NTNU
Det skapende universitet

En editor for elektroniske læringsobjekter

Implementering av Design Patterns-prinsippene

Lars Weydahl

Master i datateknikk
Oppgaven levert: Juli 2006
Hovedveileder: Arvid Staupe, IDI

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

Oppgavetekst

Å opprette læringsobjekter med metadata som korrekt beskriver objektet innenfor en definert kontekst er ansett for å være en vanskelig oppgave. Ulike implementasjoner av IEEE-standarden, manglende kvalitetskontroll fra repositorienes side og generelle vanskeligheter med å formidle riktig bruk og forståelse av metadata til designere av læringsobjektene er noen av de mest utbredte problemene for sluttbrukerne. Jeg vil i denne diplomoppgaven søke å implementere en XML-editor som i bidrar til å underlette disse problemene basert på Design Patterns-prinsippet.

Hovedidéen er at editoren gjennom fletting av maler i størst mulig grad skal kunne bidra med å definere konteksten for brukeren, slik at denne kan fokusere på det pedagogiske aspektet og innholdet i læringsobjektet.

Oppgaven gitt: 20. januar 2006
Hovedveileder: Arvid Staupe, IDI

Sammendrag

Læringsobjekter har i pedagogiske og datatekniske miljøer helt siden unnfangelsen stått ovenfor store utfordringer og komplekse problemstillinger når det gjelder implementering og riktig bruk av standarder. Blant de mer fremtredende og hyppig diskuterte problemene er den høye terskelen for å sette seg inn i de tekniske aspektene ved disse standardene og oppgaveteksten impliserer at en godt implementert editor kan senke denne terskelen.

Oppgaven begynner med å ta for seg generell teori og standarder knyttet til klassifisering av læringsobjekter, deriblant XML, DublinCore og emnekart, går videre til å spesifisere en håndfull forhåpentligvis gode premisser og teorier for fremtidige editorer og avsluttes med en rapport om mitt arbeid med å implementere en editor etter disse premissene, resultatene av dette arbeidet og en gjennomgang av hva som bør forbedres med prototypen og hva slags type funksjonalitet det kan være ønskelig å arbeide videre med.

Forord

Jeg vil rette en stor takk til min veileder Arvid Staupe som har bidratt til å holde motet mitt og troen på at jeg faktisk skulle få dette til oppe, på tross av gjentatte møter der jeg har kommet drassende med nyfunne problemer.

Vestlendinger trives åpenbart i motvind. Tusen takk!

Trondheim 07.07.2006
Lars Weydahl

Innholdsfortegnelse:

Kapitel 1: Bakgrunn 5

Kapitel 2 : Teori 7

2.1 Læringsobjekter

2.1.1 Hva er egentlig et læringsobjekt?

2.1.2 Beskrivelse av læringsobjekter

2.2 Design Patterns

2.2.1 Hva er Design Patterns?

2.2.2 Design Patterns og læringsobjekter

2.3 Emnekart (Topic Maps)

2.4 XML

2.4.1 Kort innføring i XML

Kapitel 3: Generelle prinsipper for editoren 13

3.1 Designmetoden må implisere og synliggjøre elementenes tilhørighet

3.2 Læringsobjekter kan nyttegjøre seg Design Patterns topp-til-bunn (top-down), hierarkisk perspektiv til editoren.

3.3 En universell standard for metadata

Kapitel 4: Arbeidet med implementasjonen 17

4.1 Valg av platform og utviklingsmiljø

4.2 Problem nummer 1

4.3 Problem nummer 2

4.4 Problem nummer 3

4.5 Ressurser underveis

4.5.1StarBasic

4.5.2 OpenOffice.org

4.5.3 Andrew Pitonyaks «Useful Macro Information for OpenOffice»

Kapitel 5: Resultater og konklusjon 23

5.1Hva kan editoren gjøre?

5.1.1Lettvekts DublinCore

5.1.2Konvertering til HTML

5.2Skjermbilder og instruksjoner

5.3Konklusjon

5.4Veien videre

Litteraturliste: 32

1.0 Bakgrunn

Problemstillingen jeg har valgt å ta for meg har utgangspunkt i at flere undersøkelser viser at bruk av standarder for metadata skaper problemer for sluttbrukerene, som er nødt til å forholde seg til en rekke tekniske spesifikasjoner og terminologi som ofte er utenfor deres kompetanseområder. Mest spesifikt er det vanskelig for skapere av læringsmateriell å forholde seg til metadata som beskriver de datatekniske relasjonene et læringsobjekt [1], eksempelvis unik ID spesifikasjon i LOM og SCORMS teknisk krevende definisjoner av objekthierarkier og pakking.

Premisset for denne oppgaven og dens problemstilling er troen på at forenkling av innhenting og spesifisering av metadata er et godt virkemiddel for å senke terskelen for å kunne nyttiggjøre seg de mulighetene som disse standardene gir. En editor som er enkel å bruke, som er knyttet direkte til, gjerne som en del av, produktet man produserer læringsobjektene i og som kan skjule og automatisere konstruksjonen av teknisk komplekse relasjoner var målsettingen.

Et aspekt ved problemstillingen som det er viktig å ta hensyn til er den pågående debatten om hvem det er som faktisk skal designe og sette sammen læringsobjekter i fremtiden. En del krefter tar til orde for at det å lage og beskrive disse objektene er en jobb for eksperter og at folk flest verken har kompetanse eller lyst til gjøre denne jobben, mye på samme måte som folk ikke ønsker å lage sine egne bøker.

Duval og Hodgkins har i artikkelen Metadata Matters[2] tidligere tatt opp dette problemet og deres konklusjon er at miljøet trenger tilskudd av interesserte amatører og sier rett ut at det er en myte at amatører ikke kan lage gode metadata. Andre argumenter har gått etter linjene om at det er dumt å trekke ansvaret for klassifiseringen av informasjon vekk fra de som har opprettet den. På samme måte velger jeg å tolke dette som gode argumenter for å designe en editor som er enkel og naturlig i bruk.

Det er likevel ikke til å unngå at mye av teorien bak klassifisering av data både er kompleks og abstrakt, jamnfør semantisk web og emnekart. Det er tydelig at veien er lang før man har definitive standarder for hvordan man skal beskrive informasjon. I det neste kapitlet ser jeg rask over en håndfull av begrepene og teoriene som er sentrale for denne oppgaven.

Kapitel 2 : Teori

I dette kapitlet gir jeg en kort innføring i sentrale begreper for oppgaven.

2. 1 Læringsobjekter

2.1.1 Hva er egentlig et læringsobjekt?

Definisjonene av et læringsobjekt er mange og bruken av begrepet er ikke entydig i fagkretsene det brukes i. IEEE Learning Technology Standards Committee(2000) definerer et læringsobjekt slik:

> *"Any entity, digital or non-digital, which can be used, reused or referenced during technology supported learning."*

David Wiley [Wiley 2000] velger å begrense omfanget av definisjonen, innenfor rammene av en digital kontekst, til å være:

> *"Any digital resource that can be reused to support learning."*

Denne definisjonen er fremdeles svært omfattende og relativt vag, og det er forståelig at det har vært en del kontrovers i utviklingsmiljøene om hva et læringsobjekt er og hva det bør være. Artikler som «Three Objections To Learning Objects» [3] belyser begrepets opphav, objektorientert programmering, og forsøker å forklare hvorfor både utviklere og sluttbrukere har vanskelig for å være konsise, og ikke minst enige, i sine beskrivelser av disse objektene og hva som kjennetegner dem.

Eksempler på læringsobjekter er alt fra en kakeoppskrift lagret digitalt til en kompleks HTML-side med FLASH-animasjoner og utfyllende litteraturlister.

2.1.2 Beskrivelse av læringsobjekter

En essensiell del av Wileys definisjon, og noe av hovedpoenget med digitale læringsobjekter i dag, er gjenbrukbarhet. Idealet er at læringsressursene skal beskrives så konsist og objektivt at det vil være lett å sette sammen fagmateriale fra modulære enheter utfra disse beskrivelsene.

Eksempelvis kan et mattefag i den videregående skole bestå av moduler for trigonometri og kalkulus, mens et tilsvarende kjemifag kan ha moduler fra partikkelfysikk og organisk syntese. I et fysikkfag kan det tenkes at deler av kalkulus- og partikkelfysikkinformasjonen kan gjenbrukes og det er her det er meningen at beskrivelsen av objektene skal være såpass gode at man skal kunne avgjøre om deler av en, eller en hel, modul kan brukes i en ny kontekst.

Av nødvendighet har utviklere av standarder for digital lagring av læringsobjekter definert tekniske subsett av læringsobjekter og måter å beskrive dem på, som LOM, SCORM og CanCore. Den gjennomgående trenden er å beskrive et læringsobjekt gjennom skjematisert metadata. Dette er hovedsaklig implementert i XML, fordi dette språket er godt egnet til å definere dynamiske skjemaer. XML beskrives kort et eget delkapitel.

De fleste standarder definerer også granulariteten i læringsobjektene, med andre ord: De differensierer mellom et overordnet læringsobjekt og ressursene disse bygger på. Hovedskillet går på linjene mellom et overordnet, kompositt læringsobjekt som består av andre sammensatte læringsobjekter og nederst på stigen finner man ressurser som bilder og rene tekstelementer. Eksempelvis har SCORM egne funksjoner tilgjengelige som identifiserer relasjoner mellom læringsobjekter inne i et større læringsobjekt.

2.2 Design Patterns

2.2.1 Hva er Design Patterns?

> "*A pattern describes a problem which occurs over and over again in our environment, in such a way that you can use this solution a million times over,without ever doing it the same way twice."*
>
> Christopher Alexander, *The Timeless Way of Building[4]*

I bøkene *The Timeless Way of Building*, *A Pattern Language* og *The Oregon Experiment*, beskriver arkitekten Christopher Alexander (et al.) et strukturert språk for å beskrive oppbygging og planlegging av områder og bygninger. Fundamentet for språket er entiteter han kaller Design Patterns, strengt strukturerte beskrivelser av spesifikke arkitektoniske elementer.

Kort oppsummert inneholder et Design Pattern et definert problem, problemets kontekst og en foreslått, gennerell løsning på problemet. Det mest interessante aspektet ved denne oppbyggningen er at konteksten defineres utfra relasjoner til andre Patterns, hovedsaklig hvilke Patterns som utfyller andre. Dette skaper et hierarki av Patterns der et overliggende arkitektonisk element, oftest en by, fylles ut av delelementer.

2.2.2 Design Patterns og læringsobjekter

Et Design Pattern fyller kriteriene til et læringsobjekt generelt men er mye klarere definert. En av de klare fordelene til et Design Pattern er at konteksten bestemmer hvilke Patterns man kan bruke. I denne oppgaven vil jeg argumentere for at designet av en editor for læringsobjekter kan dra nytte av mange av de samme prinsippene som Design Patterns.

Patterns har også svært mye til felles med objektorientert programmering og en bok med navnet Design Patterns: Elements of Reusable Object-Oriented Software [4] har en høy stjerne hos programmerere med et systemutviklingsperspektiv, først og fremst fordi boken beskriver gode, gjenbrukbare løsninger på problemer man ofte møter.

2.3 Emnekart (Topic Maps)

Emnekart er en måte å beskrive et informasjonsobjekt på som skiller seg markant fra de skjematiske bindingene de fleste metadatastandarder spesifiserer. I et emnekart defineres det relasjoner mellom ulike aspekter av informasjonen. I motsetning til taksonomiske oppbygginger tillater emnekart brukeren å definere relasjoner selv og disse relasjonene danner etter hvert nettverk med lokale samlinger av relasjoner der informasjon tenderer mot et samlende emne.

Teknologien Emnekart er definert i ISO13250 og består av emner, assosiasjoner og forekomster. Emnene representerer temaer (subjects) som informasjonsressursene handler om. Et tema er noe abstrakt som kan gjøres mer konkret ved hjelp av et emne. Et emne er et element i emnekartet vist med <topic>-elementet, og dette representerer (handler om) et tema.

Assosiasjoner bidrar til å knytte emner sammen, de fungerer effektivt som lenker mellom emnene i emnekartet. Et viktig skille å merke seg fra vanlig hypermedia er at assosiasjonene eksisterer uavhengig av emnene. De er separate entiteter og binder emnene sammen uavhengig av forekomstene som eksisterer til det aktuelle emnet.

Til sist har vi forekomster, som er konkrete informasjonsressurser som gir opplysning om tilknyttede emner. En forekomst i et emnekart vil som regel være målet for søket. Eksempler på forekomster er artikler, digitalisert video og sanger.

2. 4 XML

XML står for eXtensible Markup Language og er en svært aktuell standard i skrivende stund. Som HTML er det et markeringsspråk, dvs at man definerer elementer gjennom beskrivende tags, men XML lar deg i tillegg definere dine egne tags. Dette gjør brukeren effektivt i stand til å lage egne standarder ved å kategorisere og sortere elementer etter egendefinerte skjemaer. Både HTML og XML er å regne som forenklede versjoner av Standard Generalized Markup Language (SGML). SGML blir ansett som for tungt og komplekst til å være effektivt i de kontekstene XML og HTML benyttes, grunnet komplisert syntaks og regler for å definere metadata. Formålet med XML er å beholde utvidbarheten til SGML men samtidige forenkle implementasjonen.

XML i seg selv tilbyr ingen funksjonalitet utenom definering av tags og tilhørende elementer, men kan brukes til å implementere databasefunksjonalitet og objektbeskrivelser som for eksempel metadata. Om man skal gjøre mer med elementer som er beskrevet eller gjenkjent må man bruke en parser, eksempelvis en nettleser.

2.4.1 En kort innføring i XML:

Siden man definerer egne tags i XML kan man også gi dem et arbitrært navn, med unntak av enkelte beskyttede ord. Dette vil si at man kan bruke intuitive beskrivelser, for eksempel kalle et bokelement for <bok>. Alt som står mellom den begynnende og avsluttende tagen, i vårt tilfelle </bok>, vil bli definert til å høre til bokelementet. Et element kan ha unike attributter, typisk et isbn-nummer og disse defineres innenfor selve tagen <bok isbn="dE16381-v3L1">.

Dersom man definerer et nytt element innenfor et annet vil det dannes et hierarki der de indre elementene tilhører det nærmeste ytre elementet. Dette gjør det enkelt å konstruere databeholdere og systematisere informasjon.

Eksempel på et bokelement i en database:

```
<bok isbn="dE16381-v3L1" type="diktsamling" registrert="2002-11-23">
<navn>Om menn og mugg</navn>
<bilde forside="Omom.jpg" type="jpg" x="476" y="226"/>
<utgiver id="ad242">Askedahl Forlag</utgiver>
<merknader>Side 27 revet ut av kunde <kunde id="m.schanche"/></merknader>
</bok>
```

For utdypende informasjon om XML, se http://www.ucc.ie/xml

Kapitel 3: Generelle prinsipper for editoren

Basert på den teoretiske bakgrunnen fra forrige kapitel vil jeg her prøve å trekke noen konklusjoner for hva som vil være viktig for en god editor. Det må igjen bemerkes at noen designprinsippene jeg trekker frem i dette kapitlet er av en relativt kompleks art og at det på langt nær er realistisk å implementere alle i en diplomoppgave. Dette er ikke ment som en unnskyldning for å havne langt vekk fra den platonsk ideelle editor men snarere en anerkjennelse av at feltene er teoretisk tunge og at det kan dukke opp uforutsette problemer.

3.1 Designmetoden må implisere og synliggjøre elementenes tilhørighet

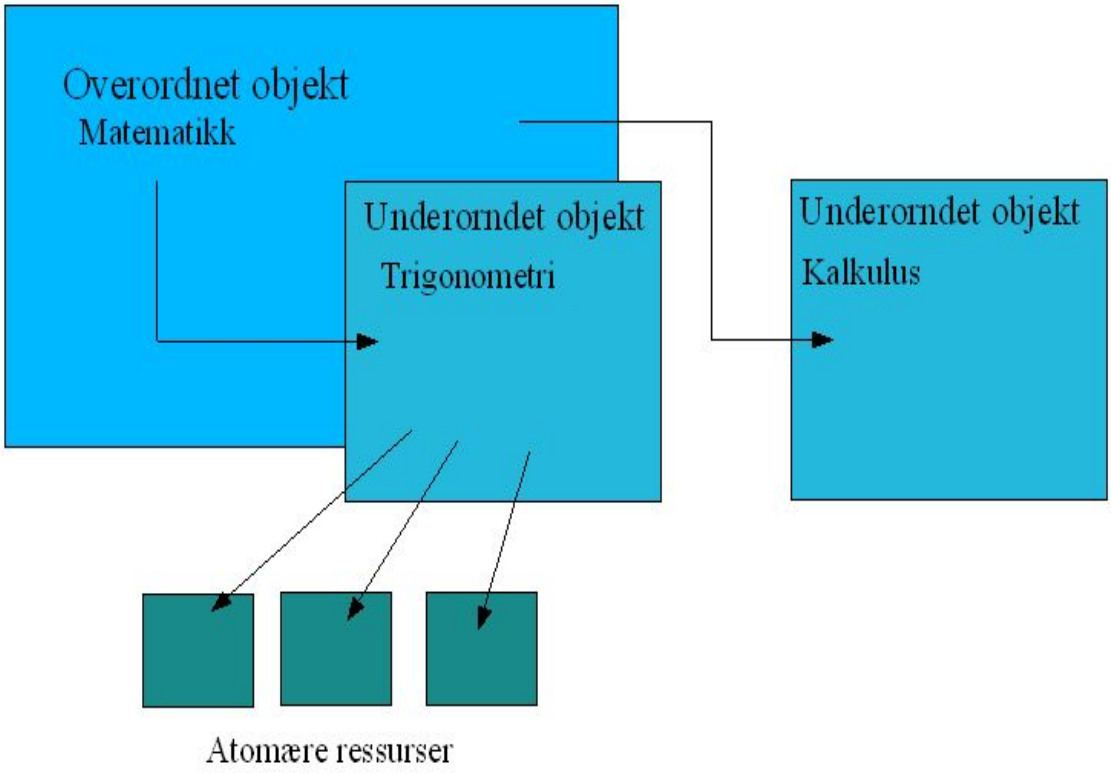
Siden hensikten med editoren skal være å forenkle prosessen er det viktig å visualisere relasjonene mellom objektene brukeren jobber med. Deler av problemet med teknisk metadata er at relasjonene er såpass abstrakte at det kan være problemer for nybegynnere å skjønne sammenhengen mellom ulike læringsobjekter utover det intuitive.

Som et konkret forslag ønsker jeg å designe en editor som dynamisk definerer relasjoner utfra hvilke læringsobjekter som blir plasser sammen. Om et læringsobjekt som omhandler disseksjon av frosker implementeres i et objekt som omhandler biologi på et mer generelt plan skal det automatisk bli definert at det første objektet omfattes av det andre.

3.2 Læringsobjekter kan nyttegjøre seg Design Patterns topp-til-bunn (top-down), hierarkisk perspektiv til editoren.

Utgangspunktet for en topp-til-bunn tilnærming er forutsetningen at designeren av et læringsobjekt vet hva han ønsker å formidle, at han har en klar idé om hva det er han ønsker å lage. Dette er grunnlaget for det øverste objektet i hierarkiet. Dette objektet definerer konteksten for de underliggende læringsobjektene og metadata knyttet til klassifisering av dette objektet arves nedover i hierarkiet etter et definert sett med regler.

Eksempelvis vil alle objekter underlagt et objekt som omhandler emnet aritmetikk arve denne kategoriseringen etter regler definert av standarden. De minste, og enklest beskrevne, elementene vil være atomære, f.eks.illustrasjoner, og disse vil kun kategoriseres etter hvilke emner deres overordnede objekt har. Enhver automatisk generert metadata vil naturligvis være åpen for redigering av sluttbrukeren.



Dette bildet illustrerer hierarkiet jeg ser for meg: Et overordnet objekt videresender deler av sine metadata, f.eks. informasjon om designeren, dato etc. Deler av denne informasjonen sendes igjen videre til de atomære ressursene. Relasjoner mellom objektene Trigonometri og Kalkulus vil likeledes genereres fordi de er del av samme overordnede objekt.

En nyttig konsekvens av byggingen av disse hierarkiene er at relasjonene mellom læringsobjektene kan føres inn i lokal database der objektene indekseres etter emnerelevans. Det er viktig å merke seg at disse dataene ikke nødvendigvis er de samme som lagres i metadataen i det endelige beskrevne objektet, men snarere et verktøy for videre katalogisering i et globalt repositorium.

3.3 En universell standard for metadata

Dersom et repositorium, eller en bruker, ønsker å kombinere læringsobjekter som er lagret med forskjellige standarder for definering av metadata er man avhengig av at disse standardene er kompatible på ett eller flere nivåer. Konvertering mellom ulike standarder er en kompleks problemstilling og det er viktig å minimalisere tap av metadata ved overføring fra et format til et annet.

Det er mulig å lage mappinger mellom individuelle standarder, men dette vil i liten grad være egnet dersom repositoriene selv skal kunne definere standardene som skal brukes. Dette vil medføre at man må definere slike mappinger for alle standarder man ønsker å kunne innlemme i basen.

Alternativet er å definere én mapping mellom standarden og en felles, universell måte å lagre data på. Denne funksjonaliteten ser jeg for meg implementert med emnekart (topic maps), slik at metadataene lagret i globale repositorier kan konverteres til de formatene som er ønskelige. Søk etter læringsobjekter basert på metadata vil dermed være uavhengig av standarden de blir lagret i og kriteriet for konvertering til en annen standard vil kun være avhengig av om bruker har tilgang på mapping mellom emnekartet og den standarden man ønsker å overføre dataen til.

Artikkelen «Living with topic maps and RDF» [6] diskuterer muligheten for å konvertere mellom mer rigide metadatastandarder og emnekart. Konklusjonen i denne artikkelen er at dette er mulig så lenge mappingene er vokabularspesifikke, men en implementasjon av dette er imidlertid såpass kompleks og tidkrevende at det blir uhensiktsmessig i forhold til denne oppgaven.

Kapitel 4: Arbeidet med implementasjonen

Arbeidet med implementasjonen var i stor grad preget av rykk og napp, med uforutsette hindre som dukket opp med jevne mellomrom. I retrospekt er det lett å se at jeg gjorde en del valg som jeg burde ha tatt meg bedre tid til å hente informasjon om.

4.1 Valg av platform og utviklingsmiljø

Mitt valg av platform og utviklingsmiljø var diktert av følgende kriterier:

1. **Tilgjengelighet**
   Er platformen/miljøet fritt tilgjengelig? Jeg anser det som et stort pluss at begge deler er gratis, lovlig å distribuere i seg selv og at det å distribuere egenimplementerte moduler/utvidelser til de valgte standardene er tillatt. Bakgrunnen for dette er at læringsobjekter ideelt sett skal kunne konstrueres av hvem som helst og at bruk av et proprietært produkt begrenser muligheten for dette.

2. **Godt dokumentert utviklingsmiljø**
   Ettersom mengden koding kommer til å bli anseelig er det ønskelig at utviklingsmiljøet er åpent og veldokumentert.

3. **Familiaritet med platformen**
   For å ikke gjøre arbeidet med implementasjonen mer tidkrevende enn nødvendig vil jeg prioritere platformer jeg har tidligere erfaring med.

4. **Pragmatikk**
   Dersom enkelte produkter tilbyr ferdige løsninger for nødvendige men ikke sentrale deler av oppgaven, som f.eks. funksjoner for GUI, filbehandling og databasesystemer, må dette anses som en stor fordel. Det er liten vits i å gjøre basale oppgaver om igjen, særlig i en oppgave som har potensiale til å bli svært omfattende dersom alt skal lages fra bunnen av.

Følgende kombinasjoner av platformer/utviklingsmiljø vil vurderes:

- **Utvikle editoren fra bunnen av for Windows XP i C++**

  Evaluering av alternativ utfra ovennevnte kriterier:

  Tilgjengelighet: Svært god

  Både kompilatorer og programvarebibliotek er tilgjengelige via OpenSource-miljøet og den ferdige editoren vil være distribuerbar etter egen diskresjon.

  U**tviklingsmiljø:** Godt

Generell bruk av C++ og hvordan lage applikasjoner i dette språket er godt dokumentert, med literatur tilgjengelig både på universitet og internett.

Familiaritet med platformen: Dårlig

Jeg har lite erfaring med programmering av applikasjoner i C++.

Pragmatikk: Dårlig

Selv om det finnes tilgjengelige bibliotek for både GUI, filbehandling og databaser på nettet vil sammensying av disse måtte gjøres manuelt. Dette vil i verste fall ikke ta mye kortere tid enn å skrive delene fra bunnen og er etter min bedømming lite sannsynlig å få til innenfor oppgavens tidsramme.

Konklusjon: Ikke egnet

- **Implementere funksjonaliteten som en plugin til en eksisterende, kommersiell xml/html-editor**

  Evaluering av alternativ utfra ovennevnte kriterier:

  Tilgjengelighet: Dårlig

  De fleste gode WYSIWYG-editorer på markedet (f.eks. XMLSpy og Stylus Studio) er proprietære og hindrer fri distribusjon/bruk av det endelige produktet.

  Utviklingsmiljø: Svært dårlig

  XMLSpy har et innebygd skriptspråk for spørringer og utforming av enkle makroer men ingen av de kommersielle produktene har åpne API/SDK'er.

  Familiaritet med platformen: Dårlig

  Siden API/SDK ikke er tilgjengelig begrenser det utvikling til skripting, noe som erfaringsmessig kan være vanskelig å få til å gjøre eksakt hva man vil.

  Pragmatikk: God

  Flere av editorene har funksjonalitet som ligger tett opp mot det jeg ønsker å få til. Dersom skriptspråket er kraftig nok vil implementering av ekstra funksjonalitet gjennom dette språket være et alternativ til å skrive kode fra bunnen.

  Konklusjon: Ikke egnet

- **Implementere editoren som en plugin til OpenOffice**

Evaluering av alternativ utfra ovennevnte kriterier:

Tilgjengelighet: Svært god

OpenOffice er gratis å laste ned og distribusjon av en egenprodusert modul til denne vil falle under GNU-lisens, noe som gjør den fritt tilgjengelig for alle.

Utviklingsmiljø: Svært godt

OpenOffice har API/SDK som inkluderer funksjoner både i C++, Java og et eget internt programmeringsspråk, StarBasic, med kjøretidstilgang på metodene i API'en. I tillegg finnes omfattende dokumentasjon og et aktivt, åpent utviklermiljø med forum.

Familiaritet med platformen: God

Jeg har lang erfaring med Java som programmeringsspråk.

Pragmatikk: Middels

OpenOffice har implementert en GUI som vil fungere godt som utgangspunkt for min editor men det kommer til å bli en god del arbeid med funksjonalitet og sammensying.

Konklusjon: Dette fremstår som det beste alternativet.

Valget falt altså på OpenOffice, Suns gratisalternativ til Microsofts Office Suite. Dette var et valg som i ettertid skulle vise seg å ha et par svært merkbare negative konsekvenser for arbeidet med editoren. På tidspunktet valget ble tatt hadde jeg dårlige forutsetninger for ta avgjørelsen og det er bare å legge seg flat for etterpåklokskapen.

4.2 Problem nummer 1: Problemer med å implementere eksterne Javaklasser.

Etter å ha strevd i to uker med en svært kompleks og lite intuitiv API , beslutter jeg meg for å skrinlegge planene om bare å bruke OpenOffices GUI som wrapper for min egen editor. Suns API på OpenOffice er stor og komplett men organsiert på en slik måte at det kreves erfaring med de svært mange containerklassene for effektivt å kunne kommunisere med OO-objektene.

Jeg bestemmer meg etter en stund for å benytte meg av StarBasic, et internt språk som gir kjøretidstilgang til funksjonene fra Java-API'en og er et relativt

kraftfullt verktøy for bygging av komplekse makroer. Jeg beslutter meg for å implementere editoren i form av et sett avanserte makroer i Impress, OpenOffices svar på PowerPoint. Dette er i tråd med min visjon om å implementere en editor som fungerer i same kjøretidsmiljø som applikasjonen man designer læringsobjekter med.

4.3 Problem nummer 2: Jeg innser etter å ha fordypet meg i stoffet om RDF og Emnekart at det er utenfor min rekkevidde å gjøre en god implementasjon av dette.

Denne åpenbaringen får meg til å fravike idealet om å mappe maler for standarder opp mot emnekart, slik at databasene som skal bygges blir nødt til å differensieres og deles opp etter hvilke standarder læringsobjektene er definert etter. Jeg begynner på dette tidspunktet å innse at jeg har tatt meg vann over hodet.

4.4 Problem nummer 3: Impress støtter ikke databasefunksjonalitet.

Etter lenge å ha slitt med å få kontakt med en SQL-database får jeg bekreftlse fra OpenOffice.orgs brukerforum om at det ikke er mulig å kommunisere med en ekstern database innenfor rammene av Impress gjennom StarBasic. Frustrasjonen blir et faktum etter noen iherdige dager med forsøk på workarounds.

4.5 Ressurser underveis

4.5.1 StarBasic

All programmeringen er gjort i StarBasic og editoren for denne programmeringen har vært den interne debuggeren i OpenOffice, som jeg faktisk har funnet svært god å jobbe med. Biblioteksfunksjonaliteten og måten modulene kan bygges på har vært godt tilfredsstillende. Jeg har også blitt imponert over hvor mye av Java-API'en som effektivt kan kompileres og kjøres i kjøretid. Selv bruk av relativt tunge klasser som filbehandling har gått relativt smertefritt.

4.5.2 OpenOffice.org

Dette er en community som ser ut til å leve og ånde for å svare på spørsmål fra brukere av OpenOffice. Min erfaring med å poste spørsmål på forumene har vært svært gode og mange ganger har jeg angret på at jeg ikke har henvendt meg tidligere i en prosess.

4.5.3 Andrew Pitonyaks «Useful Macro Information for OpenOffice»[7]

Denne 400-siders PDF-filen gir svar på mye av det som er vanskelig å få grep om av den offisielle API'en. For alle som ønsker å lære seg StarBasic eller OpenOffices indre anatomi anbefales denne på det varmeste. Svært mange nyttige og lærerrike kodesnutter.

Kapitel 5: Resultater og konklusjon

5.1 Hva kan editoren gjøre?

Editoren er i sin nåværende versjon veldig lite nyttig. Visjonen var å lage et system der brukeren kunne hente metadatastandarder fra repositorier, importere og bruke disse standardene til å beskrive de Impress- og PowerPointpresentasjonene de lager for deretter å dele disse tilbake til repositoriene.

5.1.1 Lettvekts DublinCore

For å illustrere at det er fullt mulig å videreutvikle denne editoren dette har jeg implementert en svært enkel versjon av DublinCore-standarden som man kan definere metadata i. Grunnet et problem med å formatere selvdefinerte datatyper til en bitstrøm har jeg dessverre ikke lykkes med å implementere funksjonalitet for å lagre disse dataene i objektform. Imidlertid kan man lagre dataene i form av XML-filer, som automatisk lagres til brukerens aktive arbeidskatalog.

Et stort minus med denne er imidlertid at man ikke kan definere flere elementer pr datatype slik editoren er nå.

5.1.2 Konvertering til HTML

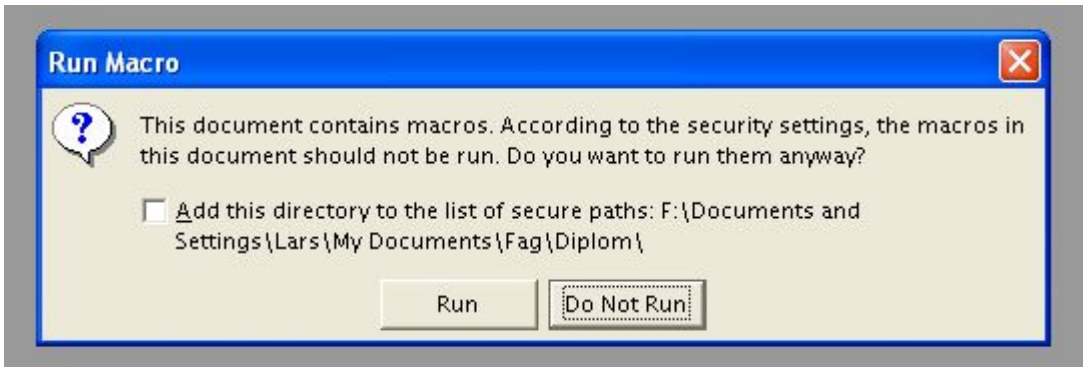
En annen funksjon som jeg la relativt stor (for mye) vekt på tidlig i oppgaveprosessene er en manuell presentasjon-til-htmlkonverteringsmekanisme. Tanken bak denne var å automatisk kunne flette sammen HTML- og XML-slik at brukeren endte opp med en komplett fil, klar for innlevering til repositoriet.

Layouten i denne er definert i CSS (Cascading Style Sheets, en formateringsstandard som komplemenerer HTML) så det er en relativivt god stuktur på outputen.

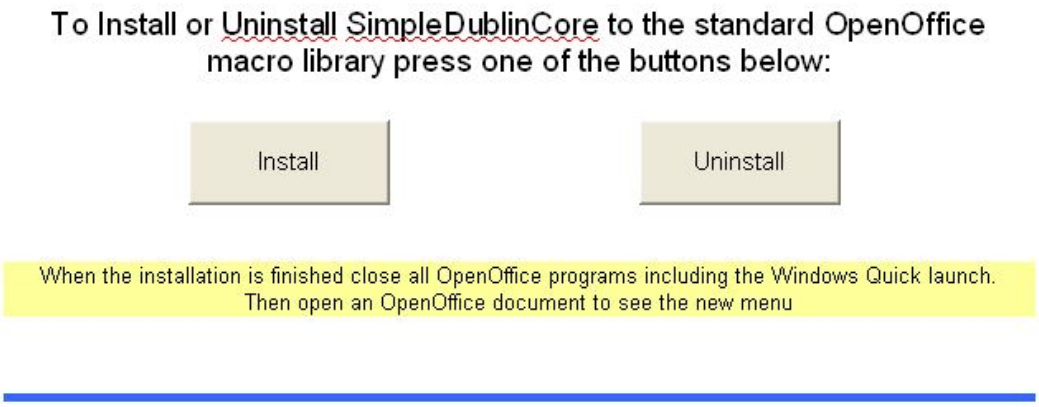
5.2 Skjermbilder og instruksjoner

Vedlagt denne rapporten leveres en zipfil som inneholder et innstallasonsdokument, SimpleDublinCore.sxc og en HTML-mal som brukes under konverterningsprosessen. For å lese SimpleDublinCore.sxc må du ha OpenOffice 1.1.1 eller høyere installert. Dette er tilgjengelig via OpenOffice.org

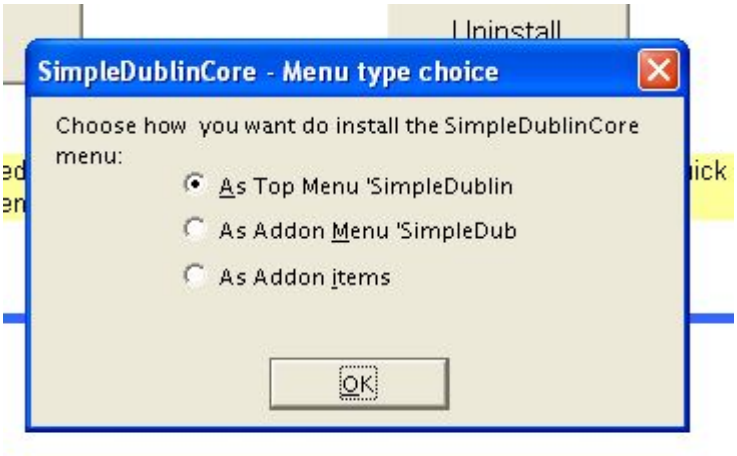
Når du åpner dokumentet vil du få en forespørsel om du ønsker å la dokumentet kjøre makroer, som vist på skjermbildet under. Velg 'Run.'



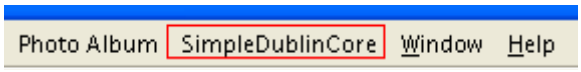
Du vil nå ha åpnet et Calc-dokument som inneholder SimpleDublinCore-biblioteket og en installsjonsmakro for denne. Bildet vil se slik ut:



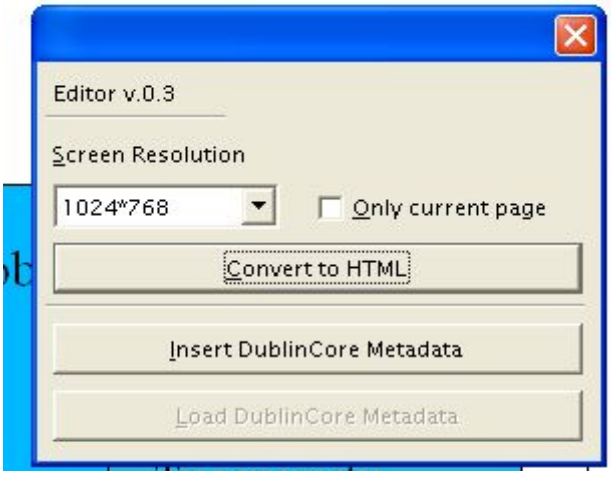
Velg install du vil få et videre valg om hvor på menyen du ønsker å ha tilgang til funksjonaliteten fra det nye biblioteket.



Personlig foretrekker jeg det øverste valget, noe som sørger for at en nye menyoverskrift vil bli tilgjengelig i Impressdokumentene på denne måten:



I denne undermenyen vil du heretter finne valgalternativet Menu som vil bringe deg til hovedfunksjonene. Det kan hende du får en engangs feilmelding som forteller deg at et metodekall er ugyldig, men dette er et engangstilfelle som skyldes at biblioteket ikke har vært lastet før.

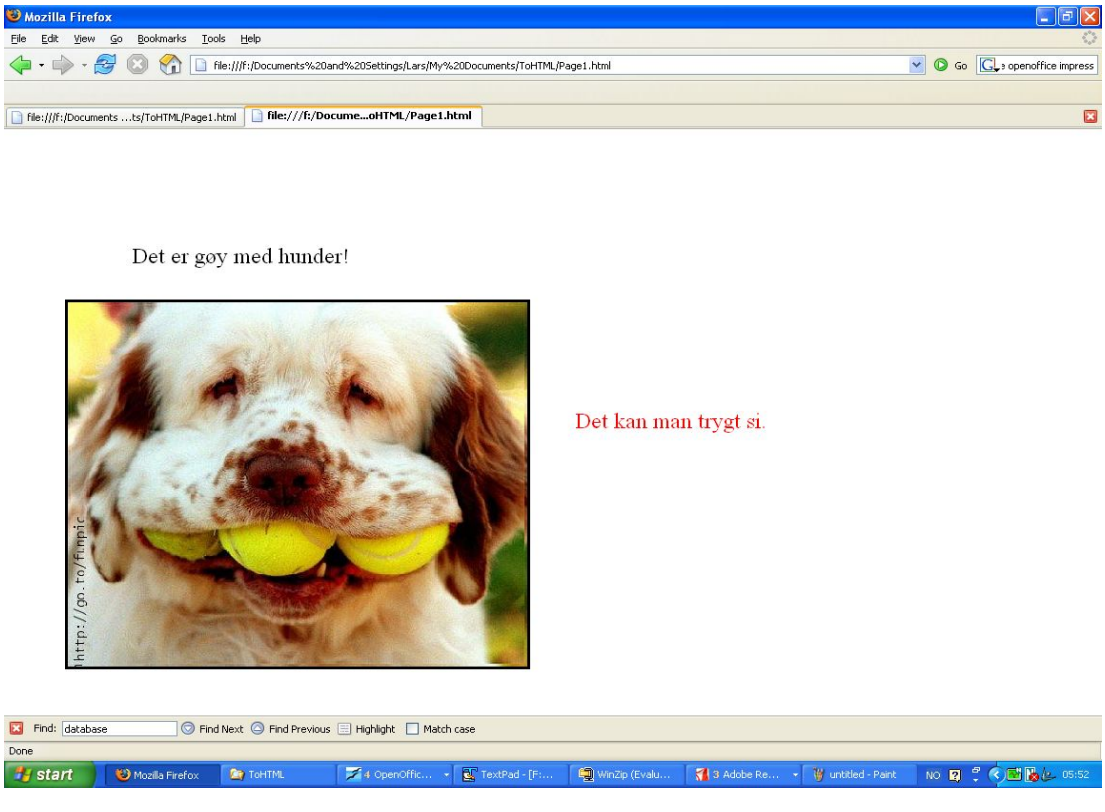


Du vil finne en ny katalog kalt ToHTML i arbeidskatalogen din (default er Mine Dokumenter/My Documents) der hver slide har blitt konvertert til en separat HTML-side.

Om du ønsker å konvertere Impressdokumentet du befinner deg i til HTML trykker du Convert to HTML etter å ha spesifisert hvilken skjermoppløsning du har. Dette er viktig fordi avstandene på skjermen er relative til oppløsningen, så med feil spesifisering av dette risikerer snodig utseende HTHML-dokumenter.

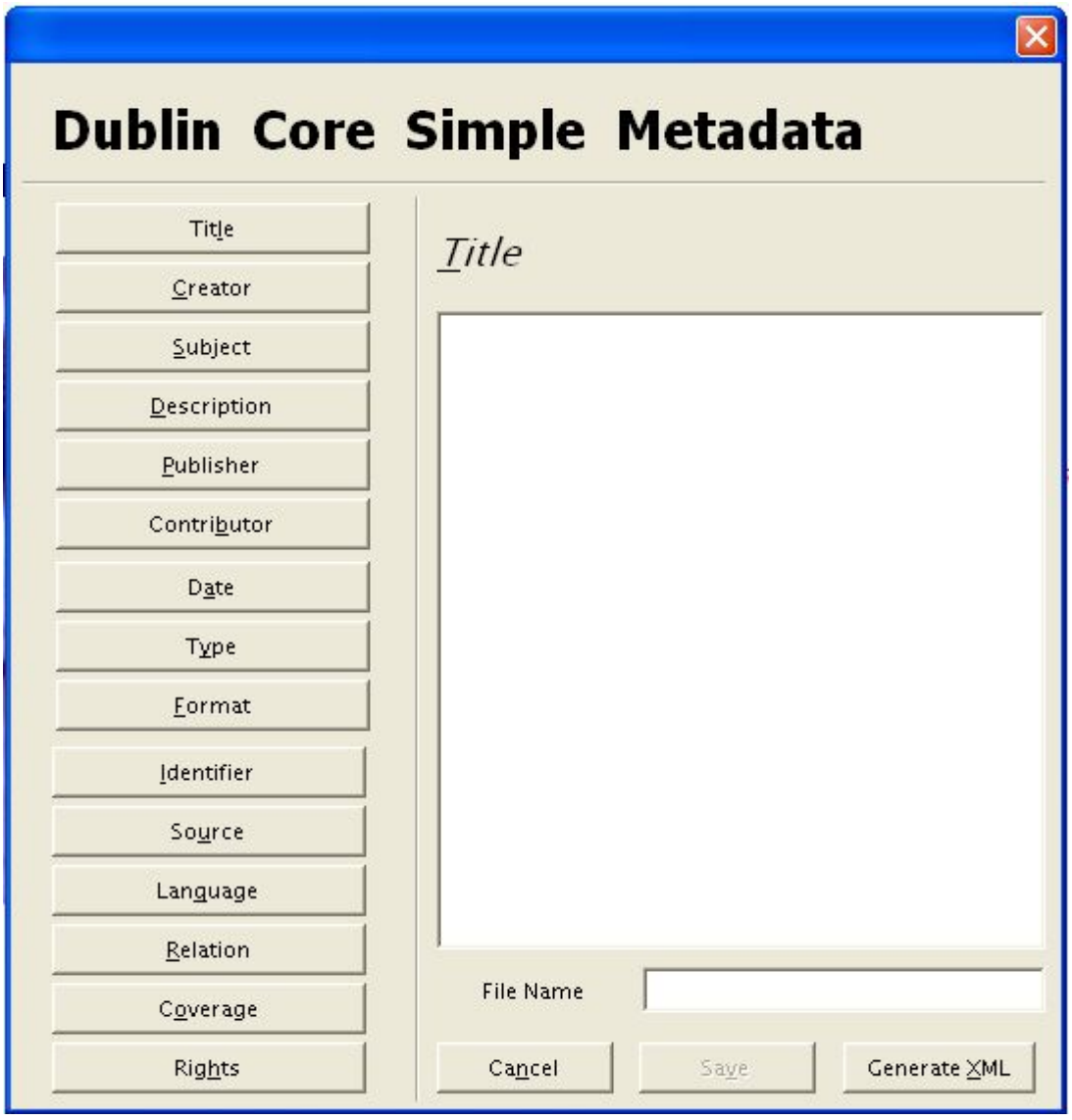
Eksempel på Impressdokument i Impress

Konvertert til HTML

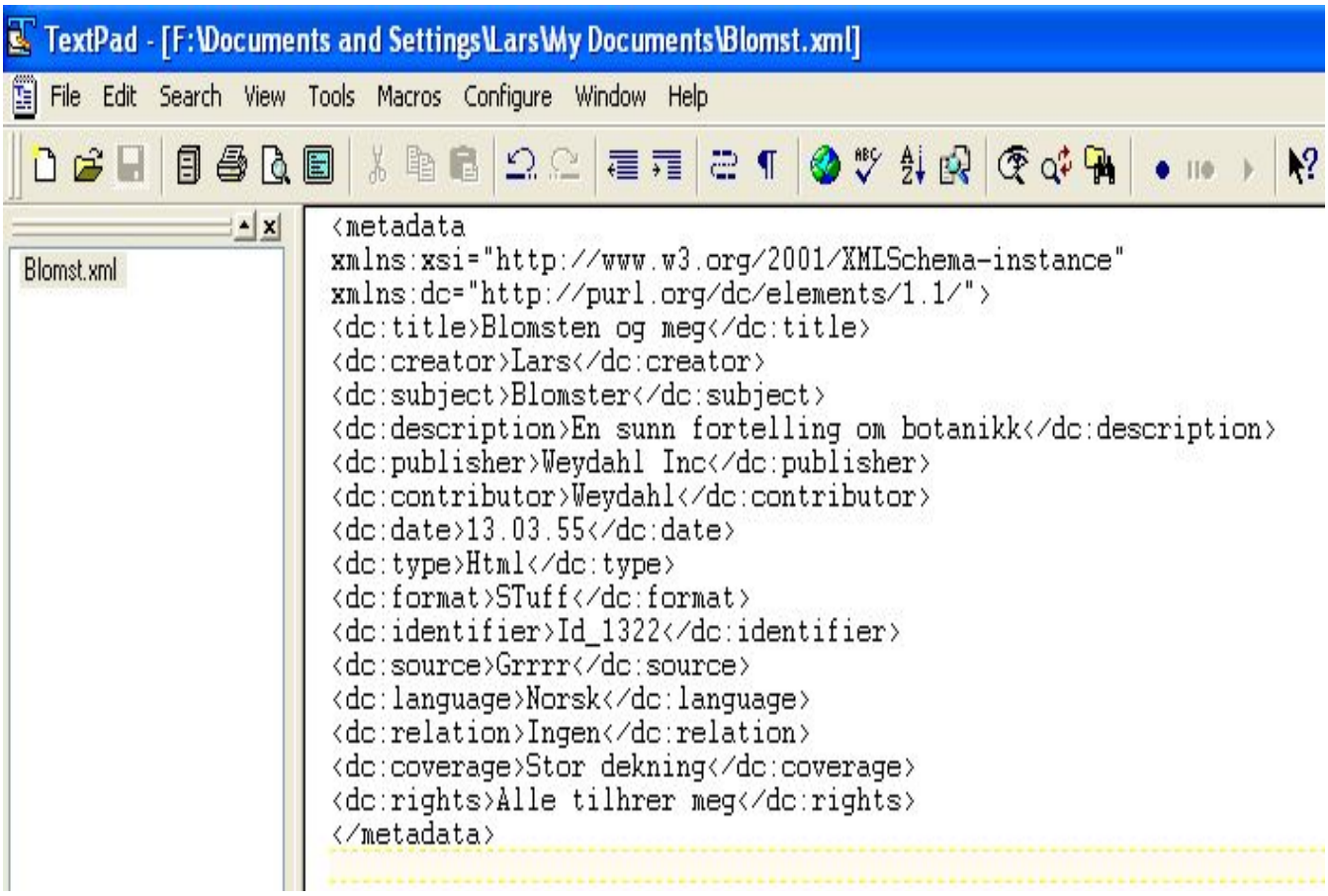


En ting man bør merke seg er at denne konverteringen noen ganger ikke gir tekst riktig sortfarge, men snarere helt hvit. Dette går imidlertid bra om man husker å farge den sorte teksten sort. Jeg mistenker at OpenOffice defaulter til svart og at dersom ingen tekstfarge er satt så defaulter min funksjon til hvit.

Skjermbilde for inntasting av DublinCore-metadata:



Strukturen metadataen får



5.3 Konklusjon

I oppgaven som forløper denne diplomoppgaven konkluderes det med at automatisk generering av tekniske metadata er et godt utgangspunkt for å hjelpe på denne situasjonen, da gjerne i form av en editor for metadata. Målsetningen var å implementere en prototyp på en editor som hjelpe brukeren å fylle ut metadata utfra programkonteksten, spesifikt gjennom fletting av maler.

Implementasjonen som foreligger som resultat av oppgaven er et godt stykke fra målsetningen. Som dokumentert videre i oppgaven har jeg måttet begrense meg til å skissere tekniske retningslinjer for hvordan en slik editor kan implementeres. Implementasjonen som følger med oppgaven er ment som et utgangspunkt, en byggesten, for en slik editor. Fra det perspektivet er det vanskelig å konkludere med annet enn at oppgaven ikke har løst problemet den definerte, men det er mitt håp å kunne se produktet videreutviklet etter de retningslinjene jeg beskriver i oppgaven.

En annen konklusjonen for meg etter arbeidet med både den bakenforliggende teorien og forsøksvise implementeringen av en slik editor er at fagfeltet er svært komplekst, at oppgaveformuleringen var mye mer omfangsrik og ambisiøs enn jeg hadde sett for meg og til sist at det å realisere et slikt produkt som jeg forestilte hadde trengt en langt bedre kartlegging av teoretiske og praktiske hindre på forhånd.

5.4 Veien videre

Det er mitt håp at min implementasjon kan fungere som en startgrop for noen som eventuelt ønsker å fortsette med å utvikle en editor i OpenOffice-miljø. Implementasjonen står fritt til å brukes og kildekoden ligger åpent i bilblioteket som installeres.

Om editoren skal videreutvikles bør utvidelse av antall standarder den støtter stå høyt på agendaen og utveksling/deling av ferdige læringsobjekter er også noe som bør forsøkes å få implementert.

Uansett er jeg realistisk nok anlagt til å se at denne implementasjonen neppe vil nyte noen bred akademisk anseelse på eget grunnlag og at basen den tilbyr ikke er det aller beste fundamentet for videreutvikling.

Etter hvert som teknologien knyttet til læringsobjekter modnes vil det likevel stilles høyere krav til editorer på markedet og jeg mener bestemt at *tanken* bak editoren var langt bedre enn utførelsen.

Litteraturliste:

1 **Erlend Øverby** (2004):

www.estandard.no - Få LOM-elementer blir brukt, viser undersøkelse

www.estandard.no/stories.php?story=04/09/13/92530184

2 **Erik Duval, Wayne Hodgins** (2004):

Metadata Matters
www.cs.kuleuven.ac.be/~erikd/chronos/2004/20041011-14_DC_Shanghai/DuvalHodginsDC2004-preprint.pdf

3 **Norm Friesen** (2003):

Three Objections To Learning Objects and E-Learning Standards
http://www.learningspaces.org/n/papers/objections.html 2.3.2,3.1

4 **Christopher Alexander et Al** (1997):

A Pattern Language
Oxford University Press

5 **Gamma, Helm, Johnson & Vlissides** (1995):

Design Patterns: Elements of Reusable Object-Oriented Software
Addison-Wesley

6 **Lars Marius Garsol**(2003):

Living with topic maps and RDF
www.ontopia.net

7 **Andrew Pitonyak**(2002-2005):

Useful Macro Information For OpenOffice
www.pitonyak.org/AndrewMacro.odt