Masteroppgave i informasjonsvitenskap

Et verktøy for visualisering av wiki-ontologier:

Enklere utvikling og bedre forståelse av ontologier?

André Sørbø
Institutt for informasjons- og medievitenskap
Det samfunnsvitenskapelige fakultet
Universitetet i Bergen
Juni 2008

Veileder: Weiqin Chen

Forord

Dette prosjektet markerer avslutningen på mitt masterstudium ved Universitetet i Bergen. Arbeidet med oppgaven har vært både spennende og lærerikt. Det har vært et krevende og utfordrende prosjekt og det er derfor med en stor lettelse og en følelse av tilfredsstillhet jeg ser alle bitene til slutt falt på plass. Jeg håper at både programmet som er blitt utviklet gjennom dette prosjektet og resultatene og erfaringene jeg har forsøkt å videreformidle vil være et viktig bidrag i et fagfelt som etter min mening vil bli svært viktig i fremtidens informasjonssamfunn.

Jeg vil til slutt takke min veileder, Weiqin Chen som har vært til stor hjelp med oppgaven.

Bergen, 26. juni 2008

Sammendrag

En ontologi er et system innenfor et bestemt fagområde som inneholder termer, spesifikasjon av disse termene og hvordan de henger sammen. Det er en konseptualisering, det vil si en forenklet beskrivelse av den verdenen eller det fagområdet vi ønsker å representere. Fagområdet kan være alt fra medisin til heissystemer eller dyrearter. Ontologier er viktige fordi de skal gi en felles forståelse av det aktuelle fagfeltet. For kunstig intelligens systemer brukes ontologier til å definere hva som "eksisterer" for det aktuelle systemet.

En wiki er en hjemmeside som kan leses og redigeres av "alle"[1] som leser den. Det kreves ingen spesiell programvare fra brukerens side. Den nye koden legges inn direkte i hjemmesiden.

Ontologier kan ofte være store og omfattende og utviklingen av disse vil derfor ofte bli utført av mange personer samtidig. Wikier kan være godt egnet for dette formålet, ettersom disse er lagt spesielt til rette for at flere personer kan samarbeide om å legge inn informasjon. Et av problemene for slike prosjekter, kan være å beholde oversikten når informasjonsmengden (ontologien) vokser. Gjennom dette prosjektet blir det foreslått at problemet kan løses ved hjelp av et verktøy som visualiserer innholdet i en wiki. Med en slik visualisering mener jeg at programvaren kan lese inn ontologien fra en gitt wiki-side for så å "oversette" den til en tilsvarende grafisk måte.

Problemstillingen som prosjektet ønsket å utforske og besvare var: "*Kan et verktøy for visualisering av wiki-ontologier føre enklere utvikling og bedre forståelse av ontologier?*"

Et slikt verktøy var ikke utviklet fra før og det ble derfor bestemt at det skulle utvikles en fungerende prototype av et slikt verktøy. Prototypen, som fikk navnet WikiVis, skulle senere bli brukt til å besvare problemstillingen. Hoveddelen av prosjektet har derfor gått ut på å planlegge, designe og utvikle en slik prototype. Denne delen er beskrevet i kapittel 3.

Problemstillingene til prosjektet ble besvart på grunnlag av en evaluering som ble foretatt av den ferdige prototypen. Resultatene og konklusjonen fra denne evalueringen viste at et slikt program for visualisering av wiki-ontologier vil være et nyttig hjelpemiddel, særlig for utviklere. Prototypen hadde likevel en del begrensninger, og det var enighet blant kandidatene som ble evaluert om at det ville være nødvendig å forbedre prototypen en god del før den ville være av særlig nytteverdi for andre enn utviklere og spesielt interesserte. Samtidig viste evalueringen at det finnes behov for et slikt program og at en fullverdig utvikling av programmet vil kunne bidra til forenkling av utvikling og bedre forståelse av wikiontologier.

[1] Krever riktignok innlogging i de fleste tilfeller. Noen sider begrenser også hvem som kan logge seg inn.

Innholdsfortegnelse

Tabeller og figurer

Prototypen

WikiVis 1.1b (Siste versjon ved avslutning av prosjektet)
URL: http://www.student.uib.no/~st01360/Master

Forkortelser og begreper

Term/Begrep	Forklaring
API	Application Programming Interface. Et sett med funksjoner og prosedyrer som et program, operativsystem eller bibliotek tilbyr for at andre programmer eller programmeringsspråk skal kunne kommunisere med det.
Applet	Et javaprogram som kan vises i en nettleser med java-støtte. Programmet kjører altså inne i nettleseren.
Class	Norsk: Klasse. I Java lagres koden i ”klasser”, hvor hver klasse representerer et dataobjekt (struktur). En javafil kan inneholde en eller flere klasser. Hver av disse blir ved kjøring oversatt til en egen fil med maskinkode (Med endelsen ”.class”).
Compiler	Et program som oversetter kildekode til maskinkode (Kjørbart program).
DAML+OIL	The DARPA Agent Markup Language. Dette språket ligner på OWL Full. Funksjonaliteten er stort sett det samme da OWL Full er bygget med dette språket som modell. DAML+OIL blir presentert i kapittel 2.
GUI	Graphical User Interface. Grafisk brukergrensesnitt. Den delen av et program som vises på skjermen og som brukeren bruker til å kommunisere med programmet.
IDE	Integrated Development Environment. Et programmeringsverktøy der de nødvendige utviklingsverktøyene er blitt integrert i et program. En IDE vil som oftest inneholde et grensesnitt for å redigere kode, en kompiler (som oversetter koden til maskinkode), verktøy for å finne feil (debugging) og verktøy for å utvikle grafiske brukergrensesnitt til programmer (GUI).
Java	Et programmeringsspråk utviklet av Sun Microssystems.
Java Package	En Java package er en måte å organisere Java-classer (filer). Hver class-fil kan tilhøre en package. Class-filer som tilhører samme package, kan få tilgang til hverandres metoder. En slik pakke kan og bli komprimert i en enkelt fil (JAR-fil).
Javadoc	Javadoc er et Java-verktøy som brukes til å dokumentere Java-kode. Dokumentasjonen skrives direkte i kildekoden som kommentarer. Javadoc henter frem disse kommentarene sammen med en oversikt over kildekodens struktur og funksjoner (metoder) og generer dokumentasjon i HTML-format.
JDBC	Java DataBase Connectivity. En API for kommunikasjon mellom Java-programmer og databaser.
Metode (Java)	I Java-sammenheng er en metode en funksjon.
Ontologi	Ontologier kan også defineres som læren om det eksisterende (ordenes mening). I datasammenheng blir ontologier da en måte å beskrive meningene og relasjonene til en term. Se kapittel 1.

Ontologi-språk	Et formelt dataspråk som kan brukes til å beskrive ontologier. Se kapittel 1.
OWL	Web Ontology Language. Standard språk for beskrivelse av ontologier utviklet av W3C. OWL formatet tillater alt som RDFS formatet tillater i tillegg til en rekke nye og mer avanserte funksjoner. OWL er delt i tre syntaks klasser: OWL Lite, OWL DL og OWL FULL. OWL blir gjennomgått i kapittel 2.
Parser	Et program som leser inndata og identifiserer strukturen til denne i henhold til en gitt grammatikk. Mer om parsere og parsing i kapittel 3.
Plug-in	En utvidelse til et program i form av et lite program som kommuniserer med hovedprogrammet og gir det en bestemt ekstra funksjonalitet.
RDF	Resource Description Framework. RDF er en modell og XML-syntaks for å representere informasjon slik at programmer kan ”forstå” meningen bak teksten. Det er bygget etter et utsagnsprinsipp (statements), med tripler i formen {predikat, subjekt, objekt}. Mer om RDF i kapittel 2.
Semantic Web	”Fremtidens internett” som bruker ontologier for å gi innholdet ”mening” for datamaskiner. Se kapittel 1.
SQL	Structured Query Language. Et standard interaksjons- og programmeringsspråk for databaser. Utviklet av IBM tidlig på 1970-tallet.
URL	Uniform Resource Locator. Internett-adresse (Link).
Visualisering	En grafisk ”oversettelse” av for eksempel en ontologi.
W3C	World Wide Web Consoritium. En internasjonal organisasjon for standardisering av verdensveven (internett) som blant annet har definert HTML-standarden.
Wiki	En nettside som tillater å redigere innholdet, uten bruk av eksterne programmer eller kode.
Wiki	En hjemmeside hvor det er mulig å endre innholdet direkte. Se kapittel 1.
Wiki-ontologi	En ontologi som er lagret på en wiki.
XML	eXtensible Markup Language. Markeringsspråk for dokumenter. Brukes i stor grad til også til overføring av data mellom ulike programmer. XML blir gjennomgått i kapittel 2.

1. Introduksjon

Et verktøy for visualisering av wiki-ontologier: Enklere utvikling og bedre forståelse av ontologier? Slik lyder tittelen og problemstillingen for denne oppgaven. Dette kapitlet vil forklare hva denne problemstillingen innebærer, hvorfor problemstillingen er viktig og hvordan den kan besvares. Den første delen av kapitlet vil gjennomgå problemdomenet og forklare de viktigste begrepene og teknologiene som er relevante for prosjektet mens den neste vil gi en klarere definisjon og forklaring av problemstillingen.

1.1 Problemdomenet

1.1.1 Ontologier

En ontologi er et system innenfor et bestemt fagområde som inneholder termer, spesifikasjon av disse termene og hvordan de henger sammen. Det er en konseptualisering, det vil si en forenklet beskrivelse av den verdenen eller det fagområdet vi ønsker å representere (Akrivi, Constantin et al. 2007). Fagområdet kan være alt fra medisin til heissystemer eller dyrearter. Ontologier er viktige fordi de skal gi en felles forståelse av det aktuelle fagfeltet. De kan være beregnet for mennesker, maskiner eller programmer. For kunstig intelligens systemer brukes ontologier til å definere hva som ”eksisterer” for det aktuelle systemet (Gruber 1992).

Ontologier kan også defineres som læren om det eksisterende (ordenes mening). I datasammenheng blir ontologier da en måte å beskrive meningene og relasjonene til en term. På denne måten kan en datamaskin ”forstå” hva de ulike termene betyr, hvilken relasjon de har til andre termer osv.

1.1.2 Semantic Web

Ontologier er en viktig del av fremtidens semantiske web. Den semantiske weben (eng: Semantic Web) er fremtidens verdensvev (World Wide Web) og en utvidelse av internett vi kjenner det i dag. Denne utvidelsen, som er definert av W3C[2] går ut på å bruke semantisk tilleggsinformasjon (ontologier) og på den måten å gi informasjonen på weben struktur og ”mening” for datamaskiner. Maskinene ”forstår” riktignok ikke innholdet på samme måte som mennesker, men den ekstra informasjonen gjør at informasjonen kan deles opp i logiske deler som kan bli mekanisk manipulert av maskiner. Dette åpner for en rekke nye muligheter da programmer vil ha mulighet til behandle informasjon på en langt mer intelligent måte. Det vil for eksempel bli mulig å foreta langt mer nøyaktige søk basert på innhold (i stedet for stikkord) og det blir lettere å kombinere informasjon fra ulike steder (Berners-Lee 2001; Denny 2002; Wikipedia 2008).

[2] W3C (World Wide Web Consortium) Se forklaring under termer og begreper etter innholdsfortegnelsen

1.1.3 Ontologispråk

For å kunne lagre og håndtere en ontologi i datasammenheng, er det nødvendig å ha et format å lagre denne i. Formatet må være i stand til å håndtere strukturen til ontologien. Strukturen vil gjerne bestå av ”klasser” som definerer begreper og ”instanser” som vil være data assosiert med den enkelte klasse. For eksempel: ”hund” kan være en klasse og hunden ”Fido” kan være en instans av denne klassen. ”Egenskaper” vil også være en naturlig del av en slik struktur (Eksempel: Fido er en ”snill” hund). Utover dette bør strukturen også innholde regler og restriksjoner på hva som er tillatt å lagre i ontologien og hvordan (Akrivi, Constantin et al. 2007). Eksisterende ontologispråk vil bli presentert i kapittel 2 (Tidligere forskning).

1.1.4 Wikier

En wiki er en hjemmeside som kan leses og redigeres av ”alle”[3] *som leser den. Det kreves ingen spesiell programvare fra brukerens side. Den nye koden legges inn direkte i hjemmesiden* (Soanes 2005; Marc 2006).

Wikier har fått en stor utbredelse de siste årene og har medvirket til å øke mengden av informasjon som deles blant nettbrukere dramatisk. En wiki forenkler nettpubliseringsprosessen betraktelig, slik at dette ikke lenger er forbeholdt de med over middels IT-kunnskaper. Wikiene er spesielt godt egnet når mange personer skal være sammen om å publisere informasjon.

1.1.5 Wiki-ontologier

Ontologier kan ofte være store og omfattende og utviklingen av disse vil derfor ofte bli utført av mange personer samtidig. Wikier kan være godt egnet for dette formålet, ettersom disse er lagt spesielt til rette for at flere personer kunne samarbeide om å legge inn informasjon.

En wiki-ontologi eller wiki-basert ontologi vil heretter være definert som en ontologi som er lagret på en wiki. For at en slik ontologi skal ha noen særlig nytteverdi bør wikiprogramvaren (serveren) ha støtte for ontologier. Dette kan være funksjonalitet for å redigere ontologier, herunder validering av kode og regler, og mulighet til å utføre spørringer (queries) på ontologien. I sin enkleste form kan en slik wiki-ontologi være en wiki der det er mulig å legge inn kode i et ontologiformat (OWL, RDF eller DAML+OIL) uten at wikien ellers har noen forståelse av hva dette er.

En wiki med ontologistøtte defineres som programvaren bak en wiki (server) som er tilrettelagt for å kunne lagre ontologier.

[3] Krever riktignok innlogging i de fleste tilfeller. Noen sider begrenser også hvem som kan logge seg inn.

1.1.6 Visualisering

Å visualisere en ontologi vil si å "oversette" den til en tilsvarende grafisk måte. En oversiktlig, høynivå visualisering kan gjøre det lettere å få et raskt overblikk og forståelse av hvordan en stor og komplisert ontologi er satt sammen. Visualisering kan også være en nyttig funksjon for navigering og utforskning av store ontologier. Interaktiv utforskning av data kan lede til flere oppdagelser innen relasjoner, mønstre osv (Babu 2008). Visualiseringer kan være alt fra enkle tre- og grafstrukturer, "topic maps" (som går ut på å lage en slags indeks eller kart over et emne) til mer avanserte tredimensjonale visualiseringer. Aktuelle visualiseringer vil bli gjennomgått og forklart senere.

1.2 Problemstilling

Wikier har fått en enorm utbredelse de senere årene. Fordi de er tilrettelagt for samarbeid ved at flere enkelt kan være med på å endre innholdet kan de også være godt egnet for utvikling av ontologier. Det finnes allerede en del wikier som støtter ontologier. Noen av disse vil bli presentert i kapittel 2. Det er også naturlig å tro at wikiontologier vil få større betydning etter hvert som ontologier og den semantiske weben tas mer i bruk. Som en konsekvens av flere, voksende wikiontologier vil det fort bli et problem å beholde oversikten over ontologien. Særlig dersom det er flere som samarbeider om utviklingen, noe som ofte vil være tilfelle for wikiontologier. Som nevnt, kan visualisering gjøre det lettere å få et raskt overblikk over en ontologi.

I dette prosjektet blir det foreslått at dette problemet kan løses ved hjelp av et verktøy som visualiserer innholdet i en wiki-ontologi. Med en slik visualisering mener jeg at programvaren kan lese inn ontologien fra en gitt wiki-side for så å "oversette" den til en tilsvarende grafisk måte. Prosjektet vil ha som mål å besvare hvorvidt det er behov for et slikt verktøy og om det kan være med på å forenkle og forbedre forståelsen av wiki-ontologier, og da særlig med tanke på utviklere av slike wiki-ontologier.

Problemstillingen for prosjektet blir derfor:

> *Kan man forenkle både utviklingen og forståelsen av wiki-ontologier ved hjelp av et verktøy som visualiserer ontologien?*

1.3 Løsningsforslag

For å besvare problemstillingen vil det være nødvendig å evaluere et slikt verktøy og kartlegge hvorvidt det vil kunne forenkle utvikling og forståelse av ontologier. For å gjøre dette, vil det bli utviklet en prototyp på et slikt verktøy som en del av prosjektet. Denne vil senere bli evaluert og resultatene av denne evalueringen vil være et viktig utgangspunkt for å kunne besvare problemstillingen. Prototypen har fått navnet "WikiVis".

Utviklingsstrategier og metoder som vil bli benyttet for å oppnå dette målet, samt resultater fra disse vil bli gjennomgått i de neste kapitlene.

1.4 Dokumentoversikt

Oppgaven er delt opp i følgende hoveddeler:

1) Kapittel 1 har forklart problemdomenet og problemstillingen for oppgaven. Kapittel 2 vil gjennomgå det teoretiske rammeverket for oppgaven og hvilke metoder som vil bli brukt for å løse oppgaven.
2) Kapittel 3 tar for seg hoveddelen av oppgaven, nemlig utviklingen og designen av en applikasjon for visualisering av wikiontologier.
3) Kapittel, 4 vil ta for seg evalueringen av dette programmet og gjennomgå erfaringene fra utviklingen.
4) I kapittel 5 vil disse resultatene bli brukt til å besvare problemstillingen for oppgaven: Kan et slikt verktøy føre til enklere utvikling og bedre forståelse av ontologier? Kapitlet vil også diskutere nytteverdien av den ferdige applikasjonen og eventuell fremtidig videreutvikling av dette.

2. Teori

Dette kapitlet vil undersøke og utforske teorier og metoder som kan bli brukt for å besvare problemstillingen som ble beskrevet i kapittel 1.

> *Metode (av gr. methodos, til hodos, vei, eg. vei mot målet), planmessig fremgangsmåte for å løse et problem, oppnå et resultat etc. — metodikk, læren om metodene, beskrivelse av den fremgangsmåte som brukes i et bestemt fag. — metodisk, planmessig, systematisk (Caplex 2008).*

Metoder en samling av regler, prosedyrer og teknikker for å nå et mål. Uten en god metode som styrer prosjektet kan utviklingen raskt ende opp som et mareritt av uforutsigbarhet, gjentatte feil, bortkastet tid og et generelt dårlig program.

Hovedformålet til denne masteroppgaven er å undersøke og besvare hvorvidt et verktøy for visualisering av wiki-ontologier kan føre til enklere utvikling og bedre forståelse av ontologier. Det ble foreslått i kapittel 1 å utvikle en prototyp på et slikt verktøy. Denne prototypen vil være et viktig bidrag for å kunne besvare problemstillingen. For det første vil utviklingen, med den nødvendige forstudien og designdelen, gi en bedre forståelse av problemstillingen. Senere vil en evaluering av den ferdige prototypen bli brukt til å måle nytteverdien av programmet og til å besvare hvorvidt prototypen, eventuelt et forbedret tilsvarende program, vil kunne føre til forenklet utvikling og enklere forståelse av ontologier. Evalueringen vil også ha som mål å avdekke om det er behov for et slikt verktøy.

Prosjektet vil derfor bestå av to deler: En design og utviklingsdel, der målet er å utvikle en prototyp for et verktøy som visualiserer wikiontologier, og en evalueringsdel, der målet er å besvare selve problemstillingen for prosjektet, nemlig i hvilken grad et slikt verktøy kan bidra til en forenklet utvikling og forståelse av slike ontologier. Metodene er derfor fordelt på to tilsvarende grupper: De som vil være aktuelle for design og utviklingsdelen, og de som vil være aktuelle for evalueringen av programmet. De konkrete valgene av metoder vil bli gjennomgått i kapitlene som omhandler bruken av disse.

2.1. Design og utviklingsdelen

Utviklingsdelen går ut på å utvikle en prototyp for et verktøy for visualisering av wikiontologier. De neste avsnittene vil beskrive ulike metoder for utvikling som vil kunne være aktuelle for denne delen. Den endelige avgjørelsen av hvilke metoder som er best egnet og som vil bli brukt til utviklingen, vil bli tatt i kapitlet som omhandler utviklingen, dvs. kapittel 3. Det finnes flere metoder som beskriver hvordan et utviklingsprosjekt bør utføres: Dette inkluderer håndtering av tekniske utfordringer, design, utvikling, styring og drift av et utviklingsprosjekt.

2.1.1. Fossefallsmetoden

I 1970 gav Winston Royce ut artikkelen ”Managing the Development of Large Software Systems”. Her delte han sine meninger om det som senere skulle bli kjent som fossefallsmetoden (Larman 2003). Denne metoden går i korte trekk ut på å dele prosjektet i faser, som skal løses fullstendig før man går i gang med neste. Fossefallsmetoden er en metode som brukes for hele livssyklusen til et program, fra planlegging, design og utvikling og til drift, utfasning og utskiftning. Hele utviklingen blir bestemt i en tidlig planleggingsfase basert på en kravspesifikasjon. Deretter følger utvikling, testing og drift (Tabell 1). Tradisjonelt har fossefallsmetoden vært mye brukt som metode for utviklingsprosjekter. Problemene med denne metoden er at utvikling kan være vanskelig å forutsi, det kan dukke opp nye krav eller problemer underveis. Metoden gir liten fleksibilitet ettersom man i utgangspunktet skal holde seg til fasene (Dix 1993).

Tabell 1. Fasene i fossefallsmetoden

Kravspesifikasjon	**Denne delen går ut på å fange opp og beskrive hva det endelige systemet skal eller bør ha av funksjoner og egenskaper.**
Design av arkitektur	Denne delen går ut på å bestemme hvordan funksjonene fra kravspesifikasjonen skal implementeres.
Detaljert design	Denne delen går ut på å finne den beste (eller mest hensiktsmessige) løsningen for hvordan de ulike funksjonene skal implementeres
Koding og komponenttesting	Denne delen går ut på å utvikle de ulike komponentene og testing for å se at de oppfyller vilkårene i henhold til gitte kriterier.
Integrering og testing	Når alle programmets komponenter er ferdigutviklet, må de bli integrert i et komplett program i henhold til arkitekturen som tidligere er blitt designet. Det vil være nødvendig med mer testing for å forsikre at systemet fungerer som det skal.
Vedlikehold	Etter at programmet er ferdig utviklet, vil alt videre arbeid med programmet regnes som vedlikehold. Denne delen går ut på å rette feil som blir oppdaget i ettertid av lansering, implementasjon av tilleggsfunksjoner osv.

2.1.2. Iterativ utvikling

Den største svakheten med fossefallsmetoden er kanskje at det er overambisiøst å tro at programutvikling kan foregå som strengt fastsatte faser. Det vil alltid bli oppdaget feil og behov kan bli endret underveis. Disse kan enten bli oppdaget under utviklingen eller gjennom testing.

Den eneste måten å være sikker på at den enkelte egenskap fungerer tilfredsstillende, er å utvikle den og teste den ut. Designet kan så bli endret dersom det viser seg at funksjonen ikke var implementert på en god nok måte. Det er dette som kalles iterativ design. Iterativ design blir av mange sett på som en modernisert fossefallsmetode (Tom 1985; Larman 2003).

Iterativ eller evolusjonær systemutvikling er basert på at systemet utvikles i "evolusjonære faser". Man utvikler en eller flere egenskaper, tester dem og retter opp eventuelle feil. Denne prosessen fortsetter til alle egenskapene fungerer tilfredsstillende. Iterativ design benytter prototyper, som inneholder eller simulerer deler av funksjonaliteten til det ferdige programmet. Det finnes tre hovedmåter å utvikle disse prototypene på:

1) Bruk og kast: Man utvikler en prototyp og tester denne. Erfaringene fra utviklingen og testingen av denne "øvelsen" blir brukt til å utvikle det endelige programmet, men prototypen brukes ikke i det ferdige produktet.
2) Inkrementell: Det ferdige programmet består av ulike komponenter, og disse blir utviklet en etter en. Det ferdige programmet blir lansert som en serie produkter. Hver lansering vil inkludere en ekstra komponent.
3) Evolusjonær: Denne metoden går ut på å utvikle en enkel prototyp og teste denne. Den foregående prototypen vil så bli brukt som basis for den neste prototypen. På denne måten blir programmet bedre. "Bedre" kan innebære mer funksjonalitet eller bedre funksjonalitet (raskere, mindre feil osv) (Dix 1993).

2.1.3. Agile metoder

Agile metoder, også kalt ekstrem programmeringsmetoder (XP) er en familie teknikker og praksiser som blant annet har fokus på fleksibilitet og tilpasningsdyktighet. Metodene er basert på stadige, raske leveranser av ferdig kode, omfattede testing og et tett samarbeid mellom utviklere og eksperter.

De agile metodene er resultatet etter at en gruppe industrieksperter, den såkalte "Agile alliansen", gikk sammen i 2001 for å løse problematikken med at stadig flere prosjekter ble sittende fast i en hengemyr av en utviklingsprosess. Agile metoder fokuserer på at mennesker er den viktigste faktoren for suksess: En god metode vil ikke være nok til å redde et prosjekt fra å mislykkes dersom

utviklerteamet ikke har gode medspillere. Samtidig er det ikke nok å ha gode folk, dersom disse ikke samarbeider godt nok. Kommunikasjon er derfor en viktig egenskap blant utviklerne. En annen viktig faktor for et vellykket utviklingsprosjekt er å ha gode verktøy. Et godt verktøy trenger derimot ikke å være det nyeste og dyreste. Det viktigste er at man forstår og håndterer verktøyene godt. Dokumentasjon er også en viktig del av et prosjekt. Men her er det viktig å begrense mengden dokumentering til det som strengt tatt er nødvendig. For mye dokumentasjon er sløsing med ressurser og er verre enn ingen eller mangelfull dokumentasjon.

Kommunikasjon med kunden er også viktig. For at et prosjekt skal lykkes, er det viktig at kunden (brukeren) av programmet følger utviklingen tett. Tilbakemeldinger og testing underveis er viktig. Det vil ofte være vanskelig, både for kunden og utvikleren, å forutsi alle egenskaper og funksjoner programmet skal ha. Misforståelser kan også oppstå. Derfor er det viktig at utviklingsplanene er fleksible slik at man kan svare på endringer og krav til nye funksjoner underveis (Martin 2001; Cockburn 2002).

2.2. Evalueringsdelen

> *En evaluering har tre hovedmål: Å vurdere omfanget og tilgjengeligheten til systemets funksjonalitet, å vurdere brukernes erfaring av interaksjonen, og å identifisere spesifikke problemer med systemet (Dix 1993).*

Slik lyder målene for en evaluering av et program. I tillegg vil evalueringen av prototypen bli brukt for å hente inn erfaringer og meninger fra brukerne for å kunne besvare oppgavens problemstilling.

Evalueringsdelen har derfor to målsetninger:

1) Å gi en vurdering av prototypen (Funksjonalitet, brukervennlighet osv).
2) Besvare problemstillingen: Kan et slikt verktøy forenkle utvikling og forståelse av wikiontologier?

For å kunne besvare (2) må prototypen være i stand til å utføre de viktigste oppgavene for et slikt visualiseringsprogram: Koble seg til en wiki, laste ned en ontologi fra denne, og gi en alternativ visualisering av ontologien.

2.2.1. Usability

Det ene formålet med evalueringen er få en vurdering av selve programmets ”brukbarhet” eller usability. Til denne delen vil det være behov for en usability metode. Usability studier er en fellesbetegnelse på metoder der man studerer eller analyserer egenskaper til et produkt for å finne ut om det oppfyller de nødvendige krav til brukervennlighet og funksjonalitet. Det finnes i hovedsak to typer usability studier: ”Expert usability review” og ”Usability testing”.

Expert usability review

Expert usability review (Abratt 2003; Tasha and David 2007) går som navnet antyder ut på å bruke eksperter under evalueringen. ”Eksperter” kan være utviklere, ”profesjonelle testere” eller andre med inngående kjennskap til produktet som skal testes. Disse vil gjennomgå programmet og dets funksjoner, enten fritt eller basert på heuristisk evaluering (basert på retningslinjer), for avgjøre hvordan programmet fungerer. Heuristic evaluation (Tasha and David 2007) er et eksempel på en slik metode. Denne ble introdusert i 1990 av Nielsen (som regnes som en ”guru” innen usability) og Molich og går ut på at en mindre gruppe usability-eksperter evaluerer brukergrensesnittet ved å bruke et sett med retningslinjer eller metoder og notere viktigheten av hvert usability-problem. I følge utviklerne av metoden, vil fem til ti slike eksperter være i stand til å avdekke 55 til 90 prosent av alle problemer med det aktuelle brukergrensesnittet. Andre eksempler på denne typen metoder er Cognitive walkthrough og Heuristic evaluation (Nielsen 2005).

Usability testing

Usability testing (US_Goverment; Wikipedia 2008) er en metode som brukes til å evaluere et produkt ved å teste det på en gruppe brukere. Det er et kontrollert eksperiment der testpersonene aktivt bruker produktet fremfor å bli vist hvordan det fungerer og så bli spurt om de skjønner det. Testingen har som målsetning å måle ulike sider ved produktet:

- Effektivitet: Hvor lang tid tar det å løse grunnleggende oppgaver.
- Nøyaktighet: Hvor mange feil gjør brukerne? Og hvor alvorlige er de i så fall?
- Recall: Hvor mye husker brukeren i etterkant. Vil han/hun klare å løse oppgavene igjen uten å måtte bli forklart hvordan det skal gjøres på nytt?
- ”Emosjonell respons”: Hvordan opplever brukeren systemet? Forstår han det? Kan programmet anbefales?

For å måle disse ulike sidene av produktet vil det være nødvendig å anvende ulike metoder for datainnsamling. Observasjon, høyttenkning eller opptak vil være nødvendig for å måle effektivitet og nøyaktighet, mens intervju, spørreundersøkelse eller lignende kan brukes for å måle emosjonell respons.

2.2.2. Datainnsamling

Observasjon

Observasjon er en svært enkel metode der den som evaluerer kun ser på og noterer hva kandidatene gjør når de bruker grensesnittet. Metoden brukes mye til testing av nettsider, men kan også brukes til å teste brukergrensesnittet til et program. Den som utfører testen, heretter kalt administrator, vil sitte nær kandidaten som blir testet for å kunne besvare eventuelle spørsmål testeren måtte komme

med underveis. Administratoren vil følge nøye med hvordan kandidaten beveger musen og navigerer i programmet. Negative kommentarer vil ofte være til mer hjelp enn positive, ettersom de negative som oftest vil være med på å avdekke problemer med brukergrensesnittet (Jerilyn and Matt 1999).

Tenke høyt

Denne metoden er nesten den samme som observasjon og går ut på at brukerne selv beskriver hva de gjør og hvorfor ved å tenke høyt. Resultatene blir enten notert eller tatt opp (Video eller lydopptak) (Abratt 2003).

Spørreskjema

Spørreskjema er en kvalitativ metode består av en serie spørsmål som på forhånd er definert av forskeren. Disse spørsmålene kan ikke endres av den som er med i testen og svarene er begrenset til et sett med alternativer. Metoden er ment å bli utført på et (ofte stort) antall potensielle respondenter og prosedyren vil være identisk for alle disse (Tett 2003).

Fordelen med å bruke spørreskjema er at man da har et presist utvalg med spørsmål og svaralternativer. Dette gjør det lettere å tolke dataene i ettertid. Selve testen er enkel å utføre. Den kan sendes på e-post, deles ut som et skjema eller legges ut på en nettside. Det er enkelt å utføre testen på en stor testgruppe og testpersonene har også mulighet til å være anonyme. Problemene med en slik test er blant annet at det kan være tidkrevende å lage et spørreskjema som fanger opp alle de nødvendige svarene (Hva med ting intervjueren ikke har tenkt på?). Dersom testpersonene ikke forstår spørsmålene har de ingen mulighet til å få avklart disse. Det kan også ta tid å få tilbake svarene.

Intervju

Et alternativ til datainnsamling er å bruke intervjuer. Intervjuer trenger i utgangspunktet mindre planlegging ettersom antall nødvendige spørsmål vil være mindre. Intervjuobjektet kan svare mer og forklare i større detalj hva han faktisk mener for hvert spørsmål. Intervjuobjektet har også mulighet til å avdekke spørsmål og problemstillinger en ikke hadde vært klar over på forhånd. Dette gir bedre svar og mer informasjon for hvert intervjuobjekt. Intervjueren har også mulighet til å gi hjelp underveis dersom intervjuobjektet ikke skjønner problemstillingen. Fordi man får mer informasjon fra hver enkelt testperson, kan man kan klare seg med færre intervjuobjekter. På den negative siden vil mangelen på systematiske svar og mye informasjon gjøre at etterbehandlingen av data bli tidkrevende.

2.3. Tidligere forskning og utvikling

Ontologier har fått mye oppmerksomhet de siste årene, ikke minst på grunn av den tidligere nevnte semantiske weben hvor de spiller en viktig rolle. De neste avsnittene vil gjennomgå ulike verktøy og tidligere utvikling som er relevant i forhold til dette prosjektet.

2.3.1. Ontologi språk

Ontologi-språk er nødvendige for å kunne lagre ontologier i datasammenheng. Noen formater som vil være aktuelle å støtte under utviklingen vil bli presentert i de neste avsnittene:

XML (eXtensible Markup Language)

XML er et fleksibelt tekstformat som er spesielt tilrettelagt for utveksling av informasjon. Det brukes for å definere dataelementer ved hjelp av ”Tagger” på samme måte som HTML. Disse brukes til å dele opp dokumentet i strukturelle deler. I motsetning til HTML er disse ikke forhåndsdefinerte og hvert dokument kan derfor tilrettelegges med sin egen struktur. Ved å tilby en standard for identifisering av data har XML blitt det mest brukte formatet for utveksling av informasjon mellom programmer og elektroniske tjenester (W3C 1996) (Figur 1).

```
<?xml version="1.0" encoding="utf-8" ?>
- <excludelist>
    <exclude art="about" val="creationDate" />
    <exclude art="Container" val="" />
    <exclude art="ConversionFactor" val="" />
    <exclude art="UnitType" val="" />
    <exclude art="NAryType" val="" />
    <exclude art="CustomType" val="" />
    <exclude art="BuiltinType" val="" />
    <exclude art="Type" val="" />
    <exclude art="Wikipage" val="" />
    <exclude art="hello" val="world" />
  </excludelist>
```

Figur 1. XML: XML-fil brukt av prosjektets prototyp, WikiVis. (Exclude list=Innhold som ikke skal tas med i en visualisering.)

RDF (Resource Description Framework)

RDF, også kjent som RDFS (RDF Skjema), er utviklet av W3C som en utvidelse av XML med støtte for ontologier. W3C (World Wide Web Consoritium) er en internasjonal organisasjon for standardisering av verdensveven (internett) som blant annet har definert HTML-standarden. W3C definerer formatet som en ”lettvekts ontologi ” for utveksling av kunnskap på weben. Formatet er basert på tripler som beskriver ”ressurser”. Triplene består av tre deler: subjekt, predikat og objekt. Eksempel: ”boken ligger på bordet” vil i RDF bli beskrevet som trippelen: Subjekt: Ressurs som beskriver boken, Predikat: Beskrivelse av egenskapen: ”ligger på” og Objekt: Ressurs som beskriver ”bordet”. De ulike triplene danner til sammen en graf, kalt metadatamodell. Denne måten å beskrive ”ressurser” på er

en viktig del av W3Cs Semantiske Web: Et forbedret verdensvev hvor automatisert programvare kan lagre, utveksle og bruke maskinlesbar informasjon som blir distribuert på verdensveven (Tauberer 2006; McBride 2007; Herman 2008).

OWL (Web Ontology Language)

OWL er en familie med språk for ontologiutvikling som er utviklet av W3C som en utvidelse av RDF. OWL består av tre "underspråk": OWL Lite er den enkleste formen og tillater enkle regler for å klassifisere hierarkier og enkle begrensninger i en ontologi. OWL DL er designet for å gjøre det mulig å utrykke så mye som mulig i ontologien samtidig som alle konklusjoner og spørringer ontologien tillater er garantert å kunne utføres av en datamaskin. OWL Full er den mest avanserte av språkene. Det finnes foreløpig ikke programvare som er i stand til å gi en komplett forståelse av alle ontologier utviklet i dette språket og det heller ikke sikkert at dette kan utvikles. Hvert av disse språkene er en utvidelse av det foregående, slik at alt som kan uttrykkes i OWL Lite kan uttrykkes i OWL DL og alt som kan uttrykkes i OWL DL kan uttrykkes i OWL Full, men ikke omvendt (Herman 2007).

DAML OIL. (The DARPA Agent Markup Language)

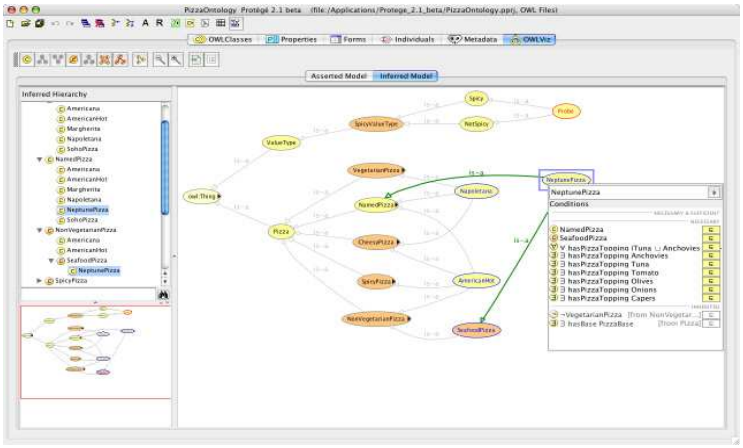
DAML var et prosjekt som gikk ut på å utvikle språk og verktøy for ontologier med den hensikt å forenkle Semantic Web. Det ble utviklet av DARPA (Defence Advanced Research Projects Agency) i perioden 2000 til 2006. Ontologispråket DAML+OIL er et resultat av dette prosjektet og er en utvidelse av XML og RDF (Horrocks 2001).

2.3.2. Eksisterende ontologiverktøy

Utforskning av eksisterende verktøy og programvare som er relatert til prosjektet vil være en viktig ressurs for å få en bedre forståelse og inspirasjon til utviklingen. Det finnes en rekke verktøy av ulike slag for å håndtere ontologier, enten det er snakk om redigering eller utvikling, utforskning (browsing), dokumentering, konvertering mellom formater eller grafisk visualisering. En del verktøy kombinerer flere av disse funksjonene. Dette avsnittet vil beskrive noen eksempler på slike verktøy som er blitt studert i forkant av utviklingen.

Protégé

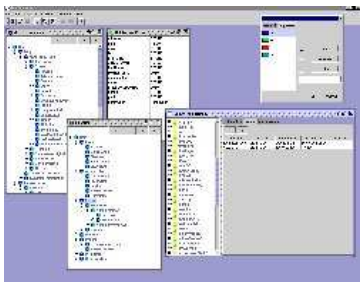
Protégé (Figur 2) er et av de mest komplette og mest brukte verktøyene for utvikling og visualisering av ontologier. Programmet er utviklet i Java og er gratis og åpen kildekodebasert. Programmet bruker OWL som hovedformat, men støtter også de fleste andre kjente ontologispråk, som RDF, DAML+OIL m.fl. En av de store styrkene til programmet er muligheten for utvidelser gjennom plug-ins. Ettersom programmet er åpenkildekodebasert finnes en rekke plug-ins som tilbyr ekstra funksjonalitet.



Figur 2. Protégé screenshot: Visualisering av OWL-ontologi

OntoEdit

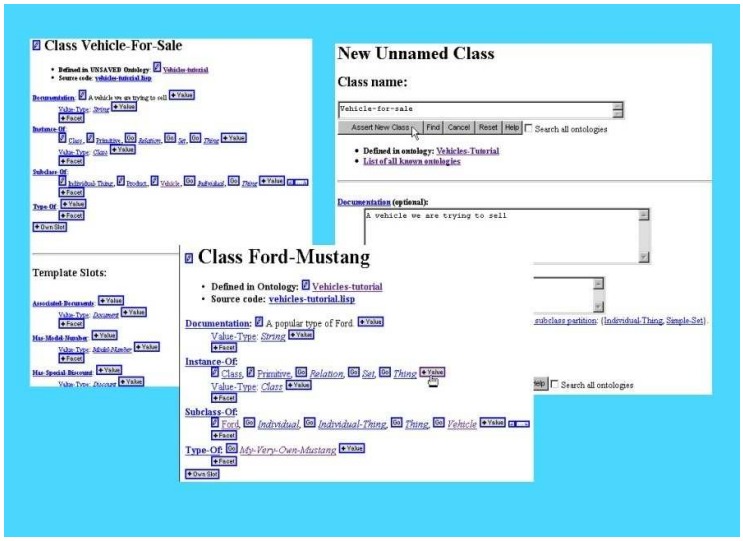
OntoEdit er et utviklingsverktøy for utvikling og vedlikehold av ontologier. Verktøyet er spesielt tilrettelagt for støtte av flere språk i samme ontologi. Ontologier utviklet med verktøyet kan redigeres gjennom et programmeringsgrensesnitt (API) og det er mulig å eksportere ontologiene til generiske, rasjonelle databaser gjennom JDBC grensesnittet. Blant formatene som støttes er RDF, OIL og F-Logic (Figur 3).



Figur 3. OntoEdit Screenshot (http://www.ontoknowledge.org/tools/ontoedit.shtml)

Ontolingua

Nettbasert verktøy. Grensesnittet består av enkel HTML-funksjonalitet, dvs. valg gjennom HTML-skjema (forms) og linker. Verktøyet bruker sitt eget format med samme navn for å representere ontologiene. Dette er et sært format som minner om programmeringsspråket LISP. Det er likevel mulig å eksportere til andre formater, hvorav DAML+OIL er det mest universelle innen ontologier. Verktøyet er tilrettelagt for samarbeid ved at flere personer kan logge seg på systemet og redigere den samme ontologien samtidig (Figur 4).



Figur 4. Ontolingua: Screenshots

DOE (The Differential Ontology Editor)

Dette er en enkel editor for å utvikle ontologier. Den har ikke noen form for visualisering, annet enn en tekstbasert utforskning av ontologiens hierarki. Den følger et spesielt sett med regler og metoder for å bestemme posisjonen til de ulike delene av ontologien dette hierarkiet. Programmet er utviklet i Java og støtter noen få ontologiformater, herunder OWL og RDF.

pOWL

pOWL er et verktøy for lagring og manipulering av RDF og OWL ontologier. Redigeringsmulighetene er relativt enkle, men verktøyet støtter også spørringer på ontologiene.

2.3.3. Samarbeid

Ontologier vil ofte være store og omfattende, derfor blir de ofte utviklet av flere personer. Støtte for samarbeid om utviklingen er derfor en viktig del av mange av de eksisterende ontologi-verktøyene. Av vertkøyene som er nevnt støtter Ontolingua og pOWL dette. Protégé støtter det gjennom ekstra plugin-funksjonalitet.

Wikier er verktøy som er spesielt tilrettelagt for at flere skal kunne samarbeide om å publisere innhold på nett. Det er derfor et naturlig steg at wikier etter hvert får støtte for å kunne håndtere ontologier. Neste avsnitt vil gjennomgå en del slike wikier.

2.3.4. Ontologi-wikier

Det finnes en rekke wikier som støtter ontologier. Enkelte av disse er utviklet fra grunnen for dette formålet. Eksempler på slike wikier er SweetWiki og IkeWiki. Andre har fått støtte for denne funksjonaliteten lagt til i ettertid. Semantic MediaWiki er et eksempel på det siste. De neste avsnittene gir en kort gjennomgang av ulike wikier som har støtte for ontologier.

SweetWiki

Dette er et prosjekt under utvikling og som er utviklet i Java. Den største forskjellen mellom denne og en vanlig wiki (uten støtte for ontologier), er at den inneholder en WYSIWYG editor (grafisk tillegg) hvor brukeren enkelt kan legge til den ekstra informasjonen som gjør dette til en ontologi. Struktur, arv, definering av begreper og lignende skjer på en grafisk måte uten bruk av kildekode slik de fleste andre ontologi-wikier gjør (SweetWiki 2007; Michel, Fabien et al. 2008).

Semantic MediaWiki

Semantic MediaWiki er et tillegg til MediaWiki (Programvaren som blant annet brukes av nettleksikonet Wikipedia). Dette tillegget gjør det mulig å legge til innhold i wikien i RDF eller OWL. Utover muligheten til å lagre innhold i disse formatene har er det stort sett opp til tredjepartsprogrammer å utnytte den ekstra funksjonaliteten disse formatene tilbyr (Heiko, Markus et al. 2006; SMW 2008).

IkeWiki

Denne wikien støtter RDF og OWL formatene. Både wikisidene og linker kan inneholde kode i disse formatene. I motsetning til Semantic MediaWiki forstår IkeWiki disse formatene og kan derfor blant annet utføre spørringer og validering av ontologiene som er lagret på wikien. Det er mulig å utføre ”smarte” søk basert på begreper og termer som er definert av ontologien, og ikke bare stikkord. For eksempel: Dersom en person som omtales i en side har egenskapene ”Forfatter” og ”kvinne”, vil det

være mulig å søke frem en liste over andre kvinnelige forfattere. En slik kobling ville vært umulig med en vanlig, stikkord-basert søkefunksjonalitet (Schaffert 2006; IkeWiki 2008).

FOAF (Friend of a Friend)

Dette er et eksperimentelt prosjekt som inneholder et Semantic web vokabular (ontologi) for å beskrive brukerne av systemet. Det er ikke en wiki i vanlig forstand, men prinsippene er omtrent de samme. Brukerne fyller ut informasjon om seg selv som lagres i ontologiformatet RDF. På denne måten kan informasjonen forstås av datamaskiner, som også vil kunne oppdage relasjoner mellom brukerne (Samme interesser, felles venner osv). Det samme prinsippet brukes også av de stadig mer populære nettsamfunnene (Facebook, MySpace osv.) og er bare en av mange måter å utnytte ontologier på.

2.4. Oppsumering

Dette kapitlet har gjennomgått det teoretiske rammeverket for prosjektet. Fra før er det blitt foreslått å besvare prosjektets problemstilling ved å utvikle og evaluere en prototyp av et verktøy for visualisering av wiki-ontologier. Metodene som er gjennomgått er derfor relatert til utvikling og evaluering av programvare. Eksempler på metoder som vil være aktuelle til utviklingen er ”Iterativ utvikling” og ”agile metoder”. For evalueringsdelen ble ulike usability metoder gjennomgått.

Siste del av kapitlet har gjennomgått eksisterende verktøy for utvikling og visualisering av ontologier og wikier med støtte for ontologier. Gjennomgangen viser at finnes det en rekke verktøy for begge disse formålene. Det finnes derimot ingen verktøy som er i stand til å koble seg til en gitt wiki og visualisere innholdet av denne. Det er dette gapet prosjektet vil forsøke å utforske gjennom utviklingen og evalueringen av en prototyp på et slikt verktøy.

3. Design –og utvikling

Dette kapitlet tar for seg design og utviklingsdelen av en prototype av et verktøy for visualisering av wikiontologier. Prototypen har fått navnet WikiVis. Kapitlet vil beskrive hovedtrekkene for hvordan utviklingen har foregått, hvilke krav som er satt og hvilke løsninger som er blitt benyttet samt hvorfor. De ulike delene av utviklingen har bestått av hver sine forprosjekt med innhenting av informasjon, design og planlegging. Detaljene for denne planleggingen vil ikke bli tatt opp, bare hovedtrekkene for designet som ble utviklet på grunnlag av disse.

3.1. Metode

I kapittel 2 ble aktuelle metoder for utviklingsdelen presentert. Blant disse var fossefallsmetoden som tradisjonelt har vært brukt som metode for utvikling, særlig for store prosjekter. Videre ble agile metoder beskrevet. Disse har fått mye oppmerksomhet de siste årene. Prototyping ble også presentert som en aktuell metode. Neste avsnitt vil forklare hvilke metoder som ble valgt for prosjektet og hvorfor.

3.1.1. Valg av metode

Fossefallsmetoden har den fordelen at den er enklest å administrere. Dersom man gjør et godt forarbeid i form av kravspesifikasjon og design og klarer å holde seg til planen vil denne metoden være godt egnet. Fordi dette er et lite prosjekt er nok administrasjon ikke det viktigste. Det vil også være et problem at de ulike fasene må gjøres ferdig før man går videre til neste fordi det kan være vanskelig å forutse alle krav og funksjoner som må implementeres på forhånd. Agile metoder har en langt bedre fleksibilitet og tilpasningsdyktighet, noe som vil være en klar fordel for prosjektet. Disse metodene er derimot basert på hurtige leveranser av kode som stadig blir endret basert på kommunikasjon med ”kunden”. Det kan bli vanskelig å overholde i dette prosjektet, ettersom det bare er en utvikler (vanskelig å produsere mye kode) og fordi det ville være nødvendig å bruke tid på å rekruttere brukere (”kunder”) til å kommunisere med. Evolusjonær design seiler dermed opp som den mest aktuelle metoden. Også denne metoden er fleksibel, men er mer fokusert på å utvikle en prototyp som blir stadig forbedret. Det kan dukke opp uforutsette problemer underveis og det er på forhånd vanskelig å vite hvilke deler av programmet som vil bli mest krevende å utvikle. Fordi det er begrenset tid og ressurser til disposisjon, er det heller ikke sikkert at det blir mulig å implementere alle ønskene til programmet. Ved å bruke en evolusjonær metode vil det være mulig å starte med de mest kritiske og krevende delene av programmet og utvide programmet gradvis. Fordi prosjektet først og fremst vil være opptatt av å utvikle en fungerende prototyp, vil det ikke være nødvendig å utføre mange små tester underveis, slik de agile metodene krever. En eller noen få tester når programmet ser ut til å fungere tilfredsstillende burde være nok.

Valget faller altså på evolusjonær (iterativ) metode for utviklingsdelen. Det vil bli utarbeidet en kravspesifikasjon som forteller hva programmet skal eller bør kunne gjøre. Utviklingen vil starte med å utvikle de funksjonene som må utvikles. Dersom det blir tid vil også funksjoner som bør/kan utvikles tatt med. Programmet vil bli utviklet i iterasjoner så langt prosjektet tillater det.

3.1.2. Proof of concept

Som det har blitt nevnt, er formålet med oppgaven først og fremst å besvare problemstillingen: ”Kan et verktøy for visualisering av wiki-ontologier føre til forenklet utvikling og bedre forståelse av slike ontologier?” Prototypen er i så måte et verktøy for å kunne besvare denne problemstillingen, først og fremst gjennom evalueringen, selv om utviklingen også vil kunne gi bedre forståelse av problemet. Prototypen vil derfor være en såkalt ”Proof of concept”. Det vil si at jeg ønsker å vise at det er mulig å utvikle et fungerende program som tilfredsstiller kravspesifikasjonen i prosjektbeskrivelsen og som kan brukes til å utføre en tilfredsstillende evaluering. Implementering av funksjonene i en eller annen form vil derfor være prioritet fremfor å perfeksjonere og optimalisere hver enkelt del.

3.2. Kravspesifikasjon

Før utviklingen kan starte er det nødvendig å avklare alle krav til programmet som skal utvikles. En kravspesifikasjon skal gi svar på spørsmålene ”Hva” og ”Hvorfor” når det gjelder hvilke funksjoner og generelle egenskaper et system må, bør og kan ha. Enkelte krav vil være viktigere enn andre (”Må” og ”bør ha”). Disse vil i størst mulig grad bli utviklet først, mens mindre viktige krav (”kan ha” / kjekt å ha”) vil bli utviklet til slutt dersom det blir tid til dette. Listen over krav ble satt opp på forhånd, men har til en viss grad blitt endret etter hvert som planlegging og utvikling har avdekket nye krav.

3.2.1. Funksjonelle krav

Funksjonelle krav er krav til hvilke *funksjoner* systemet skal eller bør ha.

Programmets funksjonelle krav er vist i Tabell 2 (Nr. vil bli brukt for å referere til de ulike kravene).

Tabell 2. Funksjonelle krav

Nr	Type	Krav	Forklaring
1	MÅ	kunne kommunisere med en wiki	En av programmets mest kritiske funksjon. Helt nødvendig for å kunne visualisere data fra en wiki.
2	MÅ	kunne laste ned en ontologi	Det er ikke nok at programmet kan kommunisere med en wiki. Det må også kunne laste ned ontologidataene for videre bearbeiding.
3	MÅ	kunne "forstå" ontologien	Forståelse er nødvendig for å kunne gi en korrekt visualisering
4	MÅ	kunne "sile" informasjon	Hensikten til programmet er å gi en bedre forståelse av ontologier. Derfor er det viktig at programmet er i stand til å fjerne unødvendig informasjon. Dette vil f.eks. være detaljinformasjon.
5	MÅ	kunne visualisere	Dette er det endelige målet for programmet og derfor en obligatorisk funksjon.
6	BØR	ha en chatfunksjon	Dette er en tilleggsfunksjon som vil gjøre det enklere å samarbeide med andre. (F.eks. hvis programmet brukes til utvikling)
7	BØR	støtte flere wikier	Programmet må støtte minimum en type wiki. Det er likevel en fordel om det støtter flere typer wikier.
8	BØR	være utvidbart	Programmet bør utvikles slik at det blir enkelt å utvide funksjonaliteten. Dette vil gjøre det lettere å forbedre programmet i ettertid.
9	BØR	tillate redigering	Redigering av ontologien i visualiseringen vil være en nyttig funksjon, men kan være vanskelig å implementere.
10	BØR	kunne lagre innstillinger m.m.	Det bør ikke være nødvendig å gjøre de samme tilpasningene hver gang programmet starter. Det kan likevel være greit å ha en mulighet for å gå tilbake til standardinnstillinger.

3.2.2. Ikke-funksjonelle krav

Ikke-funksjonelle krav er krav til *egenskaper* som programmet skal eller bør ha.

Programmets ikke-funksjonelle krav er vist i Tabell 3.

Tabell 3. Ikke-funksjonelle krav

Nr	Type	Krav	Forklaring
11	MÅ	være åpent	Programmet skal ikke være kommersielt, men fritt tilgjengelig
12	BØR	være intuitivt i bruk	Programmet bør programmet være logisk og intuitivt i bruk.
13	BØR	være optimert	Programmet bør ikke være krevende mer systemressurser enn nødvendig.
14	BØR	være stabilt	Programmet bør håndtere eventuelle feil som oppstår uten å krasje.
15	BØR	gi gode visualiseringer	Bedre visualisering = bedre forståelse. Det vil likevel være subjektivt hva som er "bra". En god visualisering skal gi et greit og forståelig overblikk av ontologien, gjerne med mulighet til å hente frem detaljer ved behov.

3.2.3. Krav til dokumentasjon

- Det skal skrives en enkel manual som dekker installasjon og innføring i bruk av programmet.
- All kode skal dokumenteres ved hjelp av Javadoc[4]
- Hjelp funksjonalitet i programmet
- Dokumentasjonen skal være på engelsk (Eventuelt norsk og engelsk) for at så mange som mulig skal kunne forstå den. Det samme gjelder for programmet.

3.2.4. Andre krav

- Programmet og koden skal publiseres på web
- Programmet skal være ikke-kommersielt og fritt tilgjengelig (Åpen kildekode)
- Som følge av punktet over skal alle eksterne biblioteker (APIer) også være ikke-kommersielle

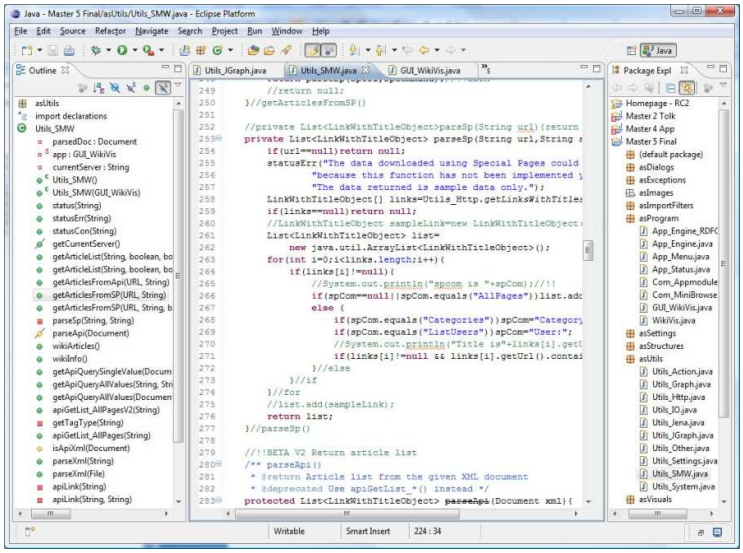
3.3. Verktøy

Det er blitt benyttet en rekke ulike verktøy under utviklingen av programmet. Dette avsnittet vil gi en kort presentasjon av de viktigste verktøyene som er blitt benyttet for prosjektet.

Java har blitt valgt som programmeringsspråk for prosjektet. Det er flere årsaker til at dette språket ble valgt fremfor andre språk. Den viktigste årsaken har vært at dette er det programmeringsspråket undertegnede er best kjent med. I tillegg er Java plattformuavhengig og kan i prinsippet kjøres på alle maskinvareplattformer (Windows, Mac, Unix, Linux osv). Det er også mulig å kjøre javaprogrammer direkte i en nettleser.

Hovedverktøyet var Eclipse 3.3 Europa (Figur 5). Dette er en komplett utviklingsplattform (IDE) for blant annet Java, utviklet av IBM. Dette programmet ble benyttet til å utvikle mesteparten av kildekoden for programmet. Mens Eclipse er godt egnet til å utvikle kode og funksjonalitet, har verktøyet noe begrenset mulighet for å utvikle grafiske brukergrensesnitt (Vinduer, dialogbokser, knapper osv). Derfor ble et annet verktøy, NetBeans 5.5.1 benyttet til denne delen. NetBeans gjør det mulig å ”tegne” det grafiske brukergrensesnittet ved å dra og slippe elementer for vinduer og knapper. NetBeans har stort sett samme funksjonalitet som Eclipse, og er utviklet av Sun Microsystems (Samme selskap som har utviklet Java).

[4] Se forklaring under termer og begreper etter innholdsfortegnelsen



Figur 5. Screenshot: Eclipse 3.3 Europa

Programmet genererer automatisk koden for dette designet. Andre verktøy som ble benyttet var grafiske verktøy for å lage skisser og planlegge det overordnede designet til programmet og designe grafiske elementer i programmet (Logoer, ikoner osv). MediaWiki, som er en wikiserver, ble også benyttet for å teste programmets funksjonalitet under utviklingen. Tabell 4 viser en liste programmer og verktøy som er blitt benyttet under utviklingen. De viktigste verktøyene og Javabibliotekene vil få en nærmere presentasjon senere.

Tabell 4. Verktøy som er brukt under utviklingen

Program/verktøy	Hensikt
Eclipse 3.3 Europa	Utviklingsplattform (Java kildekode)
NetBeans 5.5.1	Utvikling, design (Dialogbokser, brukergrensesnitt)
MediaWiki	Test wiki (XAAMP distribusjon m/PHP og MySQL + Semantic MediaWiki)
Adobe PhotoShop 7.0	Grafikk (Logoer, ikoner, knapper)
MS Visio 2007	Grafikk (Illustrasjoner, Mockups, forklaringer på programfunksjonalitet)
MS Virtual PC 2007	For å kunne teste MediaWiki og programmet i et isolert miljø, uten innblanding fra andre programmer.
Jena API	Java bibliotek for parsing av RDF/OWL ontologikode
JGraph API	Java bibliotek for å designe grafer ved hjelp av Java

3.4. Utviklingen

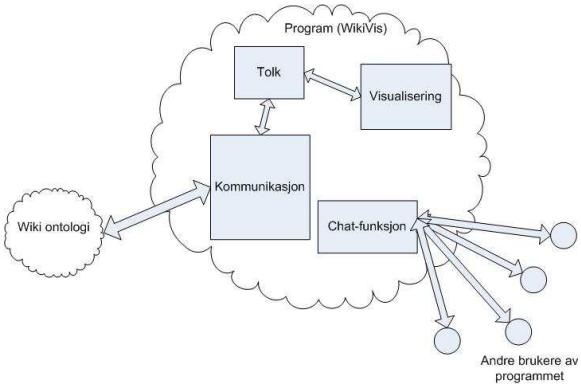
3.4.1. Planlegging og overordnet design

Utviklingen av et verktøy for visualisering av wikiontologier vil være et stort og omfattende prosjekt. For å ikke miste oversikten over utviklingen og i henhold til ”god programmeringsskikk” ble derfor de ulike hovedfunksjonene fordelt på ulike programmoduler. Ideen med modularitet er at det ferdige programmet skal bestå av uavhengige ”byggeklosser” eller ”moduler”. Hver modul har et minimalt antall inn og ut parametre som er nødvendig for å kunne utføre modulens oppgave. Resten av detaljene er skjult for de øvrige modulene (Også kjent som ”black box”). Utover dette er hver modul uavhengig. Fordelen med denne modellen er at hver modul kan byttes ut eller endres uten at dette får innvirkning på resten av programmet. Dette gjør det enklere å oppgradere og utvide programmet i ettertid, ikke minst for andre utviklere uten kjennskap til hele programmets struktur og funksjonalitet.

De ulike modulene vil være basert på generelle klasser. Funksjonaliteten og informasjonsflyten mellom de ulike modulene vil være bestemt gjennom disse klassene. På denne måten vil det være mulig å bytte ut hver enkelt modul uten at de øvrige modulene eller programmet trenger å forandres. Programmet vil være lagt opp til at det kan finnes flere implementasjoner av den enkelte modul, for eksempel flere ”tolker” for å håndtere ulike typer informasjon, flere ”visualiseringer” for å vise alternative grafiske fremstillinger av den samme ontologien osv. I Java er det to måter å implementere generelle klasser: *Interface* og *Abstract class*. *Interface* er den vageste av de to løsningene og definerer kun et sett med metoder (funksjoner) som må implementeres. Hva funksjonene faktisk gjør er helt opp til den enkelte implementasjon. *Abstract class* er en mer konkret løsning og består av en halvferdig klasse der deler av koden allerede er implementert, mens resten (det som vil skille ulike implementasjoner) blir definert som abstrakt. Abstrakte klasser må utvides for å kunne brukes og de abstrakte funksjonene må implementeres av underklassene. Alle utvidelser kan derimot behandles på samme måte fordi de har samme funksjoner. Forskjellen vil være at de gjennom ulik implementasjon håndterer funksjonaliteten på ulike måter.

Oppdeling i moduler

Programmet ble delt opp i tre moduler som ble utviklet uavhengig som egne prosjekter. Figur 6 viser en tidlig skisse av modulene og den overordnede funksjonaliteten til programmet. Chat-modulen som vises i figuren, ble ikke tatt med i det endelige programmet. Denne funksjonen er definert som mindre viktig i kravspesifikasjonen og ble fjernet fordi den ville ta for mye tid å utvikle. Av figuren fremgår det også hvordan de ulike modulene er avhengige av hverandre. Visualiseringsmodulen er avhengig av å kunne kommunisere med tolkmodulen som igjen er avhengig av dataene kommunikasjonsmodulen laster ned fra en wiki. Den mest programkritiske modulen er derfor kommunikasjonsmodulen. Den ble derfor utviklet først, etterfulgt av tolkmodulen og visualiseringsmodulen. De to siste ble delvis utviklet parallelt da utviklingen av visualiseringsmodulen avdekket nye behov og tilpasninger for å fungere optimalt med tolkmodulen.



Figur 6. Programmets moduler

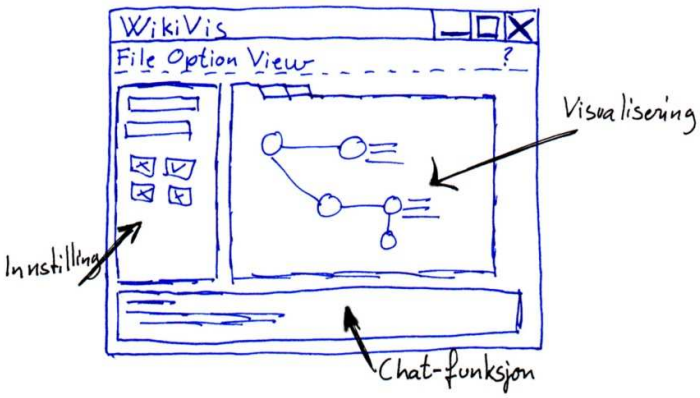
Programmets grafiske brukergrensesnitt (GUI) ble utviklet med denne modellen som grunnlag. Tabell 5 forklarer hvilken funksjonalitet de ulike modulene skal løse. Løsninger og valg som er gjort under resten av utviklingen vil bli gjennomgått i avsnittene som omhandler de ulike modulene.

Tabell 5. Tabell: Modulenes funksjoner

Modul	Oppgave/Funksjonalitet
Kommunikasjon	Tilkobling til wiki. Nedlastning av data ("Kildekode" for ontologi)
Tolk	"Forstå" og tolke nedlastet data. Klargjøre for visualisering.
Visualisering	Visualisere, dvs oversette ontologien til en alternativ, grafisk fremstilling

3.4.2. GUI

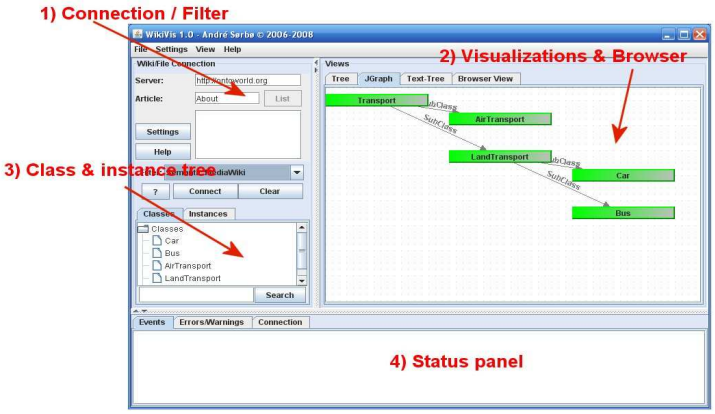
Programmets brukergrensesnitt (GUI) ble utviklet parallelt med programmets moduler. Denne delen har som formål å binde sammen de øvrige modulene samt å håndtere interaksjon mellom bruker og program. Det har vært en målsetting at denne delen av programmet skal være så intuitiv og brukervennlig som mulig. Figur 7 viser en tidlig skisse til hvordan brukergrensesnittet skulle se ut. Chat-funksjonen ble som nevnt fjernet. I stedet har denne delen av brukergrensesnittet fått en ny funksjon, ”Statuspanelet”. Dette er et tekstfelt de øvrige modulene kan bruke til å vise informasjon og feilmeldinger. Det ble også utviklet en egen nettleser for programmet. Denne kan brukes til å studere den opprinnelige wikisiden (uten visualisering). I tillegg har programmet en hjelpfunksjonalitet. Denne kan åpnes fra programmets meny.



Figur 7. Skisse til programmets brukergrensesnitt, første utkast

Figur 8 viser det endelige brukergrensesnittet til programmet. Som det fremgår av figurene har designet knapt blitt endret i forhold til den første skissen. Brukergrensesnittet er delt opp i 4 hoveddeler (Se Figur 8).

1)



Figur 8. Screenshot: De ulike delene av programmets GUI (WikiVis 1.0)

2) Denne delen inneholder funksjoner for å koble til en wiki. En kan velge hvilken type wiki programmet skal koble seg til fra en liste over ”Filter” (Nærmere forklaring i avsnittet om utvikling av kommunikasjonsmodulen). Knapper og felt for utfylling av informasjon vil være definert av filteret som er valgt.Visualisering: Programmet støtter flere visualiseringer. Det er lagt til rette for å legge til flere i ettertid. Det er mulig å veksle mellom de ulike visualiseringene.
3) Viser en oversikt over klasser og instanser i ontologien. Informasjonen hentes ved hjelp av Tolkmodulen
4) Statuspanelet: Viser meldinger og feilmeldinger (Kan brukes fritt av de ulike modulene).

Applet vs Java application

Javaprogrammer kan utvikles enten som selvstendige (Stand alone) programmer eller som ”Applets”. En ”applet” er et javaprogram som kan kjøres direkte i en nettleser. En av fordelene med denne løsningen er at brukeren ikke trenger å installere programmet på egen maskin, noe som forenkler bruken av det. På den andre siden er man da avhengig av at nettstedet som inneholder appleten er tilgjengelig når programmet skal kjøres. Den største begrensningen for appletløsningen er likevel de

sikkerhetsmessige begrensningene knyttet til denne typen programmer. Dette gjelder særlig tilgang til filsystem. Selvstendige javaprogrammer har ikke isse begrensningene. Et slikt program må derimot installeres og kjøres lokalt på en maskin og kan ikke kjøres i en nettleser. Selv om dette medfører ekstraarbeid i form av installasjon, betyr det at programmet ikke er avhengig av en eventuell tjenestetilbyder som tilbyr en applet over nett. Den tredje muligheten er å utvikle programmet slik at det fungerer både som applet og som selvstendig program. Dette krever derimot en god del tilpasninger for hvert av de to "modusene". Fordi en applet vil ha begrenset funksjonalitet ble denne løsningen forkastet. Hybridløsningen ble vurdert, men fordi dette vil medføre mye ekstraarbeid ble det bestemt at programmet skulle utvikles som et "fullverdig", selvstendig javaprogram.

Splits

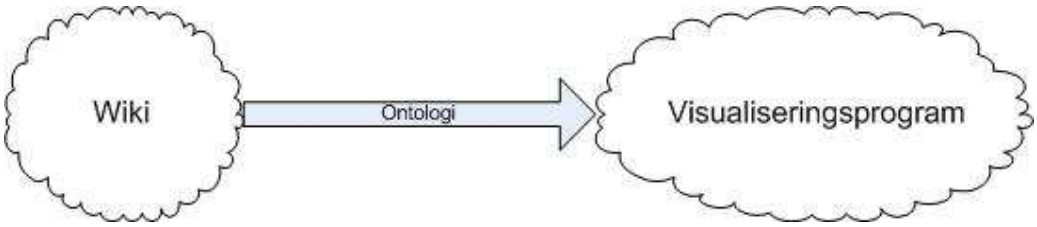
Det grafiske brukergrensesnittet er delt opp i tre hoveddeler (Tilkoblingsdel/kommunikasjon, visualiseringer/nettleser og statuspanel). Statuspanelet kan gi nyttige tilbakemeldinger, særlig dersom noe går galt (En tilkobling mislykkes eller en annen feil oppstår), men kan også ta beslag på unødvendig plass. Når en side først er visualisert, kan det dessuten være nyttig å forstørre visualiseringsvinduet så mye som mulig for å studere visualiseringen. For å få til dette er de ulike delene av programmet delt opp ved hjelp av såkalte "splits" (I Java kalt JSplitPane). Størrelsesforholdet mellom disse kan endres ved behov. De kan maksimeres eller minimeres eller man kan tilpasse størrelsen (Det er dog satt inn en del begrensninger og minimumsstørrelser på enkelte av delene).

Visualiseringer er delt opp ved hjelp av faner (Engelsk: Tabs, Java: JTabPane). Dermed kan man veksle mellom de tilgjengelige visualiseringene (Herunder nettleseren). Bare den valgte visualiseringen vil bli vist på skjermen. Statuspanelet har samme funksjon for å skille mellom ulike typer meldinger (Figur 9).

Figur 9. Inndeling ved hjelp av tabs (JTabMenu) og splits (JSplitMenu)

3.4.3. Programmodul: Kommunikasjon

Kommunikasjonsmodulen er den delen av programmet som håndterer kommunikasjonsdelen mellom program og wiki eller filsystem. Denne kommunikasjonen innebærer å koble seg til en støttet wiki og laste ned en ontologi fra denne, i kildekodeform. Figur 10 viser en forenklet skisse for denne funksjonaliteten.



Figur 10. Kommunikasjonsmodulens oppgave (Svært forenklet)

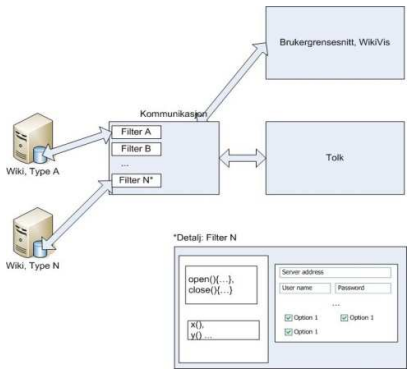
Importfilter grensesnittet

Et av kravene fra kravspesifikasjonen er at programmet bør støtte flere wikier og at det bør være mulig å utvide antall støttede wikier i ettertid. Det skal ikke være nødvendig å endre resten av programmet dersom det blir implementert støtte for en ny wiki. For å få til dette ble det utviklet et generelt grensesnitt for kommunikasjon mellom wiki (eventuelt fil) og resten av programmet. Dette grensesnittet ble kalt importfilter. På samme måte som programmets hovedmoduler ble importfiltergrensesnittet utviklet som en *abstract class*. Denne inneholder en del fellesfunksjonalitet og definerer et sett med generelle metoder. Importfiltergrensesnittet definerer to generelle (abstrakte) metoder som alle importfilter må implementere: *open()* og *close()*. Hovedprogrammet vil kalle *open()*for å opprette en tilkobling og laste ned ontologi fra en wiki og *close()* for å lukke tilkoblingen. Hvordan denne tilkoblingen og frakoblingen skjer, vil avhenge av hvordan disse funksjonene er implementert av det enkelte importfilter og vil være ukjent for hovedprogrammet. Figur 11 viser alle de abstrakte metodene et importfilter består av. De første fire (*filterName()*, *filterVersion()*, *filterDescription()* og *filterAbout()*)brukes til å beskrive det enkelte filter. For eksempel brukes *filterDescription()* av programmets hjelpefunksjon til å vise detaljinformasjon om det enkelte filter. Nederst ligger de to hovedmetodene *open()* og *close()*.

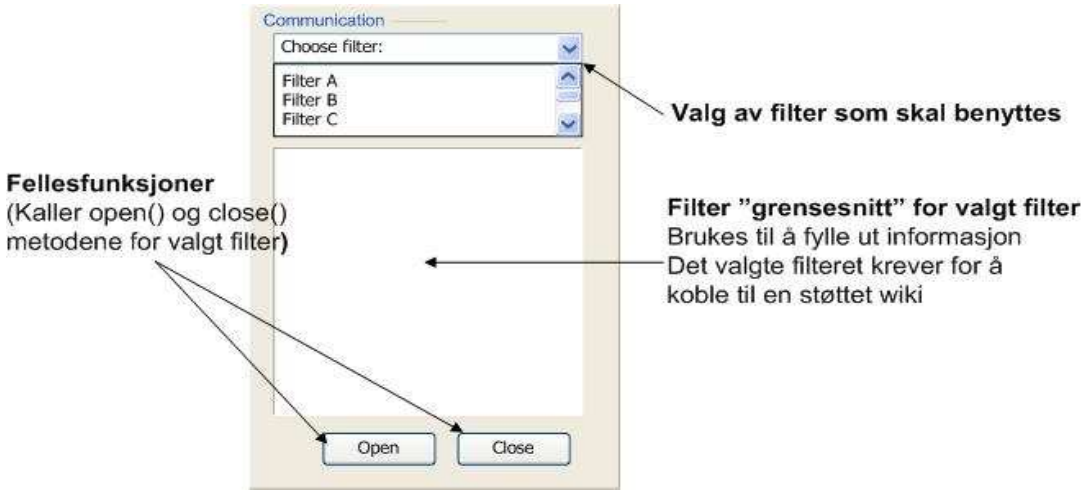
```
57    /** filterName()
58     * @return a String representing the (short) name of the filter */
59    public abstract String filterName();//filterName()
60
61    /** filterVersion()
62     * @return String representing the version of this filter */
63    public abstract String filterVersion();//filterVersion()
64    /** filterDescription()
65     * @return a String describing (in some detail) the purpose of the filter*/
66    public abstract String filterDescription();//filterDescription()
67    /** filterAbout()
68     * @return String describing credits about the filter
69     *         (Author, copyright, etc) */
70    public abstract String filterAbout();//filterAbout()
71
72    /** open()
73     * Opens a connection to the wiki/file and downloads an ontology
74     * The filter itself is then supposed to envoke the appopriate Engine
75     * and start the parsing and visualization */
76    public abstract boolean open();//open()
77
78    /** close()
79     * Closes the connection to the wiki/file */
80    public abstract boolean close();//close()
81
```

Figur 11. Importfilter - abstrakte metoder

En av de største utfordringene med å utvikle et slik felles grensesnitt, er at ulike implementasjoner av grensesnittet vil ha behov for ulik informasjon for å kunne utføre kommunikasjonen. Denne informasjonen må brukeren av programmet tilby. Et minimum av informasjon som vil være nødvendig er adressen til wikien. Annen informasjon som kan være nødvendig er påloggingsinformasjon, kommunikasjonsport og spesielle egenskaper knyttet til det enkelte grensesnitt. Dette problemet ble løst ved at hvert importfilter fikk mulighet til å lage sitt eget brukergrensesnitt for utfylling av nødvendig tilleggsinformasjon. Dette brukergrensesnittet vil bli vist i hovedprogrammet når det aktuelle filteret blir valgt. Figur 12 viser en skisse over hvordan kommunikasjonsmodulen vil fungere, mens Figur 13 viser hvordan kommunikasjonsmodulen blir vist i hovedprogrammet.



Figur 12. Skisse for design av Importfilter - Komplett



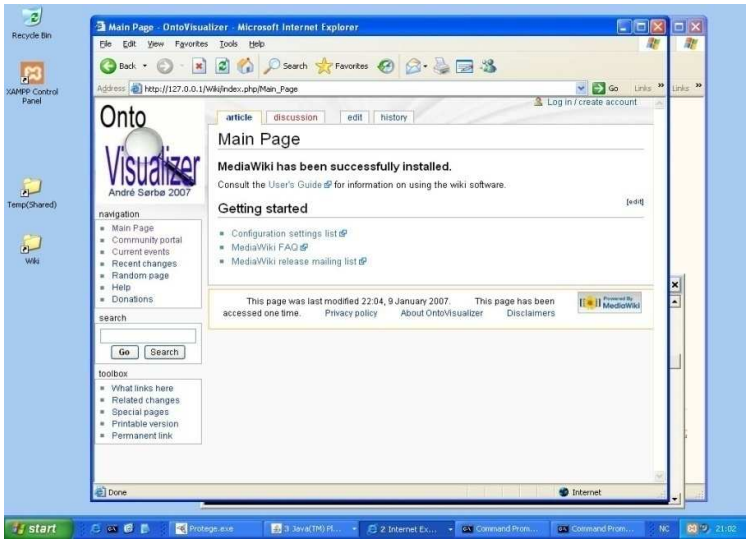
Figur 13. Mockup: Program - Kommunikasjonsmodul

Valg av støttede wikier

Prosjektets begrensede omfang gjorde det nødvendig å avgrense antall støttede wikier. I kapittel 2 ble ulike aktuelle wikier som støtter ontologier presentert. Prosjektet vil i første omgang bare støtte en eller to av disse wikiene.

Rent utviklingsmessig er wikier som er utviklet fra bunnen av med støtte for ontologier de mest velegnede for prosjektet da disse gjerne har bedre støtte for kommunikasjon med tredjepartsprogrammer. SweetWiki og IkeWiki er eksempler på slike. SweetWiki er utviklet i Java, noe som etter all sannsynlighet vil gjøre det lettere å få til kommunikasjon med prototypen, ettersom denne vil bli utviklet i samme programmeringsspråk.

Felles for disse wikiene er likevel at de er lite utbredt, noe som begrenser hvor mange som vil kunne ha nytte av programmet. Derfor ble Semantic MediaWiki (Figur 14) valgt som støttet wiki for prosjektet. Semantic MediaWiki er neppe den "beste" wikien med støtte for ontologier, men har langt større utbredelse enn de andre nevnte wikiene ettersom den er en utvidelse (Plugin) til den mest brukte og kjente wikiprogramvaren: MediaWiki. Denne programvaren brukes blant annet av nettleksikonet WikiPedia. MediaWiki støtter i utgangspunktet ikke ontologier, men Semantic MediaWiki utvidelsen gir denne støtten. Selv om det i første omgang bare vil bli utviklet et importfilter som støtter Semantic MediaWiki, er det er selvsagt fullt mulig å utvikle flere filtre i ettertid.



Figur 14. MediaWiki testserver benyttet under utviklingen av SMW Filteret

Hvordan få til kommunikasjon. Alternative kommunikasjonsmetoder

Semantic MediaWiki bruker SQL som underliggende database, mens brukergrensesnittet er webbasert. I tillegg finnes en eksportfunksjon (webbasert) som gjør det mulig å laste ned ontologi-innholdet. SQL-løsningen virket i utgangspunktet som den kraftigste løsningen, da denne ville gjøre det mulig å hente ned langt mer spesifikk informasjon enn eksportfunksjonen. Det ble utviklet et utkast til en denne løsningen som fungerte, men det ble etter hvert klart at en slik løsning ville være tungvindt og vanskelig å bruke for en vanlig bruker. De viktigste problemene med denne løsningen var:

1. Man må vite adressen til den underliggende databasen. Dette er et problem ettersom de færreste wiki tilbydere vil være villige til å gi ut denne typen informasjon av sikkerhetsmessige årsaker.
2. MediaWiki støtter flere typer SQL-databaser (MySQL, Microsoft SQL, PostgressSQL osv). Forskjeller i strukturen på disse kan by på problemer.
3. Programmet må kjenne til databasestrukturen og hvordan den kan finne frem til kildekoden for den enkelte ontologi. Siden wikien er laget for å bli redigert gjennom webgrensesnittet og ikke gjennom databasen, vil det være vanskelig å finne frem til riktig informasjon.

Disse problemene gjorde at den opprinnelige løsningen ble forkastet og den videre utviklingen baserte seg på programmets eksportfunksjon. Dette er en webbasert funksjonalitet som kan eksportere innholdet av en gitt wikiside til RDF formatet. Dette formatet er en utvidelse av XML som støtter ontologier. Det er også mulig å ”pakke inn” det mer spesialiserte formatet OWL i en RDF-fil.

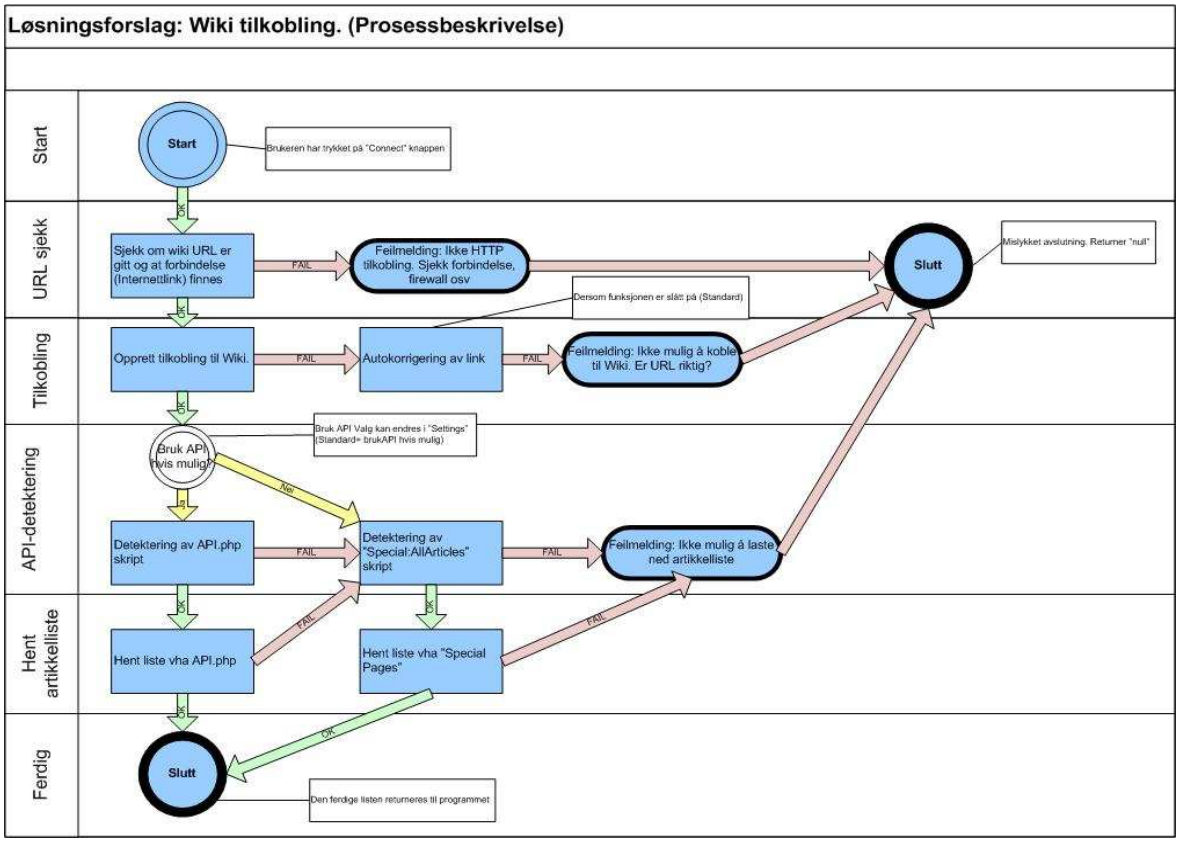
Eksportfunksjonaliteten fungerer ved at man går inn på en spesiell side i wikien (Såkalt ”Special Page”) som håndterer eksport. Her må navnet på siden som skal eksportes fylles ut før eksporteringen kan skje. Importfilteret som ble implementert håndterer dette ved å fungere som en webklient som automatiserer og skjuler denne prosessen. Det nedlastede innholdet, i RDF-format, blir så videresendt til tolkmodulen.

”Smartere” kommunikasjon

For at kravet om at programmet skal være mest mulig brukervennlig skal bli oppfylt, bør så mange som mulige avgjørelser tas av programmet selv. For kommunikasjonsmodulen vil disse avgjørelsene bli tatt i det enkelte importfilter. En viktig forenkling av kommunikasjonsdelen går ut på å utstyre denne delen med ”smarte” metoder som retter eventuelle feil og finner frem til riktige valg på egenhånd.

Et eksempel på dette er eksportfunksjonen. Denne trenger ikke ligge på en standardadresse, selv om dette er vanlig. Dersom wikien bare kjører MediaWiki (Uten Semantic utvidelsen) vil ikke funksjonen være tilgjengelig. Det er også mulig å slå av funksjonen. Alle disse unntakene bør bli håndtert automatisk. Programmet bør være i stand til å oppdage adressen til funksjonen eller gi en feilmelding hvis funksjonene ikke er tilgjengelig. For importfilteret som ble utviklet, var målet at wikiadressen skulle være den eneste nødvendige parameteren, og selv feil i denne burde bli korrigert av programmet i den grad det var mulig. Sidene som var tilgjengelige på wikien ble lastet ned automatisk og vist som en liste i programmet (Importfilterets grensesnitt). Til dette ble det brukt to metoder: Førstevalget var MediaWikis ”API-funksjonalitet”. Dette er et eget grensesnitt som tilbyr en rekke funksjoner for å utføre spørringer og hente ned informasjon gjennom tredjepartsprogrammer. Dessverre er funksjonaliteten av sikkerhetsmessige årsaker ofte slått av. Hvis API-funksjonaliteten er slått av vil nedlastningen av sidene i stedet skje gjennom en alternativ, webbasert løsning som ligner på den som ble brukt for RDF-eksport. Den siste løsningen er mer tidkrevende enn den første fordi programmet må gjennomgå listen som blir returnert i HTML-format og selv hente ut alle linker til wikisider (med forklaringer) og fjerne unødvendig informasjon (for det meste HTML-formatering).

Figur 15 viser hvordan importfilteret for Semantic MediaWiki benytter feilhåndtering for å koble seg til en wiki og laste ned en liste over tilgjengelige sider på best mulig måte. Hvert punkt vil forsøke å rette eventuelle feil eller bruke alternative løsninger hvis det er mulig. Hvis et av punktene likevel skulle feile, betyr det at en tilkobling ikke er mulig og en passende feilmelding vil bli skrevet ut:



Figur 15. Feilhåndtering ved oppkobling til Semantic MediaWiki

1) URL-sjekk. Sjekker at programmet har fått et URL (internettadresse) å jobbe med og at programmet har mulighet for å gjøre en tilkobling.
2) Opprette tilkobling. Dette innebærer å koble til URL og sjekke at det ligger en støttet wiki bak denne adressen. Dersom dette ikke går, vil programmet forsøke å rette opp adressen (legge til http://, http://www osv, fjerne deler av adressen som kanskje ikke skulle vært der osv). Eventuell adresseomdirigering må også håndteres av denne delen (Eks: ontoworld.org -> http://semanticweb.org/wiki/Main_Page).
3) Når vi dette punktet, finnes wikien. Neste steg er å "oppdage" om programmet har tilgang til wikiens API-funksjonalitet. API-funksjonaliteten brukes bl.a. til å hente ned en liste over tilgjengelige sider. Hvis funksjonen ikke er tilgjengelig vil programmet forsøke å få tilgang til "Special:AllArticles"-skriptet. Dette er en lignende, men mer krevende og dårlig løsning, derfor brukes den som andrevalg. Hvis heller ikke denne er tilgjengelig gir programmet opp. Når det gjelder "oppdaging av funksjonalitet" må programmet ta høyde for at dersom en funksjon ikke eksisterer (=ugyldig adresse for funksjonen), vil man automatisk bli omdirigert til en side som forteller at siden man forsøkte å koble til ikke finnes. For at programmet skal

forstå om den endelige siden er en feilmeldingsside eller om funksjonene bare har endret adresse, benytter programmet seg av HTTP statuskoder. I stedet for å laste ned hele siden, laster programmet ned en ”header”, som gir informasjon om siden. Blant denne informasjonen er en statuskode (tallverdi) som indikerer hvordan nedlastningen gikk (vellykket, mislykket, omdirigering osv). [5]

4) Hent ned listen. Dersom API ble gjenkjent brukes denne. Skulle denne funksjonen feile går programmet tilbake til (3) og forsøker den alternative metoden. Hvis API ikke var gjenkjent forsøker programmet å laste ned listen ved hjelp av ”Special-Pages”-funksjonen.
5) Tilkobling og nedlastning av tilgjengelige sider var vellykket. Listen returneres.

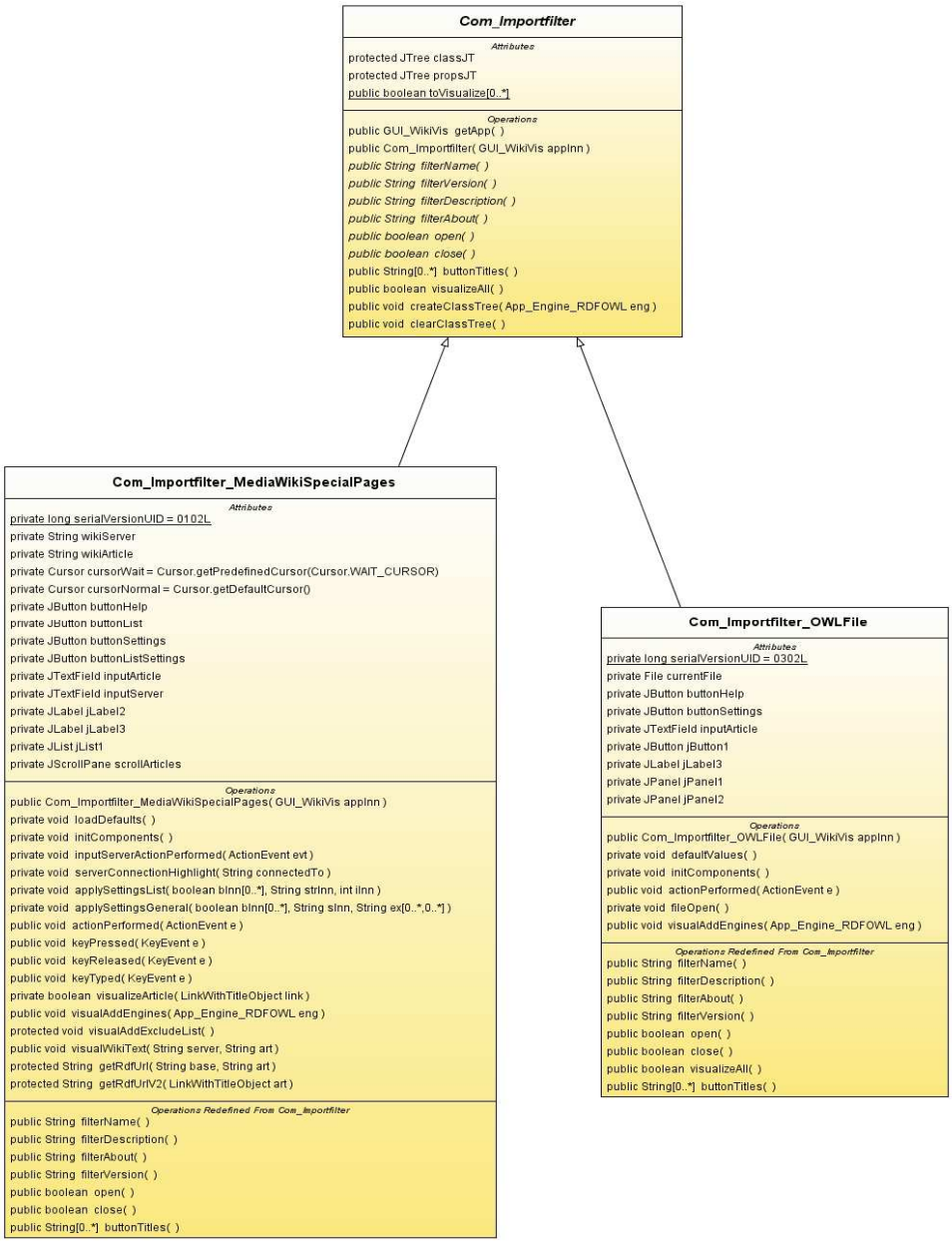
Implementasjonen av dette løsningsforslaget ble gjort med en ”top-down” strategi. I korte trekk fungerer den ved at hovedmetoden, *getArticles()* (kalt fra SMW-importfilteret) styrer og utfører de ulike stegene fra løsningsforslaget ved hjelp av en rekke hjelpemetoder. URL-sjekking og retting skjer for eksempel ved hjelp av kall til *Utils_Http.correctLink()*. Deretter brukes metodene *getArticlesFromSP()* og *getArticlesFromAPI()* til å hente listen av sider. Metoden *getArticlesFromSP()* inneholder koden som trengs for å hente ned en slik liste ved hjelp av ”Special:AllArticles”-skriptet (Men kan også brukes til andre tilsvarende skript), mens *getArticlesFromAPI()* håndterer tilgang gjennom wikiens API-funksjonalitet. For flere detaljer henvises det til kildekoden og dokumentasjonen til denne. Figur 16 viser koden til *getArticlesFromSP()* metoden. Som koden viser, bruker også denne metoden hjelpemetoder. Blant annet brukes *Utils_Http.correctLink()* til å teste om en URL er gyldig. Denne metoden bruker blant annet HTTP statuskoder for å bestemme om en URL er gyldig. Metoden *statusCon()* brukes til å skrive meldinger til programmets statuspanel.

```
207    public List<LinkWithTitleObject>getArticlesFromSP(URL url,String spCommand){
208        return getArticlesFromSP(url,spCommand,false);
209    }//getArticlesFromSP()
210    public List<LinkWithTitleObject> getArticlesFromSP(
211            URL url,String spCommand,boolean directLnk){
212        if(url==null)return null;//Nothing to do
213        if(spCommand==null)spCommand="AllPages";
214        String base=url.getProtocol()+"://"+url.getHost();
215        if(directLnk)base=url.toString();
216        //printURL(url);//!!TEST
217        String spUrl=Utils_Http.correctLink(base+"/wiki/Special:"+spCommand);
218        if(spUrl==null){
219            statusCon(base+"/wiki/Special:"+spCommand+" failed.. Trying to correct.");
220            spUrl=Utils_Http.correctLink(base+"/w/Special:"+spCommand);//if
221        }//if
222        if(spUrl==null){
223            statusCon(base+"/w/Special:"+spCommand+" failed.. Trying to correct.");
224            spUrl=Utils_Http.correctLink(base+"/Special:"+spCommand);
225        }//if
226        if(spUrl==null){
227            statusCon(base+"/Special:"+spCommand+" failed.. Giving up.");
228            return null;//Not possible to find "Special:spCommand"
229        }//if
230        statusCon("Special pages detected.. Downloading source using "+spUrl);
231        return parseSp(spUrl,spCommand);
232    }//getArticlesFromSP()
233
```

Figur 16. Importfilter, hente liste over artikler

[5] Mer informasjon om dette på http://w3.org/Protocols og http://www.w3.org/Protocols/HTTP/HTRESP.html

Figur 17 viser klassestrukturen og alle metoder i den ferdige kommunikasjonsmodulen. *Com_ImportFilter* definerer malen for et importfilter. *Com_ImportFilter_MediaWikiSpecialPages* er implementasjonen mot *Semantic MediaWiki* som er blitt beskrevet. *Com_ImportFilter_OWLFile* er et enkelt importfilter for å lese inn ontologier fra fil (OWL eller RDF format). Hovedklassen er en utvidelse av JPanel-klassen. Dette er fordi det skal være mulig å utvikle et eget brukergrensesnitt for hvert filter. JPanel brukes i Java til å lage grafiske elementer. Størrelsen er på JPanelet er definert i *Com_ImportFilter* klassen som håndterer kommunikasjonsmodulens grensesnitt i selve programmet.

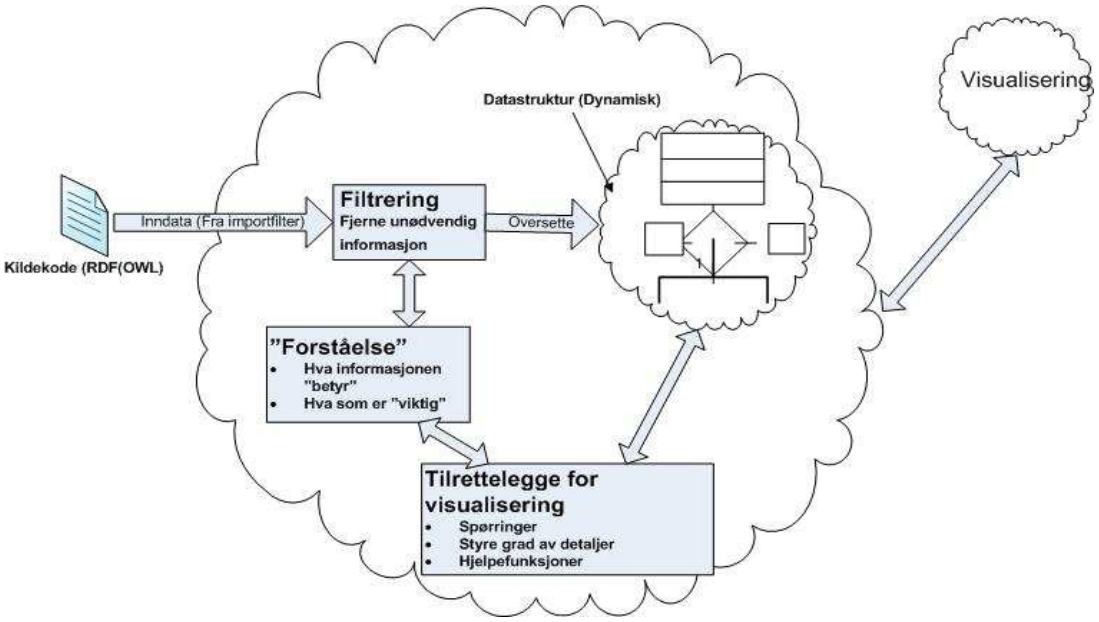


Figur 17. Class diagram - Kommunikasjonsmodul (Filter)

3.4.4. Programmodul: Tolk

De neste avsnittene tar for seg utviklingen av det som kan kalles programmets ”hjerne”, nemlig tolkmodulen. Tolkmodulen har som oppgave å tolke informasjonen som er hentet ned fra en wiki og tilrettelegge for visualisering av denne informasjon. Å ”tolke” betyr i denne sammenhengen at programmet er i stand til å ”forstå” informasjonen. Med ”tilrettelegging for visualisering” menes det at modulen skal utføre så mye som mulig av forarbeidet til visualiseringsdelen. Dette innebærer blant annet å bestemme hvilken informasjon som skal brukes i visualiseringen (detaljnivået) og bortfiltrering av irrelevant informasjon. I tillegg bør tolkmodulen oversette informasjonen som er blitt lastet ned til en dynamisk datastruktur slik at det blir mulig å hente ut målrettet informasjon, endre detaljnivå etter behov og utføre spørringer på ontologien strukturen beskriver.

Figur 18 viser en tidlig skisse for hvordan tolkmodulen vil løse disse oppgavene.



Figur 18. Skisse som viser første utkast til design av tolkmodulen

Utvidelser

For å få til dette ble tolkmodulen implementert etter samme prinsipp som kommunikasjonsmodulen: En abstrakt klasse ble brukt som mal og bestemte hvilke funksjoner som var nødvendige for en tolkmodul.

Forståelse

"Forståelse" er den viktigste funksjonaliteten tolkmodulen vil ha. I praksis betyr dette at programmet må forstå strukturen på dataene som er blitt lastet ned ved hjelp av kommunikasjonsmodulen. De foreløpige importfiltrene som er blitt utviklet for denne modulen støtter formatene RDF og OWL, derfor vil det være nødvendig å implementere en tolkmodul som forstår dette formatet. En del av denne "forståelsen" vil også innebære å oversette den nedlastede informasjonen til en tilsvarende datastruktur I en datastruktur vil det være mulig å lagre ontologien på en mer hensiktsmessig måte, slik at det blir mulig å hente ut spesifikke deler av ontologien, finne ut hvilke deler som er relaterte osv.

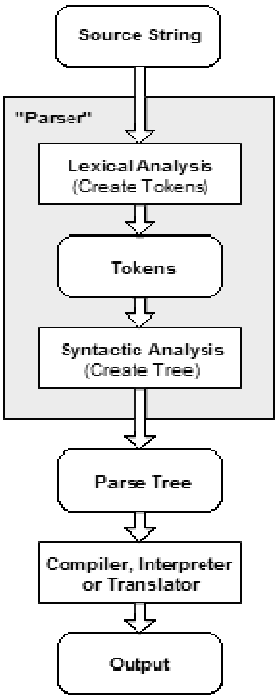
For å utføre denne oversettelsen, må programmet lese gjennom hele kildekoden og fange opp de syntaktiske sekvensene kildekoden består av. En slik analyse og oversettelse av kildekode kalles parsing:

Parsing

Definisjon:

> En parser er et program som leser inn inndata og identifiserer strukturen til denne i henhold til en gitt grammatikk. Parseren vil være i stand til å forstå strukturen i inndataene og identifisere eventuelle feil i denne *(Grune 1990; PARK 2008)*.
>
> I informatikk og lingvistikk, er parsing (også kalt syntaktisk analyse) en prosess som går ut på å analysere en sekvens av uttrykk for å bestemme deres grammatiske struktur med hensyn til en gitt grammatikk. En parser har som oppgave å transformere tekst inndata til en datastruktur, ofte i form av et tre, som er egnet for senere analyse. Prosessen innebærer også validering av koden *(Grune 1990; Wikipedia 2008).*

Fremgangsmåten for en parser vil som oftest være som følger: Første steget er en leksikalsk analyse. Dette steget skiller ut uttrykk som parseren kjenner igjen. Neste steget er syntaktisk analyse (parsing) som sjekker at uttrykkene er utformet på korrekt form (At alle nødvendige parametere er angitt, at riktig antall parenteser osv er gitt). Siste steg kalles semantisk parsing eller analyse. Dette steget vil utføre de funksjonene uttrykket beskriver. Dersom uttrykket er matematisk (Eks: 2 * 2), vil dette innebære utregning av det matematiske uttrykket. Alternativt kan oppgaven være å generere kode basert på det opprinnelige uttrykket (Wikipedia 2008). Figur 19 illustrerer den fullstendige fremgangsmåten for en parser.



Figur 19. Funksjonaliteten for en parser. URL: http://en.wikipedia.org/wiki/Parsing

Egenutvikling eller benytte eksisterende parsere?

Egenutvikling av parserdelen gir først og fremst fordelen av full kontroll over funksjonalitet og virkemåte. Det ville vært mulig å utvikle en parser som var skredderstyrt for bruk i dette programmet. Undertegnede har tidligere utviklet en enkel parser for et fiktivt programmeringsspråk som en del av kurset "*i125: Introduction to Program Translation*" ved Universitetet i Bergen. Erfaringen fra dette kurset er likevel at denne typen utvikling kan være svært tidkrevende, selv for svært enkle språk. RDF og spesielt OWL er langt mer omfattende enn det fiktive språket som ble benyttet i det overnevnte kurset. Spesielt vil det kreve lang tid og mye ressurser å utvikle en parser som er robust, det vil si at den er i stand til å fullføre oppgaven selv om kildekoden inneholder små feil eller mangler.

Ved å benytte eksiterende løsning, sparer man mye tid som i stedet kan brukes til å utvikle den delen av programmet som gjør det unikt i forhold til andre programmer samtidig som eksisterende og velprøvde løsninger vil være langt mer robuste og omfattende enn det som ville vært mulig med egenutvikling. Derfor ble det valgt å benytte en eksisterende løsning.

Valg av parser

Aktuelle parsere for prosjektet måtte støtte formatene som benyttes av importfiltrene som hittil er utviklet, det vil si OWL or RDF. Det var også et krav at parseren ikke var kommersiell ettersom programmet som utvikles er åpent. Parseren måtte dessuten være Java-kompatibel fordi Java er utviklingsspråket for programmet. Listen under beskriver noen av parserene som var aktuelle for prosjektet:

- **Jena:**

  Hjemmeside: (http://jena.sourceforge.net)

  Dette er en åpen kildekode parser som er utviklet i Java. Programmet er utviklet av Brian McBride fra Hewlett-Packard. Jena kan brukes til å parse, lage og endre RDF og OWL modeller (Ontologier) og støtter spørringer på disse modellene. Parsing kan skje enten fra kommandolinjen eller fra et annet javaprogram gjennom API'en programmet tilbyr (Verzulli 2001).

- **ICS-FORTH Validating RDF Parser (VRP)**

  Hjemmeside: http://www.ics.forth.gr/proj/isst/RDF/ , http://139.91.183.30:9090/RDF/VRP/

  Denne parseren kan parse RDF-dokumenter og validere dem mot et RDF Skjema. Den er utviklet ved Institute of Computer Science, Foundation of Research Technology i Hellas av Karsten Tolle. Mer informasjon på hjemmesiden i linken over.

- **ARP (Another RDF Parser)**

  Hjemmeside: http://www.hpl.hp.com/personal/jjc/arp/

  ARP er en RDF og XML parser for Java. Den er utviklet av Jeremy Caroll og kan parse RDF/XML dokumenter til såkalte RDF tripler. Den er utviklet for å bli brukt sammen med Jena, som er omtalt lenger oppe i listen.

- **Raptor Parser Toolkit:**

  Hjemmeside: http://librdf.org/raptor

  Et åpen kildekode bibliotek for programmeringsspråket C som tilbyr en rekke parsere, bl.a. for RDF/XML. Biblioteket kan også brukes av java-applikasjoner gjennom java-grensesnittet som tilleggsbiblioteket Redland RDF Libraries tilbyr.

- **Sesame/Rio RDF Parser**

  En Java RDF/XML parser som inngår i "Sesame Java toolkit." Sesame er et komplett, åpent Java-bibliotek for å håndtere RDF og XML (lagring, utføre spørringer osv). Rio er den delen

biblioteket som tar seg av parsing av slike filer, og er en rask og enkel parser for Java (Aduna 2008).

Basert på dokumentasjon på de ulike løsningenes hjemmesider og omtaler fra andre utviklere, som var tilgjengelig på nett, ble Jena valgt som parser. Dette var først og fremst fordi den fremstod som den mest omfattende og komplette løsningen. Parseren inneholder en god del avanserte funksjoner ikke vil være nødvendige for utviklingen, men som likevel kan være greit å ha tilgjengelige da det gir gode muligheter for videreutvikling på et senere tidspunkt. Jena støtter de mest brukte ontologispråkene, herunder XML, RDF og alle versjoner av OWL.

Tilrettelegging for visualisering

Denne oppgaven bestod i å utvikle en rekke hjelpemetoder som automatiserte og forenklet prosessen med å hente frem ulike deler av ontologien som var blitt lastet ned. Disse metodene jobbet direkte mot datastrukturen (Modellen) som parseren (Jena) oversatte den nedlastede kildekoden til. Metodene ble plassert i egne utility-klasser. De som er direkte relatert til Jena ble for eksempel plassert i *Utils_Jena* klassen. På denne måten kan de senere benyttes av andre deler av programmet, noe som kan være nødvendig i visualiseringsdelen. Eksempler på dette er *getSubClassList()* metoden som, gitt en ontologi-klasse, viser alle underklasser til denne (Figur 20). Som figuren viser, returneres klassene som en tabell (Java Array) av OntClass-objekter (En Jena objektstruktur for å lagre ontologi-klasser).

```
332     /** getSubClassList()
333      * Creates an array (OntClass[]) of the class' subclasses.
334      * If the class has no subclasses, "null" is returned
335      * @param aClass The class to extract the subclasses from
336      * @return An array of the class' subclasses, or "null" if the class
337      *         has no subclasses */
338     public static OntClass[] getSubClassList(OntClass aClass){
339         if(aClass==null)return null;//missing parameter(s)
340         List<Object> array=new ArrayList<Object>();
341         ExtendedIterator it=aClass.listSubClasses();
342         if(!it.hasNext())return null;//no subclasses
343         while(it.hasNext()){array.add(it.next());}//while
344         Object[] objArray=array.toArray();
345         OntClass[] ontArray=new OntClass[objArray.length];
346         for(int i=0;i<objArray.length;i++){
347             ontArray[i]=(OntClass)objArray[i];
348         }//for
349         return ontArray;
350     }//getSubClassList()
```

Figur 20. Hjelpemetoden getSubClassList() (Fra Utils_Jena class-filen)

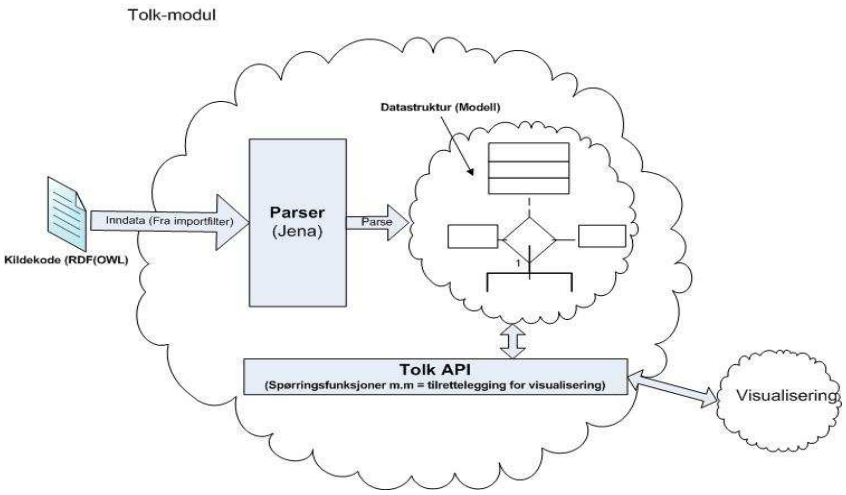
Resten av metodene vil være implementert i kildekoden for tolkmodulen: *App_Engine*. Et eksempel på det siste er metoden for å laste inn og parse en RDF eller OWL-fil (enten fra filsystem eller fra en URL). Figur 21 viser hvor enkelt dette gjøres ved hjelp av Jena. Metoden *loadOntology()* forenkler

denne prosessen ytterligere ved å kun kreve adressen til kildekoden som parameter og returnerer en boolean verdi som angir om ontologien ble lastet inn (returverdi = ”true”) eller om noe gikk galt (returverdi ”false”).

```
52      public boolean loadOntology(String source){
53          OntModel ret=ModelFactory.createOntologyModel();
54          InputStream in = FileManager.get().open(source);
55          if (in == null) {
56              System.err.println("File: "+source+" not found");
57              return false;
58          }//if
59          //Les inn RDF/XML filen:
60          ret.read(in,"");
61          if(ret!=null){
62              ontM=ret;super.setOntologyLoaded(true);return true;
63          }//load success
64          return false;
65      }//loadOntology()
```

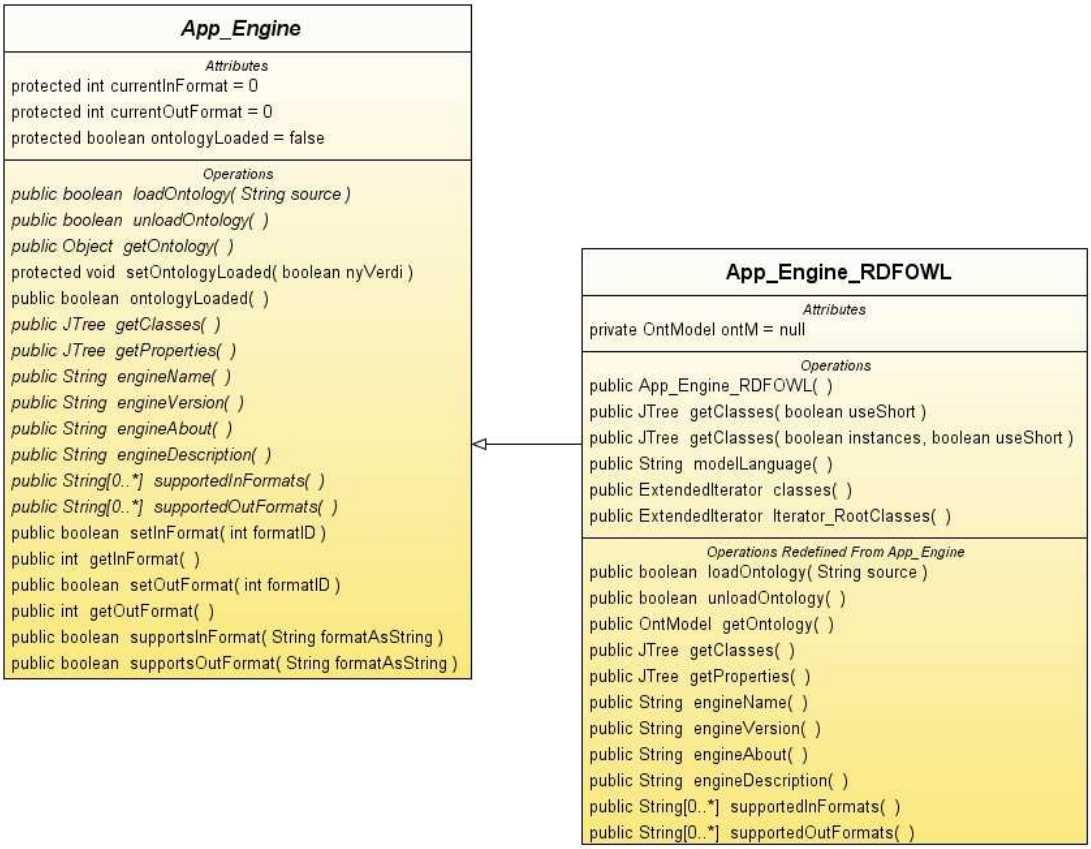
Figur 21. loadOntology() - Laste inn ontologi fra RDF/OWL vha Jena API

Figur 22 illustrerer den endelige, overordnede strukturen for tolkmodulen.



Figur 22. Skisse som viser endelig utkast til design av tolkmodulen

Figur 23 viser den endelige klassestrukturen for kildekoden den ferdige tolkmodulen består av. *App_Engine* en mal for hvordan en tolkmodul skal implementeres. Den faktiske implementasjonen ligger i klassen *App_Engine_RDFOWL* som er en utvidelse av *App_Engine*. ”RDFOWL” kommer av at denne tolkimplementasjonen støtter formatene RDF og OWL. Tolkmodulen er ellers en del av programmets hovedpakke[6] , *asProgram* (Derfor starter klassenavnet med ”App”).



Figur 23. Class diagram - Tolk modul (Engine)

[6] Java Package. Se forklaring under termer og begreper etter innholdsfortegnelsen

3.4.5. Programmodul: Visualisering

Visualiseringen er den siste funksjonelle modulen til programmet og har som mål å gi en visualisering av ontologien som er blitt lastet ned fra en wiki. En visualisering vil være en alternativ, høynivå fremstilling av ontologien. Målet er at visualiseringen skal gjøre det lettere å forstå sammenhengen mellom de ulike delene av ontologien. Det er derfor viktig å kunne gi et "sammendrag" av ontologien som gir et raskt overblikk. Hvilken metode som er "best" for å visualisere en ontologi vil være vanskelig å bestemme og avhenger både av den enkelte implementasjon og strukturen på ontologien som skal oversettes. På samme måte som de andre modulene vil også visualiseringsmodulen støtte muligheten for utvidelser ved at den definerer en generell mal for en visualisering.

Visualisering – Generell mal

Programmet vil støtte alle visualiseringer som følger en generell mal for en visualisering på samme måte som importfiltermalen fra kommunikasjonsdelen og tolkmalen fra tolkmodulen. Denne malen vil ha to hovedmetoder som må implementeres av alle visualiseringer: *draw()* og *clear()*. Når disse blir kalt fra programmet, skal visualiseringen tegnes opp (Dersom *draw()* ble kalt) eller fjernes (Dersom *clear()* ble kalt). Visualiseringen må dessuten ha mulighet til å kommunisere med tolkmodulen. Dette skjer ved hjelp av en egen metode, *getOntology()* som returnerer en referanse til den ferdig parsede strukturen som til enhver tid er lastet inn (Figur 24).

```
110    /** visualVersion()
111     * @return String representing the version of this visualization */
112    public abstract String visualVersion();//visualVersion()
113    /** visualDescription()
114     * @return a String describing in (some detail) this visualization  */
115    public abstract String visualDescription();//visualDescription()
116    /** visualAbout()
117     * @return String describing credits about the visualization
118     *          (Author, copyright, etc) */
119    public abstract String visualAbout();//visualAbout()
120    /** draw()
121     * Draws a visualization of the currently loaded model */
122    public abstract void draw();//draw()
123    /** clear()
124     * Clears the content of this JPanel (Or resets the visualization) */
125    public abstract void clear();//clear()
```

Figur 24. Visualisering: Abstrakte metoder i visualiserings-malen

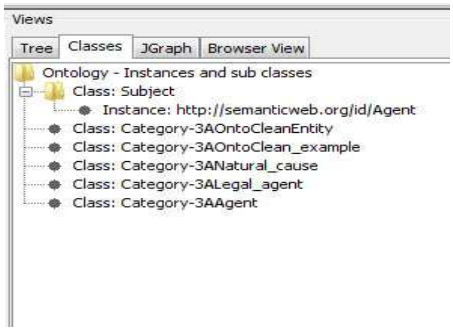
Visualiseringer (Implementasjoner)

TextTree

TextTree er en svært enkel, tekstbasert visualisering som bruker en trestruktur til å visualisere ontologiens egenskaper og struktur. Den ble ikke tatt med i det endelige programmet, men spilte en viktig rolle under utviklingen av visualiseringsmodulen. Den ble benyttet som hjelpemiddel for å utvikle og teste ut hjelpemetoder som senere ble brukt av de øvrige visualiseringene. Dette var først og fremst metoder for å hente frem spesifikke deler av ontologien, for eksempel klasser, instanser og egenskaper. Hjelpemetodene som ble brukt til dette er senere flyttet til egne "utility-klasser" (*Utils_Jena*, *Utils_Graph* m.fl). Fordi den var tekstbasert, var den enkel å tilpasse og endre under den tidlige utviklingen.

JTree (Tree og Classes/Triple)

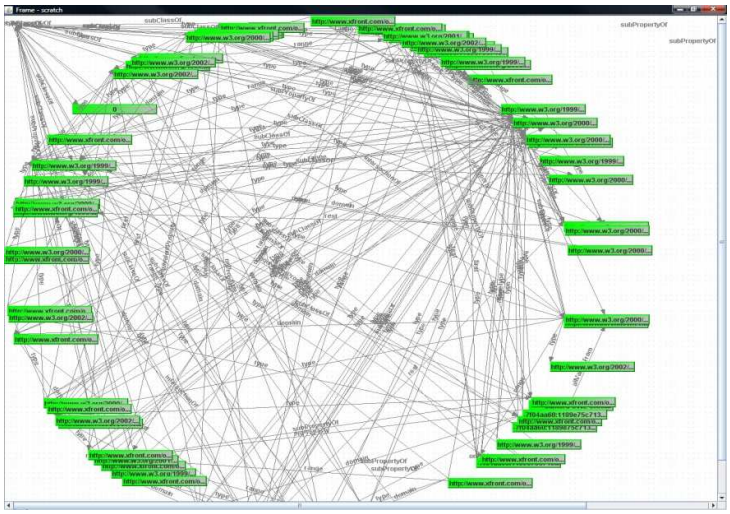
JTree-visualiseringen bruker også en trestruktur for å visualisere ontologiens hovedstruktur. Visualiseringen har lånt mange av egenskapene og metodene fra TextTree. Den benytter Javas JTree element for å vise trestrukturen grafisk. JTree er et grafisk element som kan vise en hierarkisk datastruktur av nodeelementer som en trestruktur. De enkelte delene av treet kan utvides eller kollapses ved behov. En slik utvidelse kan styres både gjennom programmet og gjennom interaksjon fra brukerens side. Det er mulig å starte med en overordnet struktur som brukeren selv kan utvide ved behov for å få frem detaljer. Det endelige programmet inneholder to implementasjoner basert på denne løsningen, "Tree" og "Classes". Tree-visualiseringen viser all informasjon som den nedlastede ontologien innholder, mens "Classes" (Figur 25) filtrerer vekk informasjon som anses som uvesentlig.



Figur 25. Screenshot: Visualisering - JTree/Classes

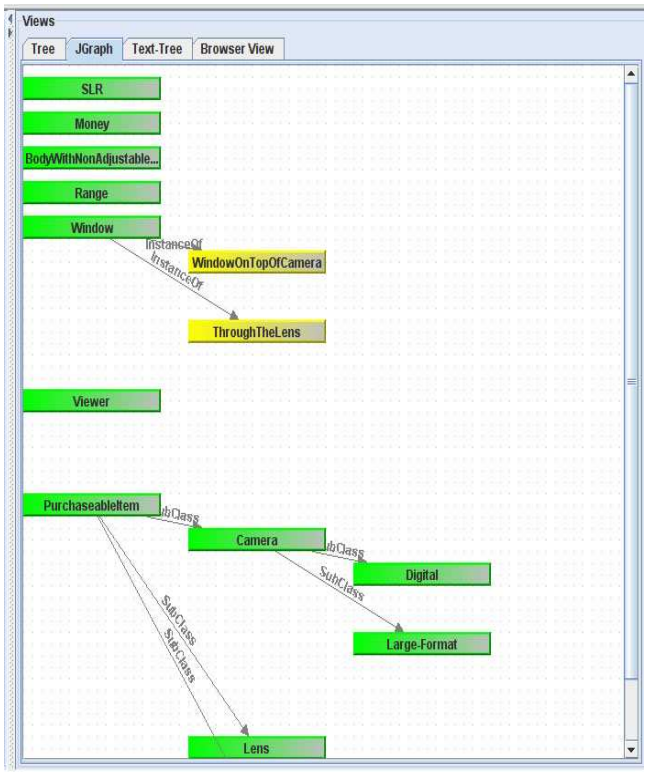
JGraph

Det var et mål at minst en av visualiseringene skulle ha en ”ekte” grafisk fremstilling som gav et godt og alternativt overblikk av ontologien. En grafstruktur ville være godt egnet for dette formålet. Den største utfordringen for å utvikle en slik løsning var å ”tegne” del ulike grafiske elementene (Noder, piler osv) en graf består av. For å gjøre dette arbeidet lettere ble JGraph tatt i bruk. JGraph er et rammeverk for å tegne og håndtere grafer ved hjelp av Java. JGraph gir store friheter til å komponere egne grafer og det er mulig å bestemme layout på grafene helt ned til minste detalj. En del utfordringer var det likevel med denne løsningen. De største problemene gjaldt posisjonering av nodene i grafen. Denne oppgaven kan utføres automatisk gjennom et kommersielt tillegg til pakken. Fordi programmet som skal utvikles er åpent, var det ikke aktuelt å benytte dette tillegget. Det ble derfor nødvendig å utvikle egne metoder for posisjonering av graf-elementene. Resultatet ble derfor noe enklere enn det som ville vært mulig med den kommersielle versjonen av programmet, men gir likevel en grei og oversiktlig, grafbasert visualisering. Det ble utviklet to ulike layout-metoder som kunne benyttes for visualiseringen. Den første og mest ambisiøse hadde som målsetning å fordele ontologien i sirkelformede ”skyer”. Hver av disse ville bestå av mindre skyer som representerte detaljinformasjon. Denne layouten ble til slutt forkastet, da det viste seg å være svært omfattende å utvikle en sorteringsalgoritme som fordelte de ulike skyene slik at de dannet et oversiktlig bilde av ontologien (Særlig viktig når ontologien inneholder mye informasjon). Figur 26 viser et utkast av denne layouten. Det store antallet elementer i figuren skyldes ikke at ontologien som er benyttet var spesielt stor, men at den visualiserer alt innholdet som Jena var i stand til å hente fra den. Det meste av dette er detaljinformasjon som ikke bør vises i en visualisering som skal gi et forenklet overblikk av ontologien.



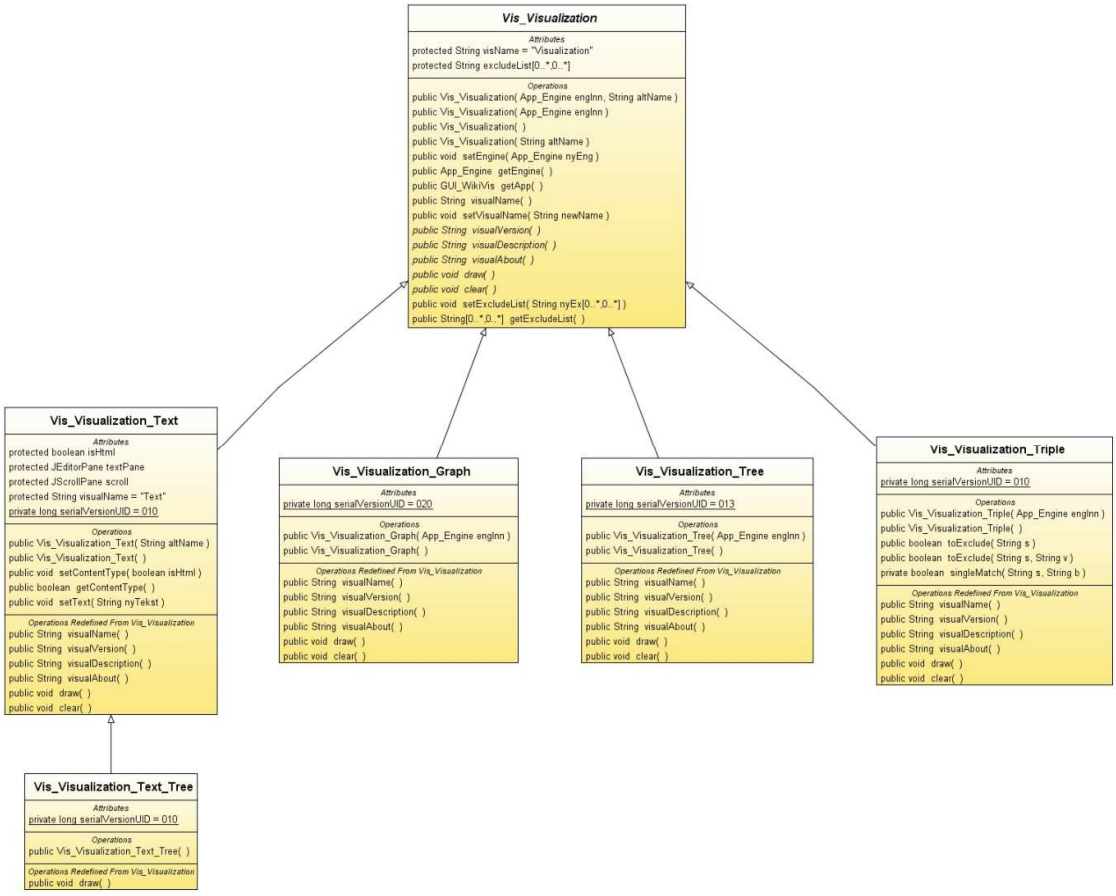
Figur 26. Screenshot: Visualisering - "Circle Layout"

Den endelige JGraph visualiseringen består av en enkel tre-lignende grafstruktur. Denne siler ut det meste av unødvendig informasjon som ”Circle-layout” visualiseringen viste og begrenser innholdet til klasser og instanser som den nedlastede ontologien inneholder. ”Circle-layout” visualiseringen kunne blitt tilpasset til å gjøre det samme, men det ville kreve mye arbeid å omplassere de ulike objektene slik at den ble oversiktlig. For mange kryssende linjer som i eksemplet over gjør det nærmest umulig å se hvilke objekter som er forbundet. Figur 27 viser den endelige layouten for JGraph visualiseringen.



Figur 27. Screenshot: Visualisering - JGraph

Figur 28 viser et klassediagram over kildekoden for visualiseringsmodulen. *Vis_Visualization* definerer hvilke metoder som må være med i en visualisering. De øvrige klassene er implementasjoner (visualiseringer). *Vis_Visualization_Text* og utvidelsen *Vis_Visualization_Text_Tree* to enkle, tekstbaserte visualiseringer som ble brukt under den innleden utviklingen. De er fjernet fra den siste versjonen av programmet ettersom innholdet overlapper de grafiske JTree baserte visualiseringene *Vis_Visualization_Triple* (I programmet kalt ”Classes”) og *Vis_Visualization_Tree*. Den siste visualiseringen er *Vis_Visualization_Graph* som visualiserer ved hjelp av en graf struktur. Denne graf-strukturen er laget ved hjelp av JGraph rammeverket.



Figur 28. Class diagram - Visualiseringsmodul

3.5. Oppsummering

Dette kapitlet har gjennomgått utviklingen og designet av WikiVis, en prototype av et verktøy for visualisering av wikiontologier. De ulike løsningene og metodene som er blitt benyttet har blitt forklart og gjennomgått. Programmet ble utviklet i Java. Foruten brukergrensesnittet ble den ulike funksjonaliteten fordelt på tre programmoduler: Kommunikasjon, Tolk og Visualisering. Disse modulene ble utviklet som selvstendige prosjekter.

Kommunikasjonsmodulen håndterer, som navnet antyder, kommunikasjonsdelen mellom program og wiki. Kommunikasjonen skjer ved hjelp av importfiltre. Et importfilter er en generell klasse som definerer funksjonene som må til for utføre kommunikasjonen med en bestemt wiki. Selv om hvert importfilter blir utviklet på sin egen måte, kan de styres på samme måte av hovedprogrammet. Dette gjør det mulig å skifte ut eller legge til filtre i ettertid uten å måtte endre resten av programmet. Det ble implementert to slike importfiltre: ”Hovedfilteret” kommuniserer med wikier som kjører Semantic MediaWiki programvaren. Til dette benyttes wikiens webbaserte eksportfunksjonalitet. Det andre filteret kan laste ned filer fra det lokale filsystemet, i RDF eller OWL format. Gjennom denne modulen er kravene 1 og 2 oppfylt. Krav 7, som går ut på å støtte flere wikier er støttet gjennom designet: Det er mulig å utvikle nye importfiltre som kan håndtere andre wikier uten at dette får noen innvirkning på resten av programmet. Det er derimot bare implementert støtte for en wiki (Semantic Mediawiki). I tillegg er støtte for RDF/OWL filer implementert på samme måte.

Tolk modulen tar seg ”forståelsen” av innholdet som er blitt lastet ned. Også tolkmodulen kan byttes ut på samme måte som et importfilter. Tolkmodulen som ble implementert benytter Jena som parser for å ”forstå” innholdet i koden som er lastet ned og støtter RDF og OWL formatene. Denne modulen står for implementeringen av krav 3.

Visualiseringsmodulen håndterer visualiseringsdelen av programmet. Som de andre modulene er også visualiseringene mulige å skifte ut. Tre visualiseringer ble implementert og tatt med i programmet: To enkle, trebaserte visualiseringer og en mer omfattende, grafbasert visualisering. Den siste benyttet JGraph rammeverket. Visualiseringsmodulen oppfyller krav 5 om at programmet må kunne visualisere en ontologi. Krav 4 (”siling” av informasjon) styres av den enkelte visualisering, men metodene for å hente ut den nødvendige informasjonen, er implementert i tolkmodulen. Krav 9 (redigering) er ikke implementert. For å implementere en slik funksjonalitet vil det være nødvendig å oppdatere både visualiseringer og importfilter, slik at de kan støtte dette.

Chat-funksjonen som ble foreslått i kravlisten er ikke blitt implementert. Denne funksjonen kunne vært greit å ha, men er ikke en obligatorisk funksjon. Implementasjon av denne funksjonen ville tatt

for mye tid i forhold til gevinsten (i form av ekstra funksjonalitet) og det har vært nødvendig å prioritere andre og viktigere funksjoner. Mulighet til å lagre innstillinger og lignende (krav 10) er delvis implementert ved at innholdet i ekskluderingslisten (liste over verdier som ikke skal tas med i visualiseringen) lagres og hentes fra fil. Ellers har det ikke vært behov for å implementere denne funksjonaliteten da det først og fremst er en prototyp. Tabell 6 oppsummerer hvilke målsettinger (krav) som er blitt oppfylt under utviklingen.

Tabell 6. Oppfylte krav (Funksjonelle og ikke-funksjonelle)

	Nr	Oppfylt	Type	Krav	Modul	Kommentar
Funksjonelle krav	1	Ja	Må	kunne kommunisere med en wiki	Kommunikasjon	Støttet: Semantic Mediawiki
	2	Ja	Må	kunne laste ned en ontologi	Kommunikasjon	Vha RDF export
	3	Ja	Må	kunne "forstå" ontologien	Tolk	Vha Jena parser
	4	Ja	Må	kunne "sile" informasjon	Tolk/Visualisering	
	5	Ja	Må	kunne visualisere	Visualisering	Tre visualiseringer
	6	Nei	Bør	ha en chatfunksjon		
	7	Ja	Bør	støtte flere wikier	Kommunikasjon	Men bare en er implementert
	8	Ja	Bør	være utvidbart	Alle	
	9	Nei	Bør	tillate redigering		Kan legges til i ettertid
	10	Delvis	Bør	kunne lagre innstillinger m.m		Kun ekskluderingsliste
Ikke-funksjonelle krav	11	Ja	Må	være åpent		
	12	Delvis	Bør	være intuitiv i bruk	GUI	En del problemer m/valg av side som visualiseres
	13	Delvis	Bør	være optimert		Kan bli bedre
	14	Ja	Bør	være stabilt		
	15	Delvis	Bør	gi gode visualiseringer	Visualisering	JGraph best, men kan bli bedre

Resultatet av utviklingen er en prototyp som er i stand til å utføre de viktigste funksjonene for et slikt visualiseringsverktøy. Programmet kan koble seg til en wiki (Semantic MediaWiki), laste ned en liste over tilgjengelige sider og visualisere disse. Programmet er utviklet med tanke på at det skal være mulig å utvikle det i ettertid.

Gjennom utviklingen av WikiVis er det ene målet for prosjektet nådd: Å utvikle et verktøy for visualisering av wikiontologier. Foruten å vise at et slikt verktøy kan utvikles, har utviklingen av denne prototypen vært med på å øke forståelsen av hvordan et slikt program kan eller bør utvikles. I neste kapittel vil programmet bli evaluert for å besvare problemstillingen for prosjektet: Kan et slikt verktøy forenkle utviklingen og bedre forståelsen av wiki-ontologier?

4. Evaluering

Dette kapitlet beskriver hvordan evalueringen av programvaren som er blitt utviklet er gjennomført og resultatene av denne. Først blir de aktuelle metodene som ble presentert i kapittel 2 drøftet og konkrete metoder blir valgt blant disse. Deretter beskrives planleggingen og utføringen av testene og til slutt resultatene. Drøfting av resultater og konkludering blir gjennomgått i kapittel 5.

4.1 Formål

Evalueringen vil ha flere formål:

- Vurdere programmets ”Usability”: Denne delen går ut på å evaluere hvordan selve programmet fungerer. Eksempler på spørsmål denne delen vil forsøke å besvare er: Fungerer brukergrensesnittet tilfredsstillende? Er alle nødvendige funksjoner implementert? Gjør programmet de oppgavene det er ment å utføre?

- Besvare prosjektets problemstilling: Vil et verktøy for visualisering av wiki-ontologier føre til forenklet utvikling og bedre forståelse av slike wikier? Er det behov for et slikt program? Testpersonene vil bruke en prototyp på et slikt verktøy. Deres vurdering av nytteverdien av programmet og i hvilken grad det forbedrer eller kan forbedre forståelse og utvikling vil bli brukt til å komme frem til en konklusjon på denne problemstillingen.

- Få innspill til forbedringer og endringer i programmet.

4.1 Metode:

Evalueringen kan enten utføres på eksperter eller på representative brukere av systemet. Eksperter vil i denne sammenhengen være utviklere eller andre med spesiell kjennskap til testing av programvare. Den største fordelen med ”expert usability review” evaluering er at metoden er billig. Billig fordi den kan utføres av utviklerne selv, slik at det ikke er nødvendig å bruke tid og ressurser til å rekruttere andre testere, nøye planlegging og avtaling av tid til gjennomføring. Eksperter har gjerne bedre forståelse over hvilke krav og funksjoner som vil være nødvendige for det ferdige produktet. Problemet med denne typen tester er at ”eksperter”, særlig om det er utviklerne selv, ikke vil være i stand til å finne alle feil, nettopp fordi de som utviklere har vært med på å utvikle de ulike funksjonene og derfor vet hvor de er og hvordan de skal brukes. For dette prosjektet vil det dessuten være en begrensning at det bare er én utvikler. Det vil derfor være lite å hente på en slik test. Det er også lite å hente til de andre formålene med evalueringen, som å besvare problemstillingen. En usability test vil derfor være å foretrekke.

En usability test skal måle både bruken av programmet (For eksempel effektivitet eller nøyaktighet) og emosjonell respons (hvordan brukeren oppfatter systemet). For å måle bruken av programmet vil

observasjon være best egnet, mens en annen form for datainnsamling (intervju eller spørreskjema) kan brukes til å fange opp emosjonell respons. Observasjon med og uten høyt-tenkning er blitt nevnt som aktuelle metoder i kapittel 2. Det ble valgt å benytte observasjon uten høyttenkning til dette formålet. Årsaken til dette er at høyttenkning kan være et forstyrrende element, både for den som blir testet (som må forklare alt han gjør) og den som utfører testen (som må følge med på både det brukeren gjør på skjermen og det han sier). Subjektive meninger og forklaringer kan i stedet bli avklart gjennom den etterfølgende datainnsamlingen.

Som aktuelle datainnsamlingsmetoder for å fange opp emosjonell respons, samt å besvare problemstillingen er spørreundersøkelse og intervju aktuelle metoder. Disse metodene har ulike krav til antall personer som bør være med i testen. Spørreskjema har den fordelen at resultatene er lette å sammenligne. På den negative siden krever disse nøye planlegging for å fange opp så mye som mulig informasjon, samtidig som den ikke vil fange opp nye problemstillinger som testpersonen kan komme opp med. Det er heller ikke mulig å komme med forklaringer og oppklaringer av utydelige spørsmål. Derfor bør det brukes et høyt antall testpersoner for å få gode resultater. Intervjuer er langt mer fleksible. De krever ikke så mye planlegging, likevel er det mulig å fange opp langt mer informasjon for hver enkelt testperson. Det vil derfor ikke være nødvendig å utføre så mange tester som ved bruk av spørreskjema. På den negative siden kan det være vanskeligere å sammenligne resultater fra de ulike testene fordi de ikke gir så entydige svar som et spørreskjema. På tidspunktet da testen skulle utføres, var det forholdsvis liten tid igjen av prosjektet. Det var derfor viktig å få gjennomført testene på kort tid, uten for mange testere og samtidig få maksimalt ut av hver test. Fordi programmet utforsker et felt som de fleste ikke har så god kjennskap til, vil det være en fordel å kunne assistere evalueringen og gi råd underveis. Også her er intervju bedre egnet enn spørreskjema. Hadde det vært bedre tid til disposisjon, hadde den beste løsningen kanskje vært å bruke begge løsningene, men da dette ikke er tilfelle ser intervju ut til å være det klart beste alternativet.

Konklusjonen blir derfor at evalueringen skal utføres i form av en usability test der brukerne får benytte programmet. Det vil bli laget et sett med oppgaver som testpersonene skal utføre ved hjelp av programmet, mens de blir observert. Observasjonen vil bli kombinert med et intervju som skal supplere informasjonen fra observasjonen. Intervjuet vil bli brukt både til å avklare meninger om programmet, hvorfor ulike avgjørelser ble tatt under oppgavedelen, og til å besvare spørsmål relatert til prosjektets problemstilling.

4.1.1 Detaljer for valg av intervju

Det finnes i hovedsak to typer intervju metoder: *Strukturert* og *semi-strukturert*.

Strukturerte intervjuer er intervjuer med et fast oppsett av spørsmål. Et slikt intervju vil ha korte og presise spørsmål. På mange måter minner dette om en assistert spørreskjemaundersøkelse. Man kan få mer utfyllende svar enn ved bruk av et spørreskjema, men uten at svarene blir for overlappende.

Det andre alternativet er semi-strukturert intervju (D'arcy 1990; Barrio 1999). Semi-strukturert intervju er en av de mest brukte kvalitative metodene. Et semi-strukturert intervju er et åpent, individuelt intervju hvor målet er å få fange opp så mye som mulig av intervjuobjektets tanker omkring et bestemt emne eller praktisk oppgave. Mens et strukturert intervju har et formalisert og begrenset sett med spørsmål, er det semi-strukturerte intervjuet fleksibelt ved at det tillater at man tar opp nye spørsmål underveis i intervjuet.

Fordi det kan dukke opp interessante og nye problemstillinger underveis, vil nok semi-strukturert intervju være å foretrekke. Denne typen intervju har større mulighet til å fange opp ny informasjon og kan tilpasses den enkelte testkandidat slik at man får mest mulig informasjon fra hver av dem.

Et manuskript (mal) blir brukt som utgangspunkt for intervjuet, men avhengig av hva intervjuobjektet svarer på de ulike spørsmålene, forsøker intervjueren å komme med nye spørsmål. På denne måten forsøker man å følge tankegangen til hvert enkelt intervjuobjekt. De enkelte intervjuene kan dermed ende opp med å bli svært ulike.

4.2 Testpersoner

4.2.1 Kriterier

Valg av testpersoner er vitalt for suksessen av ethvert eksperiment. Under evalueringseksperimenter bør testpersonene bli valgt i samsvar med personer som i høyest mulig grad samsvarer med forventede brukere av det ferdige systemet (Dix 1993).

For ”folk flest” er ontologier og wikiontologier et ukjent begrep. For å få mest mulig saklig og god tilbakemelding vil det være en fordel om testpersonene har en viss idé om hva dette fagfeltet handler om. Det vil selvsagt være nødvendig å gi en innledende forklaring av hva ontologier generelt er og hvilken funksjon de har i datasammenheng.

Andre mastergradstudenter, helst innen informatikk eller informasjonsvitenskap vil være et godt utgangspunkt for å få saklige og gode tilbakemeldinger. I den grad de kan kalles det vil dette være det nærmeste en kommer såkalte ”Ekspert brukere”.

Metodene som er valgt for oppgaven er observasjon og intervju. Dette er metoder som kan gi mye informasjon for hver person som blir testet. Det blir mulig å gi grundige utspørringer og gå i dybden dersom det dukker opp interessante problemstillinger, herunder nye problemstillinger. Fordi man kommuniserer direkte med testpersonene kan dessuten alle uklarheter oppklares, enten det er testpersonen som ikke forstår et spørsmål eller en oppgave, eller den som ufører testen som ikke forstår en handling (under observasjonen). Dette gjør at det ikke vil være nødvendig med så mange tester som for eksempel en spørreundersøkelse ville krevd. Antall personer som kan benyttes avhenger også av tid, både til å få tak i nok testkandidater, til å gjennomføre testene og ikke minst til å gjennomgå og analysere resultatene.

4.2.2 Rekruttering:

Et alternativ til rekruttering er å sende ut en forespørsel, for eksempel gjennom en e-post til alle masterstudenter. Av personlig erfaring kan slike e-poster ofte bli oppfattet som spam og med mindre det blir brukt en eller annen form for premiering av deltakelse (Gratis pizza, betaling eller lignende) vil de færreste føle seg forpliktet til å delta. Det er også vanskelig å vite hvor kvalifiserte personene som eventuelt blir rekruttert er.

Et annet alternativ til vil være å håndplukke aktuelle kandidater.

Fordi det på dette tidspunktet er begrenset tid til disposisjon, blir det ikke mulig å benytte flere enn nødvendig. Derfor ble det viktig at de som ble valgt var så godt egnet til å utføre evalueringen som mulig. Håndplukking basert på erfaring var derfor beste alternativ.

Basert på disse kriteriene ble fire personer rekruttert til å delta i testen. Alle var bekjente (i mer eller mindre grad) som ble håndplukket på grunnlag av erfaringer og kompetanse. To av dem (A og B) er masterstudenter ved UiB (Informasjonsvitenskap), en av dem(C) jobber som selger ved dataavdelingen i en elektronikkforretning og den siste (D) en er en tidligere informasjonsvitenskapsstudent som nå jobber for med IT-support innen Bergen kommune.

Det var en viss form for variasjon innen gruppen: Enkelte hadde drevet med utvikling selv, og hadde dermed en viss forståelse av hva som ville være viktige tilbakemeldinger. Andre var mindre kjent med dette, men hadde god erfaring i bruk av programmer og ulike brukergrensesnitt. Dette ble en ”sunn blanding” av testpersoner med litt ulik, men god kompetanse.

4.3 Plan for gjennomføring av testen:

Hver test vil bestå av en usability test med observasjon og et semi-strukturert intervju. Testpersonene vil bruke prototypen til å løse reelle oppgaver programmet er ment å utføre. Under denne delen er det meningen at testpersonene i størst mulig grad skal løse oppgavene selv, uten veiledning (Slik det vil være tilfelle i en reell brukersituasjon). Måten testpersonene løser oppgavene på vil bli observert og kartlagt. Til kartlegging kan en bruke lydopptak, videoopptak eller notater. Problemet med lydopptak er at det vil være vanskelig å synkronisere disse med det brukeren gjør på skjermen. Videoopptak kan være litt uvant eller få enkelte testpersoner til å føle seg "presset" i tillegg til at det krever en del utstyr og ekstra planlegging. Det enkleste alternativet, notater, ble derfor valgt til å dokumentere observasjonen.

Når den praktiske testen er ferdig vil det bli utført et kort, semi-strukturert intervju med testpersonene. Intervjuet skal kartlegge erfaringene testpersonene fikk med programmet: Gjør programmet den oppgaven det er ment å gjøre? Er det noe som burde vært annerledes? Ser de nytteverdien i et slikt program? Hele testen bør ikke ta mer enn 10-15 minutter å gjennomføre.

Her følger planen som ble brukt for testen. Planen for del 2 (semi-strukturert intervju) er først og fremst en mal for hvordan intervjuet skal foregå. Gjennomføringen av de enkelte intervjuet vil følge denne malen, men vil bli tilpasset svarene og resultatene fra observasjonen. Dersom nye og interessante problemstillinger skulle komme frem, vil disse også bli gjennomgått for de kandidatene det måtte gjelde.

4.3.1 Del 1 Usability test: Løse oppgaver under observasjon

Bli kjent med programmet

- Fortelle om / gi innføring i programmet og dets funksjoner (Programmets hensikt, hva det kan gjøre, hva det ikke kan gjøre, begrensninger osv)
- La testpersonen "leke med programmet" for seg selv og finne ut av det. Observere og notere hva testpersonen gjør. Gjør han det riktig? Har han noen kommentarer?

Oppgave 1: Koble til en wiki og visualisere en artikkel

Mål: Finne ut om de programmets viktigste funksjonalitet (Tilkobling og visualisering) fungerer bra og intuitivt.

Wikien som ble benyttet til dette formålet var Semantic Web (http://www.ontoworld.org). Semantic Web er en portal som publiserer informasjon om utvikling relatert til Semantic Web og Wikier. Siden kjører Semantic MediaWiki og inneholder en enkel eksempelontologi, og er dermed kompatibel med WikiVis. Ontologien er relativ enkel og inneholder ikke så mye informasjon, men siden er likevel et godt utgangspunkt for å teste programmet. På den første oppgaven var det ikke nødvendig å fylle ut servernavn, da programmet var satt opp til å vise denne siden som standard ved oppstart.

Spørsmål som bør kunne besvares gjennom observasjonen:

- Klarer brukeren dette på egenhånd?
- Er måten tilkobling skjer på logisk?
- Er måten man velger og visualiserer artikler logisk?

Oppgave 2: Forsøk å koble til en wiki som ikke støttes av programmet

Mål: Teste om feilmeldinger er forståelige og at brukeren skjønner hvorfor denne typen tilkobling ikke er mulig. Til dette formålet vil Wikipedia (http://wikipedia.org) bli benyttet.

Spørsmål som bør kunne besvares gjennom observasjonen:

- Koble til Wikipedia. Hva skjer?
- Forstår brukeren hvorfor dette ikke fungerer?

Oppgave 3: Åpne en OWL/RDF fil.

Mål 1: For å gjøre dette må brukeren velge et annet filter enn standardfilteret. Er denne delen logisk implementert?

Mål 2: Finne ut om filteret for åpning av RDF/OWL fungerer tilfredsstillende.

Spørsmål som bør kunne besvares gjennom observasjonen:

- Klarer brukeren selv å finne ut hvordan dette skal gjøres?
- Gjør han det på riktig måte?

Oppgave 4: Endre listen over tilgjengelige sider.

Mål: Finne ut om innstillingsfunksjonen er godt nok implementert og lett tilgjengelig.

Forklar at dette krever endringer i innstillingene.

Spørsmål som bør kunne besvares gjennom observasjonen:

- Finner brukeren frem til riktig innstilling?
- Klarer han å endre dem slik han skal?
- Klarer han så å få frem den nye listen og visualisere?

4.3.2 Del 2: Semi-strukturert intervju

Det semi-strukturerte intervjuet vil som nevnt bestå av et sett med grunnspørsmål (kategorier). Disse vil bli brukt som en mal, og vil ikke bli fulgt slavisk. Hvor mange av de enkelte spørsmålene og hjelpespørsmålene som blir besvart vil avhenge av den enkelte kandidaten som deltar i evalueringen. For enkelte av intervjuene kan det bli aktuelt å stille andre spørsmål enn dem som er med i malen dersom det skulle dukke opp nye og interessante problemstillinger. Malen som vil bli brukt som utgangspunkt er som følger (Fordelt på hovedspørsmål og hjelpespørsmål):

- Brukergrensesnittet
    - Hva er ditt generelle inntrykk av brukergrensesnittet?
    - Hvordan fungerte måten tilkoblingene skjedde på?
    - Hvordan fungerte måten man visualiserte en side?
    - Ga hjelpefunksjonen gode tilbakemeldinger? (Eks: Feilmeldingen som ble vist ved tilkobling til Wikipedia)
- Visualisering
    - Hva sier visualiseringene?
    - Forstår du hva de viser?
    - Hvilken av de ulike visualiseringene gir best informasjon?
- Forståelse og nytteverdi

  - Forstod du hensikten til programmet?
  - Oppnår programmet det som er hensikten?
  - Gir den en bedre oversikt over ontologien enn det som ville vært mulig uten programmet?
  - Hvilken nytteverdi har programmet?
  - Har det nytteverdi i det hele tatt
  - Vil det kunne bli et viktig verktøy dersom det blir videreutviklet?
- Annet
  - Hva vil du fremheve som positivt med programmet?
  - Hva vil du fremheve som negativt med programmet?
  - Hva kan gjøres annerledes?
  - Nye funksjoner?

4.4 Evalueringene

De neste avsnittene vil gi en kort gjennomgang av de fire evalueringene som ble utført og vil bli etterfulgt av en oppsummering av de samlede resultatene. Evalueringene ble utført i henhold til planen som ble presentert i forrige avsnitt.

4.4.1 Evaluering 1

Kandidat A.

Masterstudent i informasjonsvitenskap, Universitetet i Bergen

Jobber som PC-veileder ved universitetet.

Har blant annet erfaring om semantisk web fra egen masteroppgave.

Brukergrensesnittet (Observasjon og intervju)

Oppgave 1. Oppkobling og visualisering

Oppkobling går greit. Kandidaten forstår at han må bruke ”connect” knappen først. I det tilkoblingen er ferdig legger han merke til at fargen på ”server-feltet” skifter til grønt og forstår at han nå er tilkoblet. Han klikker så på en av artiklene fra listen for å visualisere. Her trykker han først en gang (enkeltklikk) uten at noe skjer (annet enn at artikkelen blir merket). Han trykker så ”connect”, men skjønner raskt at det ikke var det han skulle. Han forsøker så å dobbelklikke. Her kommenterer han at det burde vært nok med et enkeltklikk.

Visualiseringen går greit. Det tar litt tid før den kommer opp, og brukeren lurer en stund på om han har gjort riktig, før valg av visualiseringer som skal brukes vises.

Da kandidaten under intervjudelen ble bedt om å kommentere brukergrensesnittet var første kommentar at ”*Det er ikke naturlig å dobbelklikke på lister. Det skal bare være mulig å ”highlight’e” alternativene* .” Han kommenterte videre at det burde vært en OK-knapp eller lignende til dette. Dette kom egentlig som en liten overraskelse på undertegnede, som hadde brukt en del tid på å få til overstyring i listen slik at enkeltklikk markerte og dobbelklikk visualiserte. Ellers bekreftet intervjuet det som observasjonen gav uttrykk for, nemlig at resten av oppkoblingsdelen var grei og intuitiv. Spesielt syntes han det var smart at feltet hvor man fyller ut servernavn skiftet farge fra rødt til grønt etter tilkobling. Dette var en enkel måte å vise at tilkoblingen hadde funnet sted.

Oppgave 2: Wiki som ikke støttes

Kandidaten ble bedt om å visualisere en side fra WikiPedia. Han koblet seg til og fikk lastet ned en liste over tilgjengelige sider. Igjen forsøkte han først å enkeltklikke for å visualisere, før han gjorde det på riktig måte (dobbelklikk). En feilmelding som forteller at siden ikke kan visualiseres kom opp. Kandidaten leste gjennom feilmeldingen og kunne konstatere at den

gav grei tilbakemelding om hvorfor wikien ikke var støttet. I intervjudelen hadde kandidaten ikke så mye å si om denne delen annet enn at det var greit. På spørsmål om hva han mente om feilmeldinger og tilbakemeldinger svarte han at de var helt greie.

Oppgave 3: Åpne fil

Denne delen gikk helt problemfritt. Kandidaten hadde fått forklart på forhånd at ulike tilkoblingsmåter var organisert i "importfilter" og at disse kunne velges fra en dropdown meny. Kandidaten valgte fil-filteret og åpnet eksempelfilen "Camera.owl" som beskriver en ontologi over fotoutstyr.

Oppgave 4: Endre listen over tilgjengelige sider

Endre innhold i liste: Fra og med "D". Dette gikk greit. Brukeren åpnet "list settings" dialogboksen og fant selv ut hvor han skulle skrive inn "Start from". Han huket av boksen for å begrense søket til dette. Listen blir oppdatert og han så at funksjonen fungerte som ventet.

Alt i alt gav kandidaten uttrykk for at brukergrensessnittet var greit. Det som opp til flere ganger ble tatt frem som det mest problematiske, var bruken av dobbelklikking for å velge side som skulle visualiseres. Ellers fungerer både visualisering, tilkobling og innstillinger greit.

Forståelse av programmets oppgave og nytteverdi

Kandidaten mener at det er litt vanskelig å forstå hva som blir visualisert. Det burde vært mulig å vise mer informasjon. Visualiseringen burde også vært koblet mot teksten i wikien. Visualiseringen fungerer, men kan fortsatt bli mye bedre. Det viktigste er nok å legge til enda mer informasjon og samkjøre med wikiteksten. En del av forståelsen vil nok likevel være knyttet til den enkelte wiki og hvor mye informasjon som skal vises.

Kandidaten sier han forstår programmets hensikt. Til å begynne med var det litt uklart, men etter å ha visualisert kamera-ontologien (Fil) forstod han litt mer av potensialet og funksjonaliteten. Verktøyet kan bli viktig og nyttig dersom det blir videreutviklet.

Annet – Forlag til forbedringer

Ekskluderingslisten fungerer greit. Den siler ut unødvendig informasjon som den skal og gjør visualiseringen langt mer oversiktlig enn den ville vært uten. Programmet burde hatt en timeglass-funksjonalitet som indikerer når programmet arbeider.

4.4.2 Evaluering 2

Kandidat B.

Masterstudent i informasjonsvitenskap, Universitetet i Bergen

Jobber som PC-veileder ved universitetet.

Brukergrensesnittet (Observasjon og intervju)

Oppgave 1. Oppkobling og visualisering

Litt usikker på hvordan det skal gjøres. ”*Ser at det står server, hm…*”. Han forsøker først ”go-knappen” før han velger ”connect”.

Etter oppkobling merker han en wikiside fra listen og forsøker å visualisere ved hjelp av ”go-knappen”. Resultatet er at en annen side (Forhåndsutfylt av programmet) blir visualisert. Han blir derfor gjort oppmerksom på at han må dobbelklikke.

Under intervjuet gav kandidaten klart uttrykk for at han mente tilkoblingsdelen burde vært noe annerledes. Kandidaten fortalte at han mente plasseringen av knapper er ulogisk, men poengterte også at dette var hans subjektive mening. ”*Spesielt gjelder dette plasseringen av go-knappen. Bruk av dobbelklikking er heller ikke bra for å kunne visualisere sider fra listen.*” Videre sa han at ”*Tilkoblingsdelen burde være en separat del, atskilt fra listen over wikisider. Når en velger en side fra listen burde den bli lagt til i ”go-feltet” og man burde så bruke ”go-knappen” for å visualisere denne”.*

Da visualiseringen ble vist var første kommentar: ”*Hva sier den?*”. Det ble nødvendig å gi en forklaring før han forstod dette. Under intervjudelen gav han uttrykk for at visualiseringen var grei når han forstod hva den gikk ut på. JGraph var den mest oversiktlige visualiseringen. Ellers mente han at Class-diagrammet nederst til venstre er litt overflødig. ”*Viser ikke den akkurat samme informasjon som Class-visualiseringen?*”.

Oppgave 2: Wiki som ikke støttes

Kandidaten ble bedt om å visualisere en side fra WikiPedia og fikk opp feilmeldingen som forteller at det ikke går. ”*Den fungerer ikke fordi Wikipedia ikke har Semantic Mediawiki”*.

Dette var etter kandidatens mening en helt logisk feilmelding som forklarte problemet.

Oppgave 3: Åpne fil

Kandidaten forsøker å åpne ved hjelp av ”File-open”, dvs. visualiserer på nytt. Her må han forklares at han først skal bytte til ”File-importfilteret”. Denne delen ble ikke tatt opp i intervjuet men viser at brukergrensenittet kan bli bedre også her.

Oppgave 4: Endre listen over tilgjengelige sider

Kandidaten ble bedt om å endre listen over tilgjengelige sider slik at den startet på ”D”. Dette gikk problemfritt. I intervjuet fortalte kandidaten at han syntes akkurat dette var greit, men han forstod ikke hva de andre innstillingene (som ikke ble brukt) betydde. ”*Hva betyr for eksempel API?*” var et av spørsmålene han stilte. Her burde det etter kandidatens mening være bedre forklarte tekster eller hjelp.

Forståelse av programmets oppgave og nytteverd

Kandidaten sa han forstår hensikten og at han mener at programmet løser oppgaven, selv om det kan bli bedre. For vanlige brukere er programmet kanskje ikke så nyttig, men det kan være nyttig for en utvikler å kunne få et overblikk over ontologien. ”*Dersom programmet kan kombineres med en søkefunksjonalitet som bruker denne informasjonen til å ”koble sammen” sidene i wikien, kan programmet bli et svært kraftig og nyttig verktøy.*”

Annet

Visualiseringer blir ikke oppdatert før man skifter mellom dem. Dette er et problem. Et timeglass eller lignende burde indikere at programmet fortsatt jobber (En del visualiseringer tar litt tid, og da kan det virke som om programmet har krasjet eller at ingenting skjedde, selv om programmet fortsatt jobber). Visualiseringene hadde vært bedre om de kunne følge linker (Gjelder JGraph visualiseringen). Bruk av ”instance of” o.l tekster for pilene i grafen burde vært byttet ut med noe mer forklarende, for eksempel ”Subkategori”.

4.4.3 Evaluering 3

Kandidat C.

Jobber som selger i dataavdelingen i en elektronikkforretning. Har stor interesse og god kompetanse innen elektronikk og data (særlig Mac).

Brukergrensesnittet (Observasjon og intervju)

Oppgave 1. Oppkobling og visualisering

Oppkoblingen gikk raskt. Kandidaten klikker ”Connect” og kobler seg til eksempel-wikien. Han forstår ikke hvordan han skal få til å visualisere. ”*Det står ikke ”Visualiser” noe sted.*” Derfor ber han om hjelp og får vite at han må velge en av sidene som vises i listen. Han dobbelklikker på en av dem slik at den blir visualisert.

Under intervjuet kom det frem at kandidaten hadde litt problemer med å forstå hva visualiseringen betydde. ”*Hva ser jeg på?*” var første kommentaren da den første wikisiden ble visualisert. Under observasjonen fikk han en ny forklaring på hva programmet skulle gjøre og hva som var meningen å visualisere. Han mente ellers at JGraph visualiseringen var den som gav best informasjon, men at informasjonen fra de andre visualiseringene (Tre-baserte) sikkert ville være til god nytte for utviklere og andre med kjennskap til ontologier. Tilkoblingsdelen hadde han ikke noe å utsette på, selv om han trengte litt hjelp til denne delen.

Oppgave 2: Wiki som ikke støttes

Denne delen går greit. Kandidaten kobler seg til WikiPedia og får frem feilmeldingen som forteller at siden ikke kan visualiseres. Helt greit.

Oppgave 3: Åpne fil

Denne delen går uproblematisk. Kandidaten ble på forhånd gjort oppmerksom på at ulike filtre fra listen kunne brukes til ulike tilkoblinger.

Oppgave 4: Endre listen over tilgjengelige sider

Her forstod ikke kandidaten hva han skulle gjøre. Han måtte til slutt bli hintet om at dette skulle gjøres ved å klikke på ”List settings” og at endringene måtte utføres derfra. Under intervjuet bekreftet han at denne delen var noe uklar.

På spørsmål ombrukergrensesnittet som helhet sa han at det var greit og oversiktlig. ”*Det eneste er at ”artikkel” kanskje ikke er det beste ordvalget for hva som skal visualiseres*”. I motsetning til de andre kandidatene mener denne at dobbelklikking for å visualisere en artikkel fungerer greit og er

logisk. Endring av innstillinger er ikke så intuitivt. Her burde det være bedre forklart hva de ulike valgene betydde.

Forståelse av programmets oppgave og nytteverd

Kandidaten forstår programmets oppgave og mener det utfører de funksjonelle oppgavene tilfredsstillende for en prototyp. *Programmet vil nok ha en nytteverdi, men ikke for "folk flest".*

Annet

URL-adressen til siden som blir visualisert vises ikke i nettleseren og det blir derfor vanskelig å skjønne at det er samme siden som vises. Ellers kan programmet selvsagt bli mye bedre, men som prototyp er det greit nok.

4.4.4 Evaluering 4

Kandidat D.

Jobber i et firma som tilbyr IT-support. Tidligere PC-veileder ved Universitetet i Bergen.

Brukergrensesnittet (Observasjon og intervju)

Oppgave 1. Oppkobling og visualisering

Denne delen gikk greit. Tilkobling skjedde uten problemer: Kandidaten klikket på ”Connect-knappen” og ble tilkoblet. Når han skulle visualisere en av wikisidene, forsøkte han først med ”Go-knappen”. Da det ikke gikk forsøkte han først å enkelklikke, dermed dobbelklikking. Da det ikke skjedde noe umiddelbart forsøkte han å dobbelklikke på flere av artiklene helt til den første ble visualisert. Resultatet var at de andre artiklene ble visualisert en etter en (Og erstattet den som først ble visualisert).

Oppgave 2: Wiki som ikke støttes

Selve tilkoblingen til WikiPedia gikk greit. Men kandidaten var litt overivrig i valget av side som skulle bli forsøkt visualisert, noe som førte til at tre like feilmeldinger dukket opp på skjermen samtidig. Feilmeldingene gav likevel god nok forklaring på hvorfor det ikke gikk an å visualisere artiklene.

Oppgave 3: Åpne fil

Her forsøkte kandidaten å bruke ”Fil” og ”Open”. Resultatet ble at forrige wikiside ble visualisert i stedet. For å få til å åpne en fil var det nødvendig å hinte om at han først måtte bytte importfilter. I intervjuet gav han også inntrykk av at det ikke var så lett å forstå hva han skulle gjøre, men at dette bare ville være et problem første gang en brukte programmet.

Oppgave 4: Endre listen over tilgjengelige sider

Denne delen gikk også greit. Kandidaten trengte litt hjelp for å forstå de ulike valgene, men klarte selv å fylle ut de riktige feltene for å få endret listen.

Det generelle uttrykket han satt igjen med var at brukergrensesnittet fungerer greit og han mener det er enkelt og intuitivt i bruk, stort sett. Han hadde som det er blitt beskrevet enkelte problemer med å få til noen av funksjonene men hevdet selv at dette kun skyldtes at han ikke var kjent med programmet fra før. ”*Det burde ikke være noe problem for en som bruker programmet daglig*” var hans kommentar på dette. Ellers mente han at det største problemet til programmet var mangelen på respons. Programmet burde gi klar beskjed når det jobber, for eksempel ved å vise et timeglass eller en boks som viser at programmet gjør noe. Slik det er nå er det ikke mulig å vite om programmet jobber eller har kræsjet.

Forståelse av programmets oppgave og nytteverd

Kandidatene mener han forstår programmets hensikt og at det gjør det som det burde kunne gjøre. Visualiseringen er grei: ”*Den kobler seg til og viser strukturen på dokumentet og den kobler til. Så den gjør vel det den skal.*” Programmet har helt sikkert en nytteverdi. For folk som utvikler og jobber med slike wikier vil det være et nyttig verktøy.

Annet

Visualiseringen blir ikke oppdatert før man skifter mellom dem, og den hopper heller ikke til en av dem hvis den viste nettleseren fra før. Dette skjedde blant annet da kandidaten åpnet en fil for visualisering.

4.5 Oppsummering

Dette kapitlet har beskrevet evalueringen av prototypen, WikiVis. Evalueringen ble gjennomført på fire testpersoner. Disse var håndplukket som representative, reelle brukere av det ferdige systemet.

Målsetningene for evalueringen var å evaluere selve prototypen (funksjonalitet og brukergrensesnitt), få innspill til forbedringer og endringer, og sist men ikke minst: få subjektive meninger som kunne brukes i besvarelsen av prosjektets problemstilling.

Evalueringene bestod av et sett forhåndsdefinerte oppgaver kandidatene ble bedt om å utføre ved hjelp av programmet mens de ble observert. Deretter ble det gjennomført et semi-strukturert intervju. Mens observasjonen viste *hvordan* reelle brukere vil kunne bruke programmet, hva de gjør feil og hva de gjør riktig, gav intervjuet en bedre forståelse av *hvorfor* de gjorde de valgene de gjorde. Intervjuene gav også informasjon som ble brukt i besvarelsen av prosjektets problemstilling og innspill til forbedringer og endringer i programmet.

Resultatene av evalueringen viste at programmet, tross en del begrensninger, utførte de oppgavene et slikt program må utføre. Brukergrensesnittet fungerer stort sett tilfredsstillende. Et par mindre problemer gikk likevel igjen, blant annet måten valg av wikiside som skal visualiseres blir gjort. At dette skulle være et problem, kom som en overraskelse på undertegnede, men viser klart hvorfor evalueringer er nyttige for å avdekke feil og mangler. Manglende tilbakemelding om at programmet er opptatt når det utfører oppgaver (for eksempel tilkobling og klargjøring for visualisering) var også et problem som gikk igjen. Dette var en på forhånd kjent svakhet med systemet. Det ble gjort et forsøk på å løse denne delen under utviklingen, men fordi det ville innebære å skrive om mye av koden (Bruk av Threads) ble det ikke gjennomført. Problemet ble dessuten kjent svært sent i utviklingen. Visualiseringene fungerer greit, selv om det stort sett var den grafiske, JGraph baserte visualiseringen som gav nyttig informasjon.

Det er som nevnt en del begrensninger i programmet og derfor mye rom for forbedringer. Fordi programmet er en prototyp var det heller aldri meningen å utvikle et ”perfekt” program. Den viktigste funksjonen til prototypen var å vise litt av potensialet og nytteverdien for et slikt verktøy og gjennom erfaringene fra utviklingen og evalueringen kunne besvare eller få en antydning til besvarelse av problemstillingen til prosjektet: Vil et slikt verktøy føre til forenklet utvikling og bedre forståelse av wiki-ontologier?

På dette området var resultatene fra evalueringene positive: Alle testpersonene mente at et slikt program vil kunne være et nyttig verktøy for å forenkle forståelse og utvikling av ontologier. Noen mente riktignok at det kun ville være nyttig for utviklere og spesielt interesserte, i hver fall i sin

nåværende form. Det ble blant annet påpekt at en forbedret visualisering som kombineres med selve teksten som vises i wikien vil kunne gjøre dette til et nyttig verktøy også for ”vanlige” wikibrukere. Kandidat D mente at verktøyet kombinert med semantisk søk vil være svært nyttig.

Evalueringen ble utført på fire personer. Selv om intervjuer kan gi langt mer informasjon for hver evaluering enn for eksempel en spørreundersøkelse, var det en relativt liten testgruppe. Det var derfor på forhånd knyttet noe usikkerhet til om dette ville være nok. Testerne ble likevel nøye utvalgt med tanke på at de skulle ha best mulige kvalifikasjoner både til å forstå problemstillingen og hva som var viktige tilbakemeldinger. To av dem er masterstudenter som selv har drevet mye med programmering og disse visste hvilke typer feil de burde se etter og hvordan grensesnitt bør være. Resultatene, særlig for besvarelsen av prosjektets problemstilling, var rimelig entydige: Et slikt verktøy vil føre til enklere utvikling og forbedret forståelse av wiki-ontologier. Alle kandidatene så nytteverdien til programmet og mente at det vil kunne være behov for et slikt program. Det var derimot en del uenighet om hvorvidt den nåværende prototypen ville dekke alle disse behovene.

Det at alle fire tror at et slikt program vil ha en nytteverdi gir også gode indikasjoner for at det vil være behov for denne typen programvare. Den virkelige nytteverdien vil en derimot først kunne se etter å ha utviklet et endelig program uten begrensningene fra denne prototypen.

5. Oppsummering og konklusjon

Målet for denne masteroppgaven var å finne ut om et verktøy for visualisering av wikiontologier ville føre til bedre forståelse og utvikling av slike ontologier. Fordi det ikke allerede fantes et slikt verktøy, ble det som en del av prosjektet utviklet en prototype på et slikt verktøy, WikiVis. Det ble foretatt en evaluering av prototypen for å kartlegge potensielle brukeres forståelse, erfaringer og synspunkter til et slikt verktøy, blant annet hvilken nytteverdi de mente programmet ville kunne ha. Denne evalueringen er gjennomgått i kapittel 4.

I dette kapitlet er det meningen å besvare problemstillingen, basert på resultatene fra evalueringen og erfaringene fra utviklingen.

5.1 Tolkning av resultater

5.1.1 Design og utvikling

Fordi det ikke fantes noe tilsvarende program fra før, var det nødvendig å utvikle en prototyp for å kunne utføre tester og innhente erfaringer om i hvilken grad et slikt program ville kunne gi en forenklet utvikling og forståelse av wiki-ontologier eller ikke. Det første problemet som måtte avklares var derfor om det var mulig å utvikle et verktøy for visualisering av wikiontologier gjennom et begrenset prosjekt som dette. Det ble valgt å benytte en iterativ metode til denne oppgaven, nærmere bestemt evolusjonær utvikling. Metoden ble valgt fordi den var den mest fleksible av de aktuelle metodene. Dette var viktig fordi det på forhånd var vanskelig å planlegge hele utviklingen. Det kunne oppstå problemer og endringer i kravspesifikasjonen underveis og det var begrenset tid til disposisjon. Ved å bruke denne metode startet utviklingen med de mest kritiske og krevende delene av programmet og programmet ble deretter gradvis utvidet så mye som tiden tillot.

Utviklingen av prototypen, WikiVis har vært svært tidkrevende og utfordrende. Selv om den ferdige prototypen er relativt enkel, har denne delen tatt opp mesteparten av tiden for prosjektet. Målet om å utvikle et verktøy for visualisering av wikiontologier er likevel nådd og utviklingen har dermed vist at det er fullt mulig å utvikle et slikt verktøy. Selv om WikiVis er en svært enkel prototyp, løser den de viktigste oppgavene som kreves av et slikt visualiseringsverktøy. Prototypen er utviklet for å kunne utvides og forbedres i ettertid, men arkitekturen kan også brukes som en skisse til et forbedret og fullt utviklet verktøy for visualisering av wikiontologier. Det er et mål at erfaringene skal være et viktig bidrag til fremtidig utvikling også for tilvarende prosjekter.

5.1.2 Evaluering av funksjonalitet

Den første delen av evalueringen som ble utført gikk ut på å teste brukergrensesnittet og funksjonaliteten til programmet. Evalueringen avdekket en del problemer med brukergrensesnittet.

Dette gikk blant annet ut på plassering av knapper og manglende indikasjon om at programmet var opptatt. Når det gjaldt selve funksjonaliteten var det enighet om at programmet utførte de grunnleggende og nødvendige funksjonene som kreves av et slikt program, om enn i en noe begrenset form. Konklusjonen for denne delen er derfor at programmet (WikiVis 1.1b) er en tilfredsstillende prototyp for å kunne evaluere nytteverdien av et slikt verktøy.

5.1.3 Evaluering av nytteverdi, besvarelse av problemstilling

Selve problemstillingen til prosjektet har vært å besvare om et slikt verktøy ville føre til bedre forståelse og utvikling av slike ontologier. For å besvare dette spørsmålet ble altså prototypen WikiVis utviklet og evaluert. Den første delen av evalueringen viste at programmet funksjonsmessig utførte de oppgavene som kreves av et slikt verktøy. Når det gjelder nytteverdien til programmet viser resultatene at programmet selv i sin nåværende, enkle form kan være et nyttig verktøy.

5.1.4 Konklusjon

Resultatene fra evalueringen indikerer at et verktøy for visualisering av wikiontologier *vil* føre til bedre forståelse og utvikling av slike ontologier. Selv en begrenset prototyp som WikiVis 1.1b vil ha en viss nytteverdi for dette formålet. Programmet kan i sin nåværende form gi støtte i form av å tilby en forenklet oversikt eller et kart over ontologien. Det er likevel først og fremst gjennom videreutvikling av programmet, eventuelt gjennom nyutvikling av et tilsvarende, fullstendig implementert program, at programmet virkelig vil kunne være et nyttig verktøy. Behovet for et slikt program er allerede til stede. Etter hvert som ontologier får en stadig viktigere funksjon, ikke minst gjennom fremtidens semantiske web, vil bruken av wiki-ontologier også vokse. Dermed vil også behovet for denne typen visualiseringsverktøy øke.

5.2 Tanker, refleksjoner og videre arbeid

5.2.1 Antall testpersoner

Evalueringen ble gjennomført med 4 testpersoner. Dette er et relativt lite utvalg og det var på forhånd noe usikkerhet om det ville være tilstrekkelig. Både metodene som ble valgt og personene som ble bedt om å delta i testen ble valgt med tanke på å få så mye som mulig informasjon for hver evaluering. Det var begrenset hvor mye tid og ressurser som kunne brukes på denne delen, derfor ble for eksempel et spørreskjema (Som ville krevd lang flere personer og gitt mindre målbare data per evaluering) forkastet. En oppgave som denne vil ikke kunne komme med noen endelig bevis, men heller en indikasjon for hva som er svaret på problemstillingen. På spørsmål som gjaldt nytteverdien til programmet (dersom det ble videreutviklet) og hvorvidt det ville føre til forenklet utvikling og forbedret forståelse av ontologier, var svarene ganske enstemmig positive. Dette gir en klar indikasjon på at det er et behov for et slikt program og at det vil ha en nytteverdi, i hvert fall for

noen. Konklusjonen er derfor at fire personer var tilstrekkelig for å kunne besvare problemstillingen. Dersom prototypen blir videreutviklet, vil det være behov for å utføre mer omfattende evaluering, særlig av brukergrensesnittet og funksjonaliteten. Testene bør testes av både ontologi-eksperter og ordinære brukere da målet er at begge grupper skal ha nytteverdi av programmet. Testene bør i større grad tilpasses behovene for de ulike brukergruppene.

5.2.2 Programmets brukergruppe

I sin nåværende form gir programmet først og fremst informasjon om strukturen til ontologien. Denne informasjonen kan være nyttig, og gi noe forenklet utvikling og forståelse, spesielt for utviklere, men ”vanlige” brukere vil ikke nødvendigvis ha nytte av denne funksjonaliteten. Dette er likevel et problem som skyldes begrensningene til de implementerte visualiseringene. Det har under hele utviklingen vært opplagt at programmet, som er en prototyp, vil være noe begrenset i funksjonalitet, spesielt når det gjelder visualiseringene. Dersom programmet blir videreutviklet vil det kunne bli et langt bedre og nyttigere verktøy, ikke bare for spesielt interesserte, men også for utviklere generelt og en del ”vanlige” brukere av wikier. Som verktøy vil det kunne føre til forenklet utvikling og forbedret forståelse av wiki-ontologier. Dette var alle kandidatene enige om. Dersom programmet blir videreutviklet, bør det være et mål at det skal ha nytteverdi for begge disse brukergruppene. Det kan være behov for å tilby to grensesnitt, et for avanserte brukere (utviklere) som har mulighet til å gi detaljinformasjon, gjerne med mulighet til å endre innholdet som visualiseres, mens det for ordinære brukere vil være tilstrekkelig å gi en så enkel visualisering som mulig.

5.2.3 Forbedringer og videre arbeid

Som forventet i et slikt prosjekt er det en del løse tråder som gjenstår. Tabell 7 viser noen av forbedringene som fortsatt kan utføres på programmet og som vil kunne bidra til en økning av programmets nytteverdi. En del av disse var kjent på forhånd, mens andre ble avdekket under evalueringen. Alle disse forbedringene kan implementeres, men ikke som en del av dette prosjektet på grunn av prosjektets tidsramme og begrensede ressurser. Prosjektets målsetting har heller ikke vært å utvikle et fullstendig og "perfekt" program, men en prototyp som kan brukes til å besvare problemstillingen. Denne målsetningen er oppnådd gjennom WikiVis 1.1b.

Tabell 7. Forslag til forbedringer og videre arbeid

Wikistøtte	**Flere og bedre importfiltre. Herunder mulighet for pålogging, innhold basert på språk (for wikier som støtter dette).**
Visualiseringer	Bedre grafiske løsninger, mulighet for interaktivitet, kombinere med wikitekst. Mulighet for "zooming" av detaljer.
Redigering	Mulighet for å endre innholdet i ontologien/wikien gjennom visualiseringen
Tilbakemeldinger	Bedre forklaring av spesialinnstillinger. Timeglass eller lignende for å indikere når programmet er opptatt m.m.
Brukergrensesnitt	Mer logisk plassering av knapper og lignende.
Snarveier	Legge til snarveier (hurtigtaster)
Tilpasning av innhold	Det er allerede mulig å tilpasse innholdet ved hjelp av en "ekskluderingsfunksjon" for å filtrere vekk uønskede elementer. Men denne delen kan bli enklere i bruk.
Program	Programmet kan bli raskere, få bedre feilhåndtering m.m.

Referanseliste

Abratt, M., Bekker. (2003). "Usability Testing: Recipe for Success."

Aduna. (2008). "User Guide for Sesame: What is Sesame?" from http://www.openrdf.org/doc/sesame/users/ch01.html.

Akrivi, K., H. Constantin, et al. (2007). "Ontology visualization methods\—a survey." ACM Comput. Surv. **39**(4): 10.

Babu, J., van Wijk (2008). Supporting the analytical reasoning process in information visualization. Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems. Florence, Italy, ACM.

Barrio. (1999). "The Use of Semistructured Interviews and Qualitative Methods for the Study of Peer bullying." from http://www.gold.ac.uk/tmr/reports/aim2_madrid1.html.

Berners-Lee, H., Lassila. (2001). "The Semantic Web." from http://www.sciam.com/article.cfm?id=the-semantic-web&page=1.

Caplex (2008). Metode. Caplex nettleksikon, J.W. Cappelens Forlag AS.

Cockburn (2002) Agile Software Development Joins the "Would-Be" Crowd. **Volume**, 6-12 DOI:

D'arcy, D., Case (1990). The community's toolbox: Semi-structured interviews, FAO Corporate Document Repository.

Denny, M. (2002). "Ontology Building: A Survey of Editing Tools." from http://www.xml.com/pub/a/2002/11/06/ontologies.html.

Dix, F., Abowd, Beale (1993). Human-Computer Interaction, Pearson Prentice Hall**:** 318 - 364.

Gruber, T. (1992). "What is an Ontology." from http://www-ksl.stanford.edu/kst/what-is-an-ontology.html.

Grune, J. (1990). Parsing Techniques - A Practigal Guide, Ellis Horwood, Chichester, England.

Heiko, H., K. Markus, et al. (2006). Semantic Wikipedia. Proceedings of the 2006 international symposium on Wikis. Odense, Denmark, ACM.

Herman, I. (2007). "Web Ontology Language (OWL)." from http://www.w3.org/2004/OWL/.

Herman, S., Brickley. (2008). "Resource Discription Framework (RDF) / W3C Semantic Web Activity."

Horrocks, v. H., Patel-Schneider. (2001). "DAML Language: DAML+OIL."

IkeWiki. (2008). "IkeWiki homepage." from http://ikewiki.salzburgresearch.at/.

Jerilyn, P. and C. Matt (1999). Usability testing: a quick, cheap, and effective method. Proceedings of the 27th annual ACM SIGUCCS conference on User services: Mile high expectations. Denver, Colorado, United States, ACM.

Larman, B. (2003) Iterative and Incremental Development: A Brief History. Computer **Volume**, 47-56 DOI:

> Although many view iterative and incremental development as a modern practice, its application dates as far back as the mid-1950s. Prominent software-engineering thought leaders from each succeeding decade supported IID practices, and many large projects used them successfully. These practices may have differed in their details, but all had a common theme-to avoid a single-pass sequential, document-driven, gated-step approach.

Marc, P. (2006). "Review of "Wiki: Web Collaboration by Anja Ebersbach, Markus Glaser and Richard Heigl," Springer, 2005, $64.95, ISBN: 3540259953." Queue **4**(2): 59-59.

Martin, R. C. (2001) Agile Processes. **Volume**, DOI:

McBride, B. (2007). "An Introduction to RDF and the Jena RDF API." Jena Tutorial.

Michel, B., G. Fabien, et al. (2008). "SweetWiki: A semantic wiki." Web Semant. **6**(1): 84-97.

Nielsen, J. (2005). "Heuristic Evaluation." from http://www.useit.com/papers/heuristic/.

PARK. (2008). "Glossary of Sensemaking Terms." from Human-Document Interaction Area PARC.

Schaffert (2006). IkeWiki: A Semantic Wiki for Collaborative Knowledge Management. Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, IEEE Computer Society.

SMW. (2008). "Semantic MediaWiki." from http://semantic-mediawiki.org/wiki/Semantic_MediaWiki.

Soanes, C. (2005). Blikis, phlogs, and moblogging. Oxford English Dictionary, Oxford Dictionary.

SweetWiki. (2007). "SweetWiki - Introduction." from http://argentera.inria.fr/wiki/data/Main/MainHome.jsp.

Tasha, H. and G. N. David (2007). Usability inspection methods after 15 years of research and practice. Proceedings of the 25th annual ACM international conference on Design of communication. El Paso, Texas, USA, ACM.

Tauberer, J. (2006). "What is RDF." from http://www.xml.com/pub/a/2001/01/24/rdf.html.

Tett (2003). Scientific methods. School of Life Sciences, Napier University, Edinburgh.

Tom, G. (1985). "Evolutionary Delivery versus the "waterfall model"." SIGSOFT Softw. Eng. Notes **10**(3): 49-61.

US_Goverment. "Learn About Usability Testing." from http://www.usability.gov/refine/learnusa.html#whatis.

Verzulli (2001) Using the Jena API to Process RDF. **Volume**, DOI:

W3C. (1996). "Extensible Markup Language (XML)." from http://www.w3.org/XML/.

Wikipedia. (2008). "Parsing." from http://en.wikipedia.org/wiki/Parsing.

Wikipedia. (2008). "Semantic Web." from http://en.wikipedia.org/wiki/Semantic_Web.

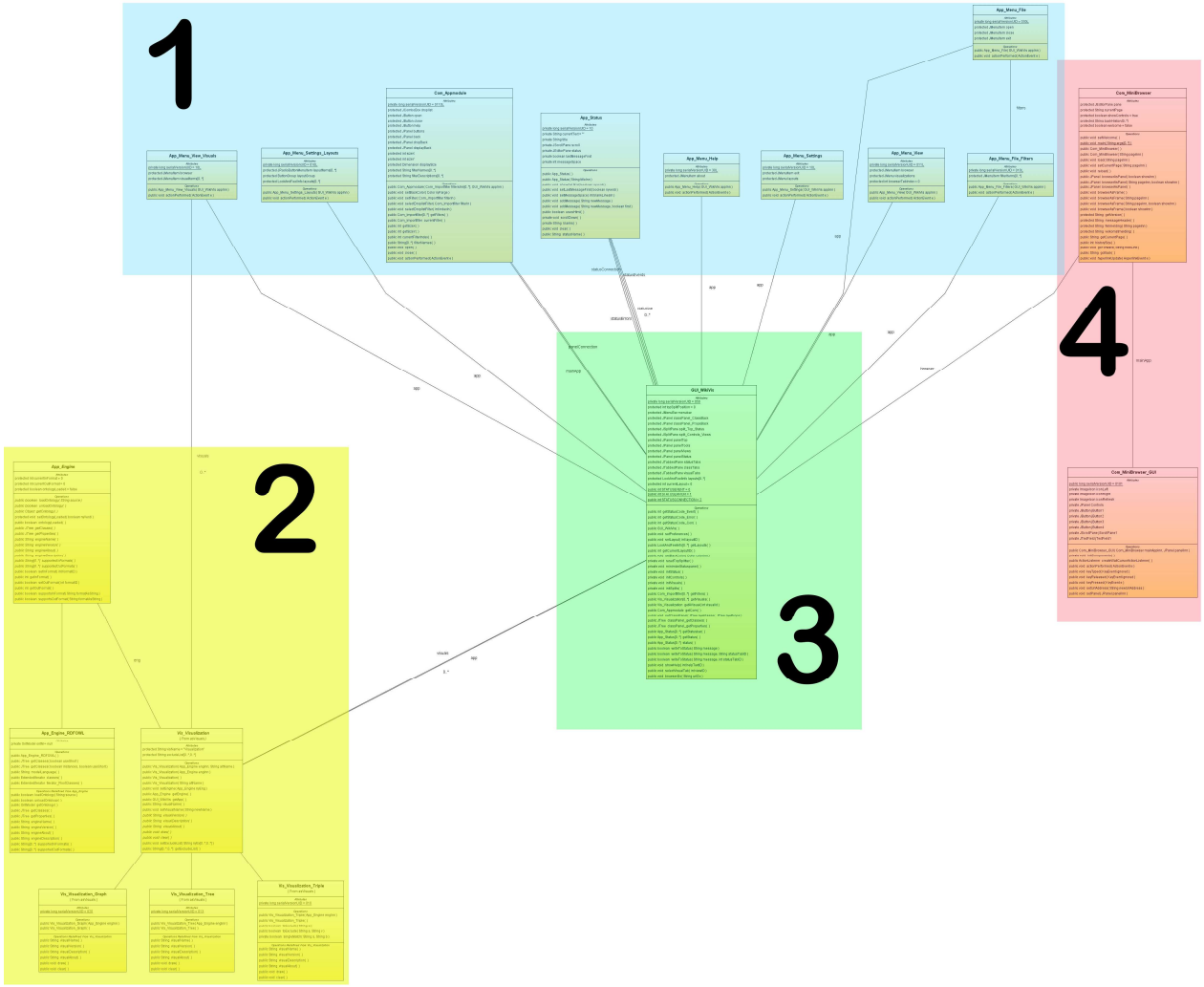
Wikipedia. (2008). "Usability testing." from http://en.wikipedia.org/wiki/Usability_testing.

Appendiks

Appendiks A: Klassediagram, WikiVis

Klassediagram: Oversikt

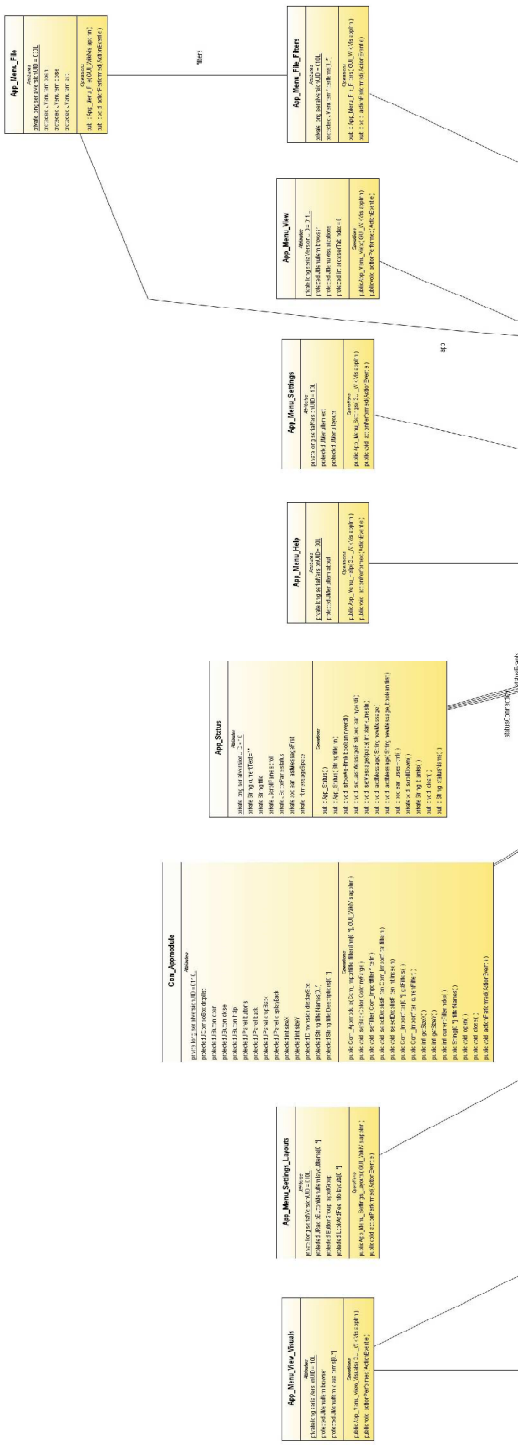
Figur 29 viser en oversikt over de viktigste klassene til WikiVis og relasjoner mellom disse. Generelle klasser (For eksempel Utility-klasser) er ikke tatt med. Detaljer for de nummererte feltene følger.



Figur 29. Klassediagram for WikiVis: Oversikt over de viktigste klasser og relasjoner

Klassediagram: Detaljer Meny (1)

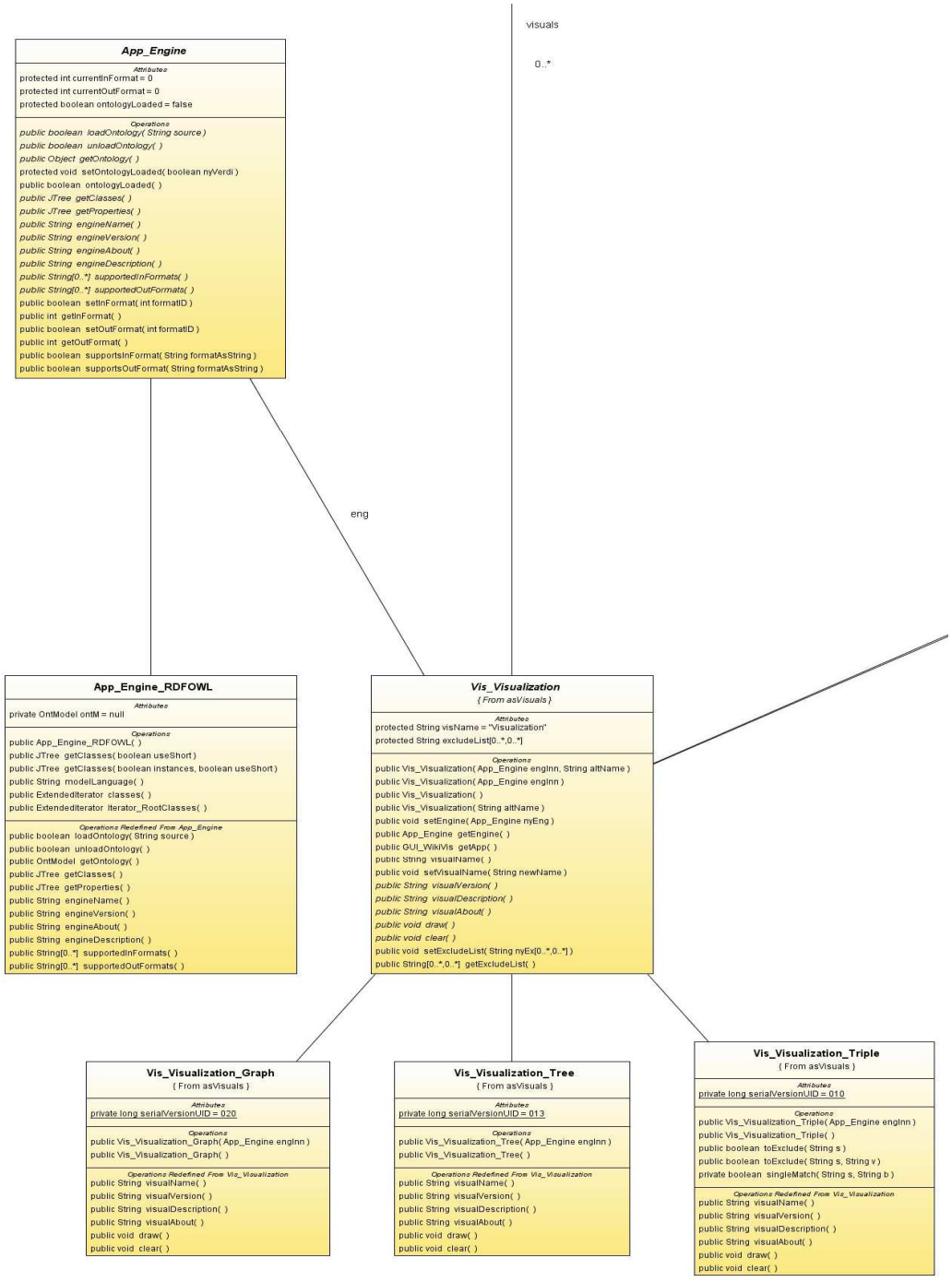
Figur 30



Figur 30. Detaljer fra klassediagram: Meny (1)

Klassediagram: Detaljer Tolk og visualiseringer (2)

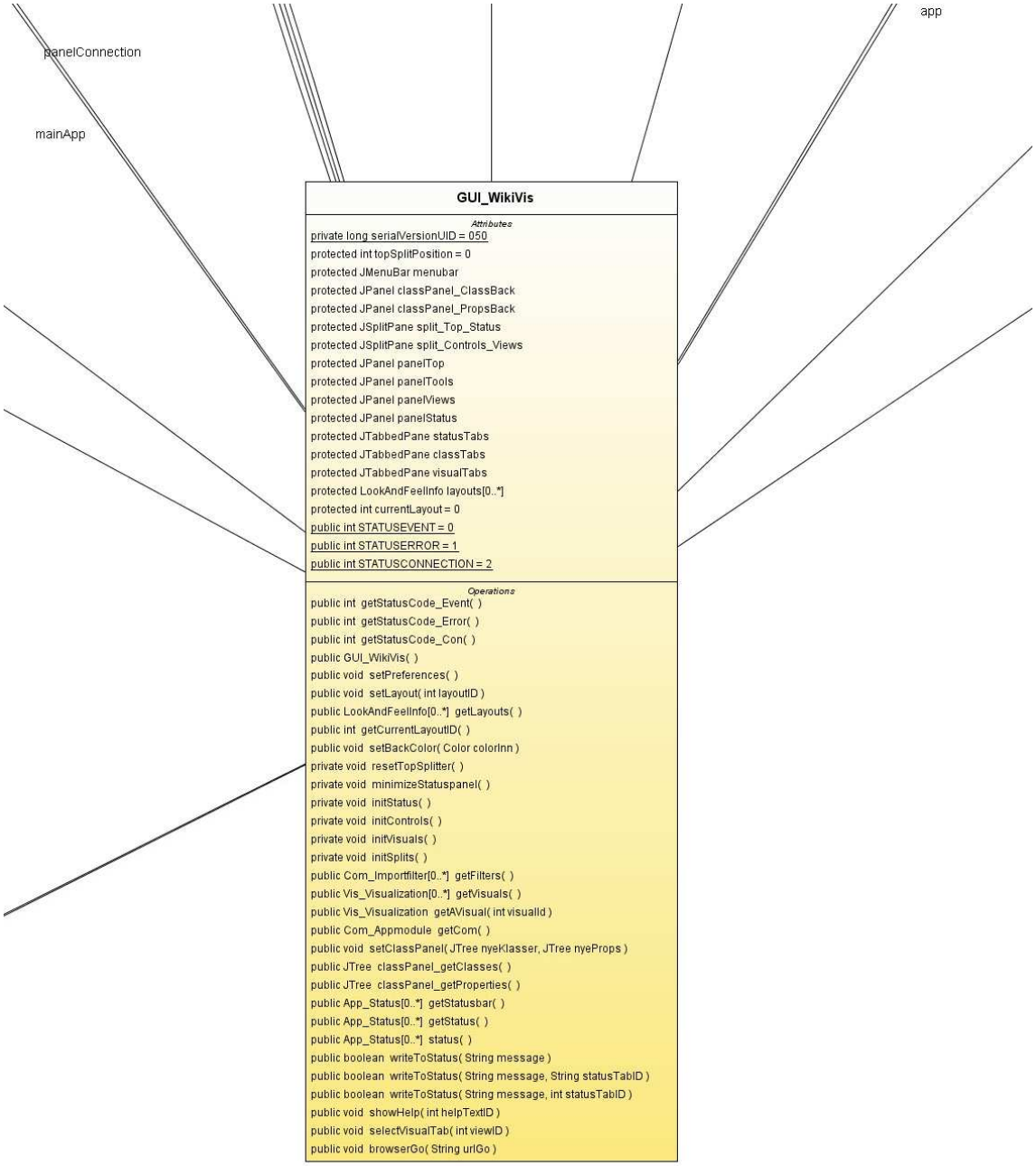
(Figur 31)



Figur 31. Detaljer fra klassediagram: Tolk (Engine) og visualiseringer (2)

Klassediagram: Detaljer WikVis hovedmodul (3)

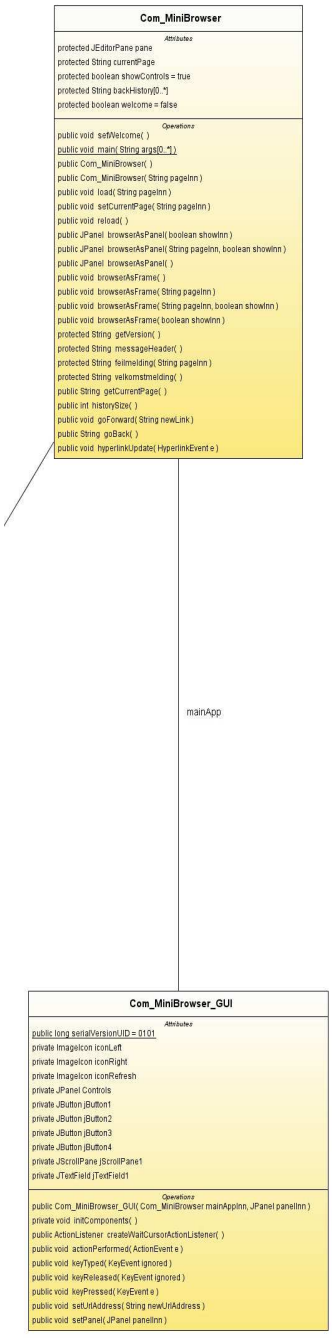
(Figur 32)



Figur 32. Detaljer fra klassediagram: WikiVis hovedmodul (3)

Klassediagram: Browser (4)

(Figur 33)



Figur 33. Detaljer fra klassediagram: Browser (4)

Appendiks B: Oversikt over kildekodefiler

Rot-katalog og oppstartsfiler

(Tabell 8)

Tabell 8. Rot-katalog, main() metode for oppstart av program

Pakke / katalog	Klassse
\	Rot-klassen
	WikiVis

Dialogbokser

(Tabell 9)

Tabell 9. asDialogs - Dialogbokser

Pakke / katalog	Klassse	Beskrivelse
asDialogs	Dialogbokser som brukes av programmet (Innstillinger, hjelp, osv)	
	Dialog_About	Viser informasjon om programmet
	Dialog_About_Logo	Hjelpeklasse til Dialog_About (Viser logoen til WikiVis)
	Dialog_Alert	Generell Feilmeldingsboks
	Dialog_Alert_Logo	Hjelpeklasse til Dialog_Alert (Viser logoen til WikiVis)
	Dialog_ChooseVisuals	Dialogboks for å velge visualisering(er) som skal brukes
	Dialog_FilterSMW_Exclude	Innstillinger for Exclude-list funksjon (Innhold som skal filtreres vekk)
	Dialog_FilterSMW_Settings	Innstillinger for Semantic MediaWiki-filteret
	Dialog_FilterSMW_Settings_List	Innstillinger for å endre listen. Brukes av Semantic MediaWiki-filteret
	Dialog_Help	Hjelp-dialog. Generell klasse hvor teksten som vises kan endres
	Dialog_Help_Logo	Hjelpeklasse til Dialog_Help. (Viser logoen til WikiVis)
	Dialog_PleaseWait	"Please-wait"-dialog. Brukes ikke i det endelige programmet

Exceptions (Feilhåndtering)

(Tabell 10)

Tabell 10. asExceptions: Unntak (Java Exceptions).

Pakke / katalog	Klassse	Beskrivelse
asException	Egne unntak for feilhåndtering (Java Exceptions)	
	Exception_Canceled	Avbrutt-unntak
	Exception_NotReadable	Ikke-lesbar-unntak

Importfiltre

(Tabell 11)

Tabell 11. asImportFilters: Importfiltre

Pakke / katalog	Klassse	Beskrivelse
asImportFilters	Importfilter	
	Com_Importfilter	Abstrakt klasse (mal) for Importfiltre
	Com_Importfilter_Http_Filter	Jena-basert filter for RDF OWL filer fra URL (Ikke brukt)
	Com_Importfilter_MediaWikiSpecialPages	Filter for SMW. Nedlastning av ontologi vha RDF-eksport-funksjon
	Com_Importfilter_OWLFile	Filter for lokale RDF og OWL-filer

Hovedfiler for program og GUI

(Tabell 12)

Tabell 12. asProgram: Programrelaterte klasser (GUI)

Pakke / katalog	Klassse	Beskrivelse
asProgram	**Hovedkomponenter i programmet**	
	App_Engine	Abstrakt klasse (mal) for tolkmoduler (Engine/motor)
	App_Engine_RDFOWL	Tolkmodul som støtter RDF og OWL (vha Jena)
	App_Menu	Meny, hovedklasse
	App_Menu_File	Meny: Fil
	App_Menu_File_Filters	Meny: Fil, Filter (Valg av filter som skal brukes)
	App_Menu_Help	Meny: Hjelp
	App_Menu_Settings	Meny: Innstillinger
	App_Menu_Settings_Layouts	Meny: Innstillinger, Layout (Valg av layout/utseende av programmet)
	App_Menu_View	Meny: View
	App_Menu_View_Visuals	Meny: View, Visuals (Valg av visualiseringer)
	App_Status	Status-bar
	Com_Appmodule	Den delen av programmets brukergrensesnitt som håndterer kommunikasjon.
	Com_MiniBrowser	Programmets nettleser (Browser), hovedklasse og funksjonalitet
	Com_MiniBrowser_GUI	GUI for nettleseren
	GUI_WikiVis	GUI for WikiVis (Den egentlige hovedklassen til programmet)

Innstillinger

(Tabell 13)

Tabell 13. asSettings: Innstillinger

Pakke / katalog	Klassse	Beskrivelse
asSettings	Innstillinger	
	Settings_FilterSMW_Connection	Tilkoblingsinnstillinger til bruk med SMW-filteret
	Settings_FilterSMW_General	Generelle innstillinger for SMW-filteret
	Settings_FilterSMW_List	Innstillinger for hvordan liste over artikler skal lastes ned (ved bruk av SMW-filter).
	Settings_FilterSMW_WikiInfo	Wiki-innstillinger (Server-adresse, hovedside, osv). En del av denne informasjonen må først "oppdages" av programmet. Ved å lagre dem slipper en å gjøre dette neste gang verdiene skal hentes frem.
	Settings_StringArray	Generell klasse for å lagre innstillinger i tekst-format

Datastrukturer

(Tabell 14)

Tabell 14. asStructures: Datastrukturer utviklet for programmet

Pakke / katalog	Klassse	Beskrivelse
asStructures	Datastrukturer brukt av programmet	
	ActionController	Håndterer doble actionlisteners (Bl.a. til overstyring av dobbelklikk)
	ActionController_Cursor	Implementasjon av ActionController som overstyrer musepeker (vanlig/vente-modus)
	CellNeighbors	Struktur for å få enklere tilgang til nabonoder i en graf
	EdgeNeighbors	Struktur for å få enklere tilgang til nabokanter i en graf
	JTObject	Struktur som forenkler bruk av Javas JTree trestruktur
	JTSearcher	En utvidelse av JTree som inneholder søkefunksjon for treet
	LinkWithTitleObject	Struktur bestående av en link og en tittel til denne. Brukes for å bytte mellom visning av tittel (dersom linken har dette) og link-adresse
	OntTreeObject	En trestruktur bestående av elementer fra en Jena-ontologi
	RDF_Class	Struktur som forenkler håndtering av RDF-klasser (Ikke brukt)
	RDF_Instance	Struktur som forenkler håndtering av RDF-instanser (Ikke brukt)
	RDF_Statement	Struktur som forenkler håndtering av RDF-statements (Ikke brukt)
	ResourceFilter_isAnonymous	Jena "filter". Brukes til å fjerne uønskede deler av ontologien
	ResourceFilter_isNothing	Jena "filter". Brukes til å fjerne uønskede deler av ontologien
	String_Int_Object	Struktur bestående av to verdier: en tekstverdi og en tilhørende heltallsverdi
	Table_AutoSize_String	Utvidelse av Java-tabell som endrer størrelse automatisk ved behov
	Table_AutoSize_String_Model	"Modell" som håndterer strukturen til en slik tabell

Utilities (Nyttige funksjoner)

(Tabell 15)

Tabell 15. asUtils: "Utilities/verktøy". Samling av nyttige funksjoner

Pakke / katalog	Klassse	Beskrivelse
asUtils	Nyttefunksjoner og "verktøy"	
	Utils_Action	Funksjoner for å håndtere hendelser, bl.a. overstyring av dobbelklikking i lister.
	Utils_Graph	Graf og tre-relaterte funksjoner
	Utils_Http	HTTP og internett-relaterte funksjoner
	Utils_IO	I/O og filfunksjoner
	Utils_Jena	JENA-relaterte funksjoner
	Utils_JGraph	Funksjoner relatert til JGraph
	Utils_Other	Andre nyttefunksjoenr
	Utils_Settings	Nyttefunksjoner som brukes av til å lagre/hente innstillinger
	Utils_SMW	Funksjoner relatert til Semantic MediaWiki
	Utils_System	System-relaterte funksjoner (OS-versjon, Java-informasjon m.m)

Visualiseringer

(Tabell 16)

Tabell 16. asVisuals: Visualiseringer

Pakke / katalog	Klassse	Beskrivelse
asVisuals	Visualiseringer	
	Vis_Visualization	Abstrakt klasse (mal) for visualiseringer
	Vis_Visualization_Graph	JGraph visualiseringen
	Vis_Visualization_Text	Text visualisering (Ikke brukt i det endelige programmet)
	Vis_Visualization_Text_Tree	Text-Tree visualisering (Ikke brukt i det endelige programmet)
	Vis_Visualization_Tree	JTree visualiseringen
	Vis_Visualization_Triple	Triple-visualiseringen

Appendiks C: WikiVis Manual

WikiVis Manual

Master thesis program WikiVis:
A tool for visualization of wiki ontologies.

Version 1.1.
Last modified: 24.04.2008

This program is still under development.
To get the latest release, check out the project's homepage:
http://www.student.uib.no/~st01360/Master

Quick install guide:

Having downloaded the zip-file containing the program, unzip the content to a new folder. This will be the programs root folder. Before the program can run, the CLASSPATH variable must be updated to include all the required libraries needed by the program. These are located in the /lib subfolder. If you are running Windows, just run the "run.bat" file and the class variables will be set automatically and the program will be started. After setting the CLASSPATH the program may also be started by typing "java WikiVis" from the folder where the code has been installed.

Contents of the archive containing the program:

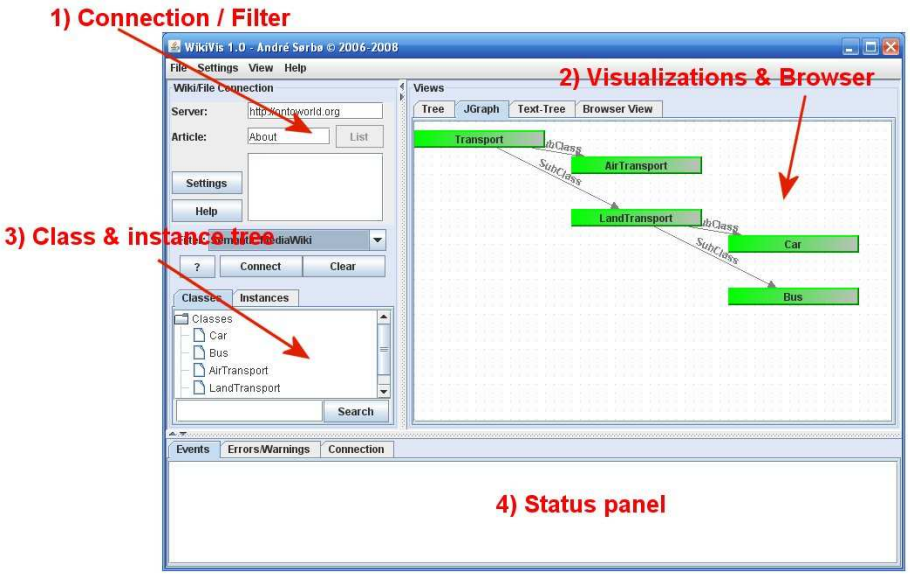
- /lib/ All required, external libraries
- /examples/ Example ontologies. These files may be opened and visualized using the "RDF/OWL File" filter.

The rest of the subdirectories contain the program itself

Dependencies:
Use the links to download or get further instructions about the libraries the program depends on:

- Jena Semantic Web Framework for Java:
  http://jena.sourceforge.net/
- JGraph – Java Graph Visualization and Layout:
  http://www.jgraph.com/

The program's layout:

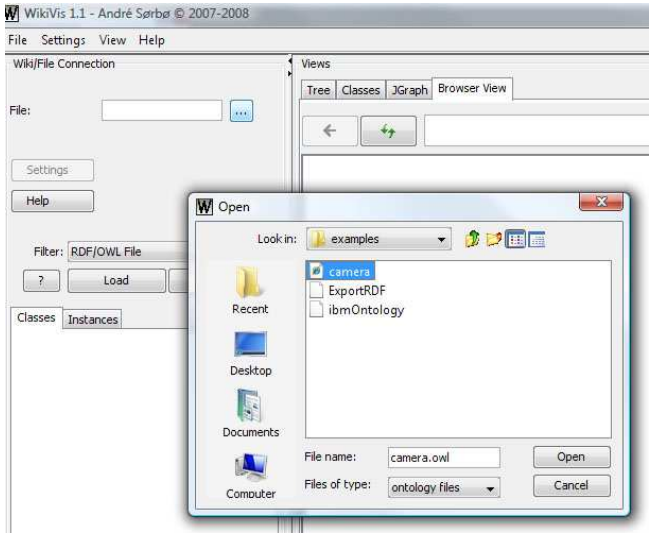


1) Connection / Filter:
   Used to load an ontology. Select one of the available "importfilters" from the droplist. Each filter provides the necessary settings to start a connection and visualize a specific type of ontologies or wikis.
2) Visualizations & browser:
   This is where the ontology visualization is displayed. After connecting to a wiki (or file), a list of available visualizations is displayed. Chose the visualization(s) you want to use and press OK. The ontology is visualized with the selected visualization(s).
   Browser. This is not a visualization, but a simple internet browser. It may be used to look at the wiki the usual way (Using HTML). However the browser is quite minimalistic and does only support HTML 1.0, so only basic browser support is available.
3) Class & Instance tree:
   Once the ontology is loaded, a list of all the classes will be displayed in this field. To get a list of a class' instances, select the class and press the "instances" tab. The search filed may be used to locate a class if the list.
4) Status panel:
   This panel will be used to show messages (Warnings, progression etc). Different tabs are used to show different kinds of information. The panel can be minimized if this information is of no interest.

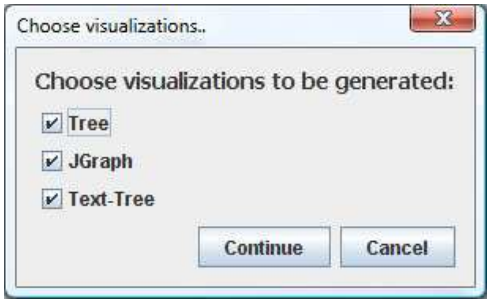
Example: Visualizing an ontology:

Step 1: Connecting and loading of the ontology.
The connection is done using the connection panel. Different wikis require different methods for making connections and different information may be required by the user to do this. Each of the filters available from the list will display a set of input fields required to start the visualization.

For the "Semantic MediaWiki" filter the URL of the server and the article that contains the ontology code to be visualized are required. For the "RDF/OWL File" filter a filename is required. Write a filename in the text field or press the "…" button to choose the file from a "Open file dialog". Alternatively, choose "File, open" from the menubar.

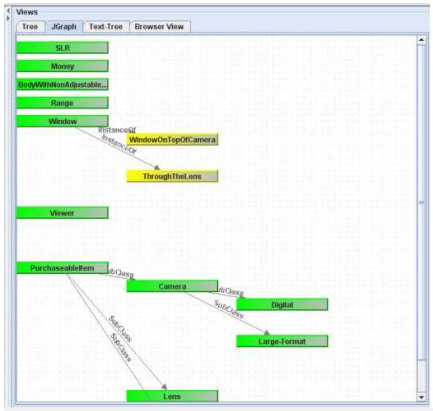


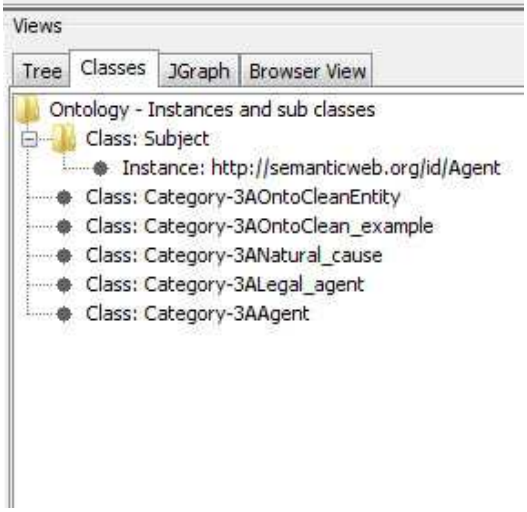
Before the visualizing happens, a dialog will appear asking for which visualizations to use. Only the visualizations selected (all by default) will be used. Select the visualizations to use (At least one):

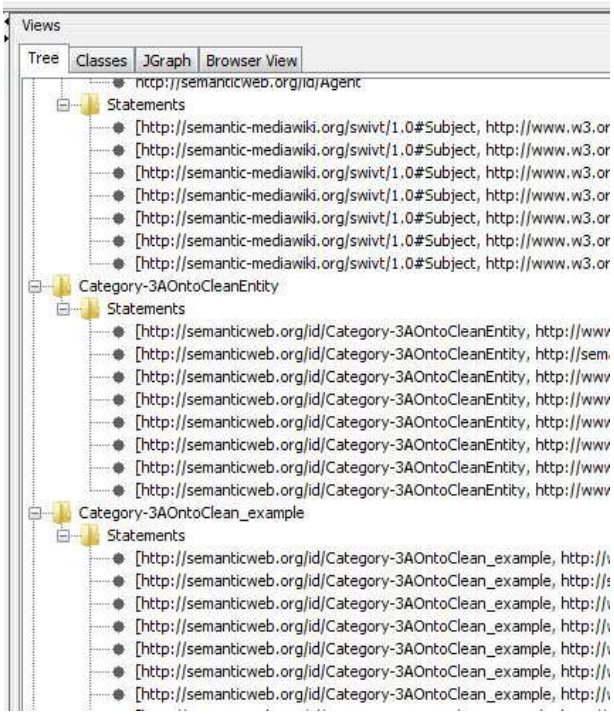


The program will then visualize the given ontology using the visualizations chosen. Use the tabs to switch between the different visualizations (Only the ones chosen will show a visualization). When using the Semantic MediaWiki filter, the browser will also try to load the original wikipage. In this version of the program, v1.1, three visualizations are available:

- Tree: Displays a complete tree structure of the ontology page loaded (including items in the exclude list (The meaning of this will be explained later)
- Classes: Displays a more compact tree structure, and excludes items in the exclude list.
- JGraph: A graphical representation of the ontology, based on the JGraph API.
- Browser view: This is a simple browser, and will be used to show the original wiki page when used with the "Semantic MediaWiki Filter".



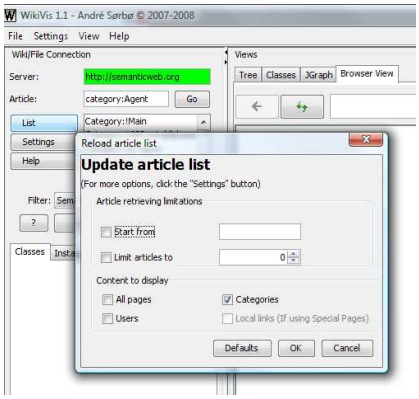




Main filter: Semantic MediaWiki

The "Semantic MediaWiki filter", which is the main filter of the program, works the same way as the "File filter", but has some more options to choose. Instead of a filename, it requires a Wiki server address. This address must be a valid MediaWiki URL. (Missing http:// and redirections will be automatically corrected). When the background of the server has the color red (As in the figure above), this means that a connection has not been established. After entering the server URL press the "Connect" button. If the connection succeeds, the background of the server field will change color to green and a list of available articles will be displayed. Choose the article to visualize, either by double clicking on an article in the list, or by writing a (valid) article name in the "Article" field and pressing go.
By default the list of articles displays only "Category articles", as the ontology is most likely to be distributed on the wiki using articles. Clicking on the "List" button (After a connection has been made), will display a dialog with options for how to create the article list and what to include:

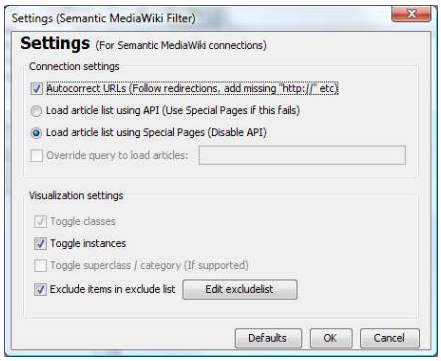
After the file or server/article has been chosen, the ontology may be loaded by pressing the Connect/Open button. The program will now load the ontology. If the settings given were wrong, a internet connection is missing or another error occurred, a warning dialog will be displayed, telling that the loading failed and why. If not a list of the visualizations to use will be displayed (see figure). Content to be displayed can be "Categories" (Default), "Users" or "All pages".



To visualize the ontology content, the wiki must be running Semantic MediaWiki. "Semantic MediaWiki" is an extension or plug in for the MediaWiki wikisoftware. The most important feature of Semantic MediaWiki (at least for this program) is the possibility to encode all pages in the ontologyformat RDF and to export all pages to this format. This format will be used by the wiki to encode all ontology related information within the page. The OWL format, which is a more powerful ontology format is also supported (It will be wrapped in the RDF-code and automatically recognized by the program).

If the wiki does not support RDF export, the program cannot get any useful information to visualize, and will display a error message telling that RDF Export is not supported on the wiki.

Another problem for general wiki ontologies is that the pages may contain other, irrelevant ontology data (Information about the page, the wiki, the formats being used, author etc). This information can be different depending on each wiki. The program has a special feature to handle filtering of this type of information: A "exclude list". This is a list of OWL/RDF tags that will not be displayed in the visualizations (Except the "Tree visualization"). The default list excludes the irrelevant data used in http://semanticweb.org and should be quite general. The list can be edited by clicking the "Settings" button and then "Edit excludelist". It can be enabled/disabled without deleting the list by checking or unchecking the "Exclude items in exclude list" check box.

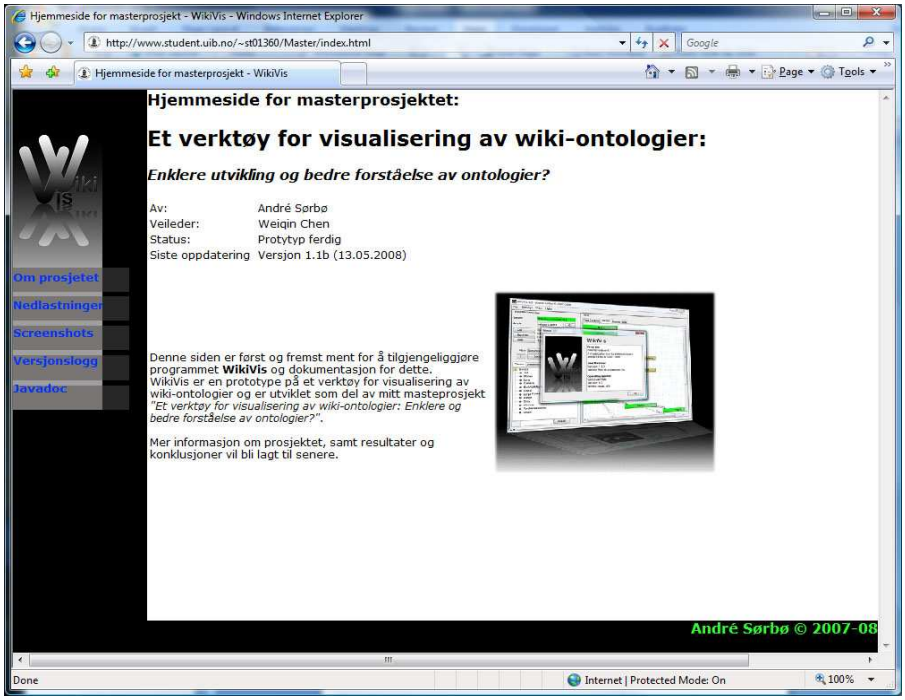


Program history:

2008-04-24	**New functions/Improvements:**
Version 1.1	• "Smart" URL handling (Auto correction of URLs, redirection support etc). RDF-support and scripts used on the wiki (Api.php, Special pages, etc) are automatically detected. If the Api.php scripts is not detected or not present, "Special pages" script is automatically detected instead. • Exclude list: Makes it possible to filter the information to be visualized. • Error handling. The program handles errors and gives warnings / error messages instead of crashing. The status bar is also used to display event and error messages during connection. • Layout: Some minor improvements. Changed default layout from "Java/Metal" to "Windows" • A list of articles is automatically downloaded from the wiki. The user can either doubleclick on an article from this list or write the article in the article field and press go to visualize it. • Settings can now be changed by the user, using different settings dialogs. • The "Text-tree" visualization has been removed, and a new Tree based visualization, "Classes" has been added. • The "JGraph" visualization now shows only the namespace of the URI for each item, making it easier to read and understand its content. Some minor bugs have also been corrected. • The "Browser" has been updated to support redirections, URL correction. Some minor other improvements and bug fixes has been done as well.
2008-03-11	**New functions**
Version 1.1	• Visualizing of RDF/OWL, either from file or from Semantic MediaWiki . • 3 Visualizations: JTree, TextTree and JGraph. • A simple HTML 1.0 browser (AS MiniBrowser 1.0). • Design. GUI, modularity: Visualizations, Importfilters, Engines

Appendiks D: Prototypen på nett

Kildekode, kjørbare filer (inkludert nødvendige tilleggsbiblioteker) og Javadoc-dokumentasjon kan hentes ned fra prosjektets hjemmeside (Figur 34). Her finnes også versjonshistorikk og bilder (screenshots) av programmet.



Figur 34. Prosjektets hjemmeside: http://www.student.uib.no/~st01360/Master/

Siste versjon ved innlevering: WikiVis 1.1b (13.05.2008)

Prosjektets hjemmeside: http://www.student.uib.no/~st01360/Master/

Dokumentasjon (Javadoc): http://www.student.uib.no/~st01360/Master/WikiVis_Doc/