavn.getOrd().addFeature("ANNET");
                        return true;
                    }
                }
            }
        }
        return false;
    }
```

```
}
```

SammenMedRegel

```
import java.util.ArrayList;

/**
 * @author ellenroy
 * Regel
 */
public class SammenMedRegel extends Regel {

    public SammenMedRegel() {
        super();
    }

    /* (non-Javadoc)
     * @see Regel#analyserSetning(Egennavn, java.util.ArrayList)
     */
    public boolean analyserSetning(Egennavn egennavn, ArrayList setning) {
        if(egennavn.getPos()-3 >= 0){
            //han sammen med Frode Skogvold
            Ord pers = (Ord)(setning.get(egennavn.getPos()-3));
            Ord sammen = (Ord)(setning.get(egennavn.getPos()-2));
            Ord med = (Ord)(setning.get(egennavn.getPos()-1));
            if(sammen.getOrd().equals("sammen")&& med.getOrd().equals("med")){
                for (int l = 0; l < (pers.getFeatures()).size(); l++) {
                    if (pers.getFeatures().get(l).equals("pron")) {
                        annet.add(egennavn);
                        egennavn.getOrd().addFeature("ANNET");
                        return true;
                    }
                }

            }

        }
        return false;
    }

}
```

ApposisjonsRegel

```
import java.util.ArrayList;
/**
 * Regel
 * @author ellenroy
 */
public class ApposisjonsRegel extends Regel {

    public ApposisjonsRegel() {
        super();
    }

    /* (non-Javadoc)
     * @see Regel#analyserSetning(Egennavn, java.util.ArrayList)
     */
    public boolean analyserSetning(Egennavn egennavn, ArrayList setning) {
        if(egennavn.getPos()+ 2 < setning.size()){
```

```
            //Ingrid, søsteren
            Ord komma = (Ord) setning.get(egennavn.getPos() + 1);
            Ord pers = (Ord) setning.get(egennavn.getPos() + 2);
            if(komma.getOrd().equals(",")){
                ArrayList liste = lesFraFil("tekst/person.txt", "#");
                for (int i = 0; i < liste.size(); i++) {
                    if(pers.getLemma().equals( (String)liste.get(i))){
                        annet.add(egennavn);
                        egennavn.getOrd().addFeature("ANNET");
                        return true;
                    }
                }
            }
        }
        return false;
    }

}
```

KommaRegel

```
import java.util.ArrayList;

/**
 * @author ellenroy
 * Regel
 */
public class KommaRegel extends Regel {

    public KommaRegel() {
        super();

    }

    /* (non-Javadoc)
     * @see Regel#analyserSetning(Egennavn, java.util.ArrayList)
     */
    public boolean analyserSetning(Egennavn egennavn, ArrayList setning) {
        if (egennavn.getPos() - 2 >=0 && setning.size() > egennavn.getPos() + 2) {

            Ord komma = (Ord) setning.get(egennavn.getPos() - 1);
            Ord navn = (Ord) setning.get(egennavn.getPos() - 2);
            Ord neste = (Ord) setning.get(egennavn.getPos() + 1);
            if (komma.getOrd().equals(",")
                    && (((Character.isUpperCase(neste.getOrd().charAt(0))))
                            || neste.getOrd().equals("og")
                            || neste.getOrd().equals("eller"))) {
                for (int i = 0; i < (navn.getFeatures()).size(); i++) {
                    if (navn.getFeatures().get(i).equals("STED")) {
                        stedsnavn.add(egennavn);
                        egennavn.getOrd().addFeature("STED");
                        return true;
                    }
                    if (navn.getFeatures().get(i).equals("ANNET")) {
                        annet.add(egennavn);
                        return true;
                    }

                }
            }
```

```
            //ble lagt til for å tolke Hafjell, Lillehammer
            if (komma.getOrd().equals(",")) {
                for (int i = 0; i < (navn.getFeatures()).size(); i++) {
                    if (navn.getFeatures().get(i).equals("STED")) {
                        stedsnavn.add(egennavn);
                        egennavn.getOrd().addFeature("STED");
                        return true;
                    }
                }
            }
        }
        return false;
    }

}
```

KonjRegel

```
import java.util.ArrayList;

/**
 * @author ellenroy
 *
 */
public class KonjRegel extends Regel {


    public KonjRegel() {
        super();
    }

    /**
     * @see Regel#analyserSetning(Egennavn, java.util.ArrayList)
     */
    public boolean analyserSetning(Egennavn egennavn, ArrayList setning) {
        if (egennavn.getPos() - 2 >= 0) {
            // egennavnet etter en konjunksjon vil vare av samme
            // type som det før konjunksjonen
            Ord og = (Ord) setning.get(egennavn.getPos() - 1);
            Ord navn = (Ord) setning.get(egennavn.getPos() - 2);
            if (og.getLemma().equals("og")||og.getLemma().equals("samt") ) {
                for (int i = 0; i < (navn.getFeatures()).size(); i++) {
                    if (navn.getFeatures().get(i).equals("ANNET")) {
                        annet.add(egennavn);
                        egennavn.getOrd().addFeature("ANNET");
                        return true;
                    }
                    if (navn.getFeatures().get(i).equals("STED")) {
                        stedsnavn.add(egennavn);
                        egennavn.getOrd().addFeature("STED");
                        eg = egennavn;
                        return true;
                    }
                }
            }
        }
        return false;
    }

}
```

VerbRegel

```
import java.util.ArrayList;

/**
 * @author ellenroy
 * Regel
 */
public class VerbRegel extends Regel {


    public VerbRegel() {
        super();
    }

    /*
     * @see Regel#analyserSetning()
     */
    public boolean analyserSetning(Egennavn egennavn, ArrayList setning) {
        if (egennavn.getPos() - 1 >= 0) {
            // verb før et egennavn => annet, med unntak
            // av forlate og slutte og være
            Ord verb = (Ord) setning.get(egennavn.getPos() - 1);
            if (verb.getLemma().equals("forlate")) {
                return false;
            }
            if (verb.getLemma().equals("slutte")) {
                return false;
            }
            if (verb.getLemma().equals("være")) {
                return false;
            }

            for (int i = 0; i < (verb.getFeatures()).size(); i++) {
                if (verb.getFeatures().get(i).equals("verb")) {
                    annet.add(egennavn);
                    egennavn.getOrd().addFeature("ANNET");
                    return true;
                }
            }
        }
        return false;
    }

}
```

PersRegel

```
import java.util.ArrayList;
import java.util.StringTokenizer;

/**
 * @author ellenroy
 * Regel
 */
public class PersRegel extends Regel {

    public PersRegel() {
```

```
        super();
    }

    /* (non-Javadoc)
     * @see Regel#analyserSetning(Egennavn, java.util.ArrayList)
     */
    public boolean analyserSetning(Egennavn egennavn, ArrayList setning) {
        String navn = egennavn.getNavn();
        ArrayList fornavn = lesFraFil("tekst/personNavn.txt", "#");
        StringTokenizer st1 = new StringTokenizer(navn);
        ArrayList person = new ArrayList();
        while (st1.hasMoreTokens()) {
        person.add(st1.nextToken());
         }

        // egennavnet er en person
        for (int i = 0; i < fornavn.size(); i++) {
            if(person.size()>= 2 && ((String)person.get(0))
                    .equals((String)fornavn.get(i))){
                annet.add(egennavn);
                egennavn.getOrd().addFeature("ANNET");
                return true;
            }
        }

        return false;
    }

}
```

YrkeRegel

```
import java.util.ArrayList;


/**
 * @author ellenroy
 * Regel
 */
public class YrkeRegel extends Regel {

    public YrkeRegel() {
        super();
    }

    /*
     * (non-Javadoc)
     *
     * @see Regel#analyserSetning(Egennavn, java.util.ArrayList)
     */
    public boolean analyserSetning(Egennavn egennavn, ArrayList setning) {

        if (egennavn.getPos() - 2 >= 0) {
            // lederen i Venstre => Venstre blir annet
            Ord yrke = (Ord) setning.get(egennavn.getPos() - 2);
            Ord prep = (Ord) setning.get(egennavn.getPos() - 1);
            for (int j = 0; j < (prep.getFeatures()).size(); j++) {
                if(prep.getFeatures().get(j).equals("prep")){
                    ArrayList liste = lesFraFil("tekst/personYrke.txt","");
                    for (int i = 0; i < liste.size(); i++) {
```

```
                    if(yrke.getLemma().equals((String)liste.get(i))){
                        System.out.println(yrke.getLemma());
                        annet.add(egennavn);
                        egennavn.getOrd().addFeature("ANNET");
                        return true;
                    }
                }
            }
        }

    }
    return false;
}

}
```

PrepRegelListe

```
import java.util.ArrayList;

/**
 * @author ellenroy
 *
 * Denne regelen må være i FASE B.3 og ta inn lister for å bestemme hva det kan være, ellers for mye
støy
 */
public class PrepRegelListe extends Regel {

    /**
     * Det lages en regel som sier at et egennavn er et stedsnavn hvis det er en preposisjon forann det.
     * @param setning
     * @param egennavn
     */
    public PrepRegelListe() {
        super();
        //skrivUt();
    }

    /**
     * Denne metoden sjekker om ordet forann er en preposisjon,
     * hvis det er tilfelle, blir egennanvet sett på som et stedsnavn
     */
    public boolean analyserSetning(Egennavn egennavn, ArrayList setning) {

        boolean iListe = false;
        Egennavn nyttEgennavn = null;
        if (egennavn.getPos() - 1 >= 0) {

            Ord prep = (Ord) setning.get(egennavn.getPos() - 1);

            if(prep.getOrd().equals("ifølge")){
                annet.add(egennavn);
                egennavn.getOrd().addFeature("ANNET");
                return true;
            }

            if (egennavn.getPos() - 2 >= 0) {
                Ord tiltale = (Ord) setning.get(egennavn.getPos() - 2);

                //tiltale mot
```

```
    if (prep.getOrd().equals("mot")
            && tiltale.getOrd().equals("tiltale")) {
        annet.add(egennavn);
        egennavn.getOrd().addFeature("ANNET");
        return true;
    }
}

for (int i = 0; i < (prep.getFeatures()).size(); i++) {
    if (prep.getFeatures().get(i).equals("prep")) {
        ArrayList org = lesFraFil("tekst/orgListe.txt", "#");
        for (int j = 0; j < org.size(); j++) {
            if (egennavn.getNavn().equals((String) (org.get(j)))
                    || egennavn.getOrd().getLemma().equals(
                            (String) (org.get(j)))) {
                annet.add(egennavn);
                egennavn.getOrd().addFeature("ANNET");
                return true;
            }

            //*skole, *sykehus
            if (((String) org.get(j)).startsWith("*")) {
                String suffix = ((String) org.get(j)).substring(1,
                        ((String) org.get(j)).length());
                if (egennavn.getNavn().endsWith(suffix)) {
                    annet.add(egennavn);
                    egennavn.getOrd().addFeature("ANNET");
                    return true;
                }
            }
            //Frp-verv ol Frp
            if (egennavn.getNavn().startsWith((String) (org.get(j)))
                    && ((String) org.get(j)).length() > 1) {
                annet.add(egennavn);
                egennavn.getOrd().addFeature("ANNET");
                return true;

            }
        }
        //Skansentunnelen og EØS-avtalen
        ArrayList annetSuffix = lesFraFil("tekst/annetSuffix.txt", "#");
        for (int j = 0; j < annetSuffix.size(); j++) {
            if(egennavn.getNavn().endsWith( (String) annetSuffix.get(j))){
                System.out.println(annetSuffix.get(j));
                annet.add(egennavn);
                egennavn.getOrd().addFeature("ANNET");
                return true;
            }
        }

        //Solidaritetsfondet Bouchiki
        ArrayList annetPrefiks = lesFraFil("tekst/annetprefiks.txt", "#");
        for (int j = 0; j < annetPrefiks.size(); j++) {
            if(egennavn.getNavn().startsWith( (String) annetPrefiks.get(j))){
                annet.add(egennavn);
                egennavn.getOrd().addFeature("ANNET");

                return true;
            }
        }
```

```
                // EØS-avtalens bestemmelser, Kurt Oddekalvs kollega osvS
                if(egennavn.getPos()+ 1 < setning.size()&&egennavn.getNavn().endsWith("s")){

                    Ord subst = (Ord)setning.get(egennavn.getPos()+1);
                    ArrayList stedSub = lesFraFil("tekst/stedSub.txt", "");
                    for (int j = 0; j < stedSub.size(); j++) {
                        if( subst.getLemma().equals( (String)stedSub.get(j))){
                            iListe = true;
                            nyttEgennavn = new Egennavn(egennavn.getNavn().concat(" "
+subst.getOrd()),egennavn.getPos(),egennavn.getSetningNr());
                        }
                    }

                    for (int j = 0; j <subst.getFeatures().size(); j++) {

                        if( ((String)subst.getFeatures().get(j)).equals("subst")&&!iListe){
                            annet.add(egennavn);
                            egennavn.getOrd().addFeature("ANNET");
                            return true;
                        }
                    }
                }
                if(iListe){
                    stedsnavn.add(nyttEgennavn);
                    egennavn.getOrd().addFeature("STED");
                    return true;
                }
                ArrayList sted = lesFraFil("tekst/sted.txt", "#");
                for (int j = 0; j < sted.size(); j++) {
                if (egennavn.getNavn().equals((String) (sted.get(j)))) {
                        stedsnavn.add(egennavn);
                        egennavn.getOrd().addFeature("STED");
                        return true;
                    }

                }
            }
        }
        return false;

    }
}
```

KommaRegel2

```
import java.util.ArrayList;

/**
 * @author ellenroy
 * Regel
 */
public class KommaRegel2 extends Regel {

    public KommaRegel2() {
        super();
    }

    /* (non-Javadoc)
```

```
 * @see Regel#analyserSetning(Egennavn, java.util.ArrayList)
 */
public boolean analyserSetning(Egennavn egennavn, ArrayList setning) {

    if (egennavn.getPos() - 2 >=0 && setning.size() > egennavn.getPos() + 2) {
        Ord komma = (Ord) setning.get(egennavn.getPos() - 1);
        Ord navn = (Ord) setning.get(egennavn.getPos() - 2);
        Ord neste = (Ord) setning.get(egennavn.getPos() + 1);

        char temp = neste.getOrd().charAt(0);

        if (komma.getOrd().equals(",")&& (((Character.isUpperCase(temp)))
                    || neste.getOrd().equals("og") || neste.getOrd().equals("eller"))) {
            ArrayList org = lesFraFil("tekst/orgListe.txt", "#");
            for (int i = 0; i < (navn.getFeatures()).size(); i++) {
                for (int j = 0; j < org.size(); j++) {
                    if((navn.getOrd().equals((String) org.get(j)))){
                        annet.add(egennavn);
                        egennavn.getOrd().addFeature("ANNET");
                        return true;
                    }
                    if (((String) org.get(j)).startsWith("*")) {
                        String suffix = ((String) org.get(j)).substring(1,
                                ((String) org.get(j)).length());
                        if (egennavn.getNavn().endsWith(suffix)) {
                            annet.add(egennavn);
                            egennavn.getOrd().addFeature("ANNET");
                            return true;
                        }
                    }
                    if (navn.getFeatures().get(i).equals("STED")) {
                        stedsnavn.add(egennavn);
                        egennavn.getOrd().addFeature("STED");
                        return true;
                    }
                    if (navn.getFeatures().get(i).equals("ANNET")) {
                        annet.add(egennavn);
                        return true;
                    }

                }


            }
        }
        //ble lagt til for å tolke Hafjell, Lillehammer
        if (komma.getOrd().equals(",")) {
            for (int i = 0; i < (navn.getFeatures()).size(); i++) {
                //System.out.println("SE HER " +navn.getFeatures().get(i));
                if (navn.getFeatures().get(i).equals("STED")) {
                    stedsnavn.add(egennavn);
                    egennavn.getOrd().addFeature("STED");
                    return true;
                }
            }
        }
    }


    return false;
```

```
    }

}
```

KonjRegel2

```
import java.util.ArrayList;

/**
 * @author ellenroy
 * Regel
 */
public class KonjRegel2 extends Regel {

    public KonjRegel2() {
        super();
    }

    /* (non-Javadoc)
     * @see Regel#analyserSetning(Egennavn, java.util.ArrayList)
     */
    public boolean analyserSetning(Egennavn egennavn, ArrayList setning) {
        if (egennavn.getPos() - 2 >= 0) {
            Ord og = (Ord) setning.get(egennavn.getPos() - 1);
            Ord navn = (Ord) setning.get(egennavn.getPos() - 2);

            if (og.getLemma().equals("og")||og.getLemma().equals("samt")||og.getLemma().equals("eller")
) {
                ArrayList org = lesFraFil("tekst/orgListe.txt", "#");
                for (int i = 0; i < org.size(); i++) {
                    if(egennavn.getNavn().equals( (String) org.get(i))){
                        annet.add(egennavn);
                        egennavn.getOrd().addFeature("ANNET");
                        return true;
                    }

                    if (((String) org.get(i)).startsWith("*")) {
                        String suffix = ((String) org.get(i)).substring(1,
                                ((String) org.get(i)).length());
                        if (egennavn.getNavn().endsWith(suffix)) {
                            annet.add(egennavn);
                            egennavn.getOrd().addFeature("ANNET");
                            return true;
                        }
                    }
                }

                for (int i = 0; i < (navn.getFeatures()).size(); i++) {
                    if (navn.getFeatures().get(i).equals("ANNET")) {
                        annet.add(egennavn);
                        egennavn.getOrd().addFeature("ANNET");
                        return true;
                    }
                    if (navn.getFeatures().get(i).equals("STED")) {
                        stedsnavn.add(egennavn);
                        egennavn.getOrd().addFeature("STED");
                        eg = egennavn;
                        return true;
                    }
                }
```

```
            }
        }
        return false;
    }

}
```

PrepRegel

```
import java.util.ArrayList;

/**
 * @author ellenroy
 * Regel
 */
public class PrepRegel extends Regel {

    public PrepRegel() {
        super();
    }

    /* (non-Javadoc)
     * @see Regel#analyserSetning(Egennavn, java.util.ArrayList)
     */
    public boolean analyserSetning(Egennavn egennavn, ArrayList setning) {
        if (egennavn.getPos() - 1 >= 0) {
            Ord prep = (Ord) setning.get(egennavn.getPos() - 1);
            for (int i = 0; i < (prep.getFeatures()).size(); i++) {
                if (prep.getFeatures().get(i).equals("prep")) {
                    stedsnavn.add(egennavn);
                    egennavn.getOrd().addFeature("STED");
                    return true;
                }
            }
        }
        return false;
    }

}
```

ListeRegel

```
import java.util.ArrayList;

/**
 * @author ellenroy
 * Regel
 */
public class ListeRegel extends Regel {

    public ListeRegel() {
        super();
    }

    /* (non-Javadoc)
     * @see Regel#analyserSetning(Egennavn, java.util.ArrayList)
     */
    public boolean analyserSetning(Egennavn egennavn, ArrayList setning) {
        ArrayList sted = lesFraFil("tekst/sted.txt", "#");
        for (int i = 0; i < sted.size(); i++) {
```

```
            if(egennavn.getNavn().equals((String)sted.get(i))){
                stedsnavn.add(egennavn);
                egennavn.getOrd().addFeature("STED");
                return true;
            }
        }

        ArrayList org = lesFraFil("tekst/orgListe.txt", "#");
        for (int i = 0; i < org.size(); i++) {
            if(egennavn.getNavn().equals((String)org.get(i))){
                annet.add(egennavn);
                egennavn.getOrd().addFeature("ANNET");
                return true;
            }
            // *skole, *sykehus
            if(((String) org.get(i)).startsWith("*")){
                String suffix = ((String) org.get(i)).substring(1,((String) org.get(i)).length());
                if(egennavn.getNavn().endsWith(suffix)){
                    annet.add(egennavn);
                    egennavn.getOrd().addFeature("ANNET");
                    return true;
                }
            }
            if (egennavn.getNavn().endsWith(".")&& egennavn.getNavn().startsWith((String) org.get(i)))
{
                annet.add(egennavn);
                egennavn.getOrd().addFeature("ANNET");
                return true;
            }
        }

        return false;
    }


}
```