Dynamic Presentation Generator 2.0

Utvikling av ny dynamisk presentasjonsgenerator og presentasjonsmønsterspesifikasjon

av

Bjørn Christian Sebak

Institutt for informatikk
Universitetet i Bergen
Norge

Lang masteroppgave
August 2008

Forord

Denne masteroppgaven er utarbeidet ved Institutt for Informatikk ved Universitetet i Bergen og inngår i min mastergrad i Informasjons- og kommunikasjonsteknologi, 2003-2008.

Jeg vil rette en stor takk til mine medstudenter, Bjørn Ove Ingvaldsen, Karianne Berg og Kristian S. Løvik, for godt samarbeid gjennom hele denne tiden, og til min veileder, Khalid A. Mughal for gode råd og forslag til oppgaven. En spesiell takk til Torill Hamre for grundige tilbakemeldinger på oppgavens innhold, hjelp til rettskriving og forslag til forbedringer underveis.

1 Innledning

1.1 Bakgrunn

Denne masteroppgaven er en del av JAva FjernUndervisningsprosjektet (JAFU) på Institutt for Informatikk ved Universitetet i Bergen. Formålet med dette prosjektet er først og fremst å kunne tilby nettbasert undervisning (også kalt fjernundervisning) for privatpersoner og næringslivet. Fjernstudiene har vært et tilbud ved instituttet siden høsten 1999 og fått gode resultater og positive tilbakemeldinger fra de påmeldte studentene.

Hovedsakelig er det kursene *INF100-F – Grunnkurs i programmering (Programmering I)* og *INF101-F – Videregående programmering (Programmering II)* som har blitt tilbudt. Disse kursene følger samme pensum som de tilsvarende kursene på campus, slik at fjernundervisningsstudentene kan ta eksamen på slutten av semesteret sammen med de øvrige universitetsstudentene.

For å kunne tilby disse studiene har det opp gjennom årene blitt utviklet en rekke nettbaserte verktøy for publisering av kursmateriell og annen informasjon. For å kunne drive med e-læring på et slikt nivå at man er konkurransedyktig med andre aktører på markedet, er det ikke nok med statiske nettsider for å formidle denne informasjonen. Behovet for et mer dynamisk publiseringsverktøy var sterkt, og prosjekter ble satt i gang for å få implementert et slikt system.

Opp gjennom årene har det vært flere ulike systemer tilknyttet JAFU-prosjektet. Blant dem finner vi bl.a. Java Presentation Generator (JPGEN) (Cruickshanks 2004), et system for publisering av kurssider, som ble i januar 2005 byttet ut med Dynamic Presentation Generator (DPG) (Espelid 2004). DPG-en er det systemet som er brukt i dag, og er systemet som er utgangspunktet for denne oppgaven. DPG-en bygger på ideen rundt såkalte presentasjonsmønstr (Mughal 2003), der man konsekvent skiller mellom struktur og innhold, med det mål å kunne gjenbruke strukturen for annen type innhold. De forskjellige systemene som DPG-en baserer seg på blir gjennomgått nærmere i neste kapittel.

1.2 Motivasjon

Det er ingen tvil om at det nåværende systemet har vært meget vellykket. DPG-en har demonstrert at konseptet med presentasjonsmønstre definitivt har livets rett, og at det er mange muligheter knyttet til det som enda ikke er utnyttet. Men til tross for DPG-en suksess, har man etter hvert innsett at den nåværende implementasjonen har en rekke svakheter.

Det mest problematiske med DPG-en slik den fremstår i dag er nok selve administrasjonsverktøyet, Repository Administration Tool (RAT). Det er i dette verkøyet all redigering av presentasjonenes innhold foregår, fra opplasting av ressursfiler til endring av informasjonen på nettsidene. Slik som det er nå, må personer som ønsker å endre innholdet ha kjennskap til eXtensible Markup Language (XML), og være veldig

forsiktig med hva som legges inn og fjernes fra innholdsfilene. Dersom feil blir gjort, kan det føre til at kurssidene blir ustabile, noe som først og fremst går ut over studentene, men som også skaper mye ekstraarbeid for kursets administratorer. Dette er noe som må analyseres nærmere og utbedres i den nye versjonen.

Skal man opprette nye presentasjoner basert på eksisterende presentasjonsmønstre, må man i dag gjøre en del manuelt arbeid som krever stor grad av teknisk innsikt og forståelse av DPG-systemet. Man må som regel kopiere presentasjonskatalogen fra en eksisterende presentasjon, redigere en omfattende mengde komplekse XML-filer og manuelt teste om alt er koblet opp riktig. Det er med andre ord ikke noe hvem som helst kan gjøre. I det nye systemet bør opprettelsen av nye presentasjoner kunne skje via en enkel veiviser i administrasjonsverktøyet, uten av man trenger å vite noen ting om hva som foregår i bakgrunnen.

Sist, men ikke minst, er det viktig å se nærmere på hvordan presentasjonsmønstrene er bygget opp i den eksisterende DPG-en. Siden dette systemet ble ferdigstilt i 2005 har det dukket opp nye ideer og tanker rundt presentasjonsmønsterkonseptet (Rossini og Liberati 2005), og disse må undersøkes og tas med under utviklingen av det nye systemet. Det er også gjort en del arbeid rundt utvikling av nye presentasjonsmønstre for andre sammenhenger enn fjernundervisning (Lervik Larsen 2006). Ved å se på bruken av presentasjonsmønsteret i andre sammenhenger enn nettkurs, sikres det at mønsterspesifikasjonen for det nye systemet er mest mulig fleksibel.

1.3 Målsetninger

1.3.1 Hovedmål

Oppgaven har følgende hovedmål:

1. Design av den nye presentasjonsmønsterspesifikasjonen
2. Design og implementasjon av en dynamisk presentasjonsmotor for den nye presentasjonsmønsterspesifikasjonen
3. Utvikling av redigeringsfunksjonalitet for presentasjonsinnhold
4. Utvikling av verktøy for å opprette og administrere presentasjoner
5. Vurdering av hva som er oppnådd og forslag til fremtidige utvidelser

Først og fremst er det behov for en ny spesifikasjon for presentasjonsmønsteret. Den som finnes i dag er bl.a. lite fleksibel og vanskelig å forstå. Et av målene for denne oppgaven er derfor å ta utgangspunkt i det den siste utgaven av presentasjonsmønsteret utviklet av Alessandro Rossini og Graziano Liberati (Rossini og Liberati 2005), og se på hvilke syntaktiske og semantiske forbedringer som kan gjøres.

Deretter må det utvikles en ny motor for visningen av de dynamiske presentasjonen, tilpasset den nye mønsterspesifikasjonen. Motoren bør være så generisk så mulig, slik at den kan takle alle presentasjoner som oppfyller presentasjonsspesifikasjonen. Det er lite trolig at kode fra den gamle motoren blir brukt igjen, da denne er sterk bundet til den gamle presentasjonsmønsterspesifikasjonen. Ved å utvikle motoren helt fra bunnen av vil

vi også kunne dra nytte av nye teknologier, eksempelvis Spring (The Spring Framework, SpringSource), og gode designmønstre. Enhetstesting og integrasjonstesting skal også få en større rolle enn det tilfelle var i den gamle DPG-en for å sikre god kvaliteten på kodebasen.

Tredje hovedmål omfatter utviklingen av redigeringsfunksjonalitet for innholdet i en presentasjon. I den nåværende versjonen av DPG-en måtte man logge seg inn i et separat administrasjonssystem (Repository Administration Tool, RAT) (Espelid 2004) når innholdet skulle endres. Ikke bare var grensesnittet i dette systemet vanskelig å bruke, det var også veldig lett å gjøre fatale feil i viktige systemfiler, noe som gjorde at DPG-en ble ustabil. Derfor må det i den nye versjonen utvikles et godt redigeringsverktøy som lar brukeren endre informasjon gjennom enkle, brukervennlige HTML-skjema.

Det fjerde hovedmålet er å utvikle et verktøy for å opprette og administrere presentasjoner. Verktøyet skal la administratorer enkelt opprette nye presentasjoner, samt slette og konfigurere eksisterende presentasjoner gjennom et lettfattelig nettbasert grensesnitt. Dette verktøyet skal være like enkelt å bruke som verktøyet for å redigere innhold, og det skal derfor ikke være nødvendig for en administrator å redigere konfigurasjonsfiler direkte. For eksempel bør det å opprette en ny presentasjon basert på et presentasjonsmønster kunne gjøres ved å følge en enkel veiviser.

Det femte og siste hovedmålet dreier seg først og fremst om å vurdere de ulike løsninger som har blitt utarbeidet i de øvrige hovedmålene. Det er svært viktig å se på hver av disse løsningene med kritiske øyne for å hindre at det nye systemet ivaretar de samme feil og mangler som DPG 1.0. En grundig vurdering der man presiserer de ulike fordeler og ulemper er derfor nødvendig. I tillegg til dette, skal vi også se nærmere på en rekke forslag til fremtidige utvidelser av det nye systemet.

1.3.2 Delmål

For å gjøre det lettere for utviklere av presentasjonsmønstre å, for eksempel opprette sider eller «views», bør det være bedre støtte for gjenbruk. I en bestemt presentasjon vil det alltid finnes sider som inneholder lik informasjon. Informasjon som gjerne går igjen på tvers av sidene er for eksempel stilark (CSS), tilgangsrettigheter, topp og bunntekst, osv. Den nye presentasjonsmønsterspesifikasjonen må altså være mer fleksibel og modulær enn den eksisterende versjonen. Dette delmålet hører i hovedsak under det første hovedmålet.

For å sørge for at kvaliteten på kildekoden som produseres holder en høy standard, vil man i utviklingen av det nye systemet ta i bruk mer moderne rammeverk og utviklingsverktøy. Spring-rammeverket (The Spring Framework, SpringSource) vil bli brukt for å gjøre det lettere å dele opp systemet i konfigurerbare, testbare moduler. Dette rammeverket har blitt svært populært i Java-verden de siste årene, og vil være et uvurderlig hjelpemiddel i utviklingen av det nye systemet. Siden dette går direkte på implementasjon, inngår dette delmålet hovedsakelig i det andre, tredje og fjerde hovedmålet.

Det er også ønskelig å se på muligheten for utvikling av verktøy for en tredje brukertype, *pattern designer* (eller mønsterdesigner). Selv om utviklingen av et slikt verktøy vil være for omfattende til å få plass i denne masteroppgaven, vil det likevel være svært interessant å drøfte ulike problemstillinger rundt dette. Dette vil derfor bli et delmål i denne oppgaven, og vil kunne danne grunnlag for videre utvikling av et slikt verktøy. Da dette regnes som en mulig fremtidig utvidelse, går dette delmålet inn under det femte hovedmålet.

1.4 Hvordan denne oppgaven er organisert

Hvert hovedtema har sitt eget kapittel. Oppgaven har en kronologisk struktur, der innholdet i ett kapittel ofte bygger på det som ble avdekket i det forrige. Det er derfor anbefalt at man leser kapitlene i riktig rekkefølge.

Kapittel 2 – Bakgrunn for oppgaven

I dette kapittelet får man først en generell introduksjon til hva et innholdshåndteringssystem er og noen av de mest populære systemene. Deretter presenteres ideen om presentasjonsmønstre, og hvordan dette er realisert i DPG 1.0. Til slutt får man en kort oversikt over utviklingen av DPG 1.0 og hvordan denne er bygget opp.

Kapittel 3 – Analyse av DPG 1.0

Her ser man nærmere på de problemer og utfordringer som finnes i DPG 1.0. En rekke problemer diskuteres, blant annet knyttet til sikkerhet, kodekvalitet og innholdsredigering. Denne analysen danner grunnlaget for utviklingen av DPG 2.0.

Kapittel 4 – Introduksjon til det nye DPG-systemet

I dette kapittelet blir den overordnede arkitekturen for den nye DPG-en presentert. Fokuset er her på de tre viktigste delsystemene og deres hovedoppgaver. Siden disse delsystemene har et ganske stort omfang, har hver av dem fått et eget kapittel i denne oppgaven.

Kapittel 5 – Analyse av tidligere forslag til presentasjonsmønster

Her analyseres det fremste forslaget til ny presentasjonsmønsterimplementasjon, utarbeidet av (Rossini og Liberati 2005). Grunnet omfanget av dette forslaget fikk dette sitt eget kapittel. Denne analysen danner grunnlaget for utviklingen av den nye presentasjonsmønsterimplementasjonen for DPG 2.0.

Kapittel 6 – Presentasjonsmønstre i DPG 2.0

Her presenteres den nye presentasjonsmønsterimplementasjonen for DPG 2.0. Vi går her gjennom begge spesifikasjonene, samt de viktigste komponentene og deres funksjoner. Til slutt sammenligner vi den nye implementasjonen med den gamle, og peker ut de viktigste forbedringene som er gjort.

Kapittel 7 – Presentation Viewer (PV)

Her ser vi nærmere på den nye presentasjonsmotoren, kalt Presentation Viewer (PV), og hvordan den bruker mønsterspesifikasjonen sammen med XSLT og CSS for å generere XHTML som bygger opp nettsidene brukeren ser. Vi vil se nærmere på hvordan rendringsprosessen skjer og hvilke komponenter som er involvert. Til slutt ser vi på hva som er oppnådd i forhold til presentasjonsmotoren (DPE) i det gamle systemet.

Kapittel 8 – Presentation Content Editor (PCE)

Dette kapittelet omhandler den nye verktøyet for redigering av innhold i en presentasjon., kalt Presentation Content Editor (PCE). De viktigste funksjonene blir gjennomgått, og sammenlignet med det eksisterende verktøyet, RAT. Spesielt interessant er støtten for multimedia og betydelig enklere brukergrensesnitt for innholdsredigering.

Kapittel 9 – Presentation Manager (PM)

Det siste delsystemet i DPG 2.0 er selve administrasjonsverkøyet, Presentation Manager (PM). Dette systemet lar administrativt ansvarlige opprette nye presentasjoner basert på eksisterende presentasjonsmønstre, samt gjøre endringer i eksisterende presentasjoner. Vi skal se på de viktigste funksjonene og hvilke fordeler dette verktøyet har i forhold til administratorverktøyet i DPG 1.0.

Kapittel 10 – Forslag til videre arbeid

I dette nest siste kapittelet diskuteres det en rekke forslag til utvidelser av det nye systemet. I all hovedsak gjelder dette nye mønstre, støtte for plugin-moduler, samt verktøy for å gjøre det lettere å utvikle nye presentasjonsmønstre. Dette kapittelet kan være spesielt interessant for kommende masterstudenter som ønsker å videreføre JAFU-prosjektet.

Kapittel 11 – Evaluering og konklusjon

Her blir det arbeidet som er utført i forbindelse med oppgaven evaluert, sett opp mot de mål som ble satt i startfasen. Viktig lærdom blir trukket fram før oppgaven konkluderes.

2 Bakgrunn for oppgaven

Dette kapittelet er skrevet i fellesskap av følgende personer: Karianne Berg (Berg 2008), Bjørn Christian Sebak og Bjørn Ove Ingvaldsen (Ingvaldsen 2008), og inngår dermed i alle oppgavene. Årsaken er at alle hadde behov for et slikt bakgrunnskapittel med omtrent det samme innholdet. For å unngå at det ble produsert nesten identiske kapitler, ble det bestemt at dette kapittelet skulle skrives i samarbeid.

2.1 Innholdshåndteringssystemer

2.1.1 Hva er et innholdshåndteringssystem?

For å kunne definere et innholdshåndteringssystem, er det nødvendig å kunne definere og skille mellom følgende termer: *data, informasjon* og *innhold*. Disse tre begrepene er ofte overlappende definert i litteraturen, og forskere strides om betydningen av dem. For formålet til denne oppgaven defineres begrepene slik:

(Nordbotten 2008) definerer data som "symbols inscribed in formalized patterns, representing facts, observations and/or ideas, that are capable of being communicated, interpreted and manipulated by some human or mechanized process." Denne definisjonen sier implisitt at data ikke har noen kontekst, og det er heller ikke definert hvordan dataene er representert. Data som begrep er altså ikke begrenset til datamaskiner.

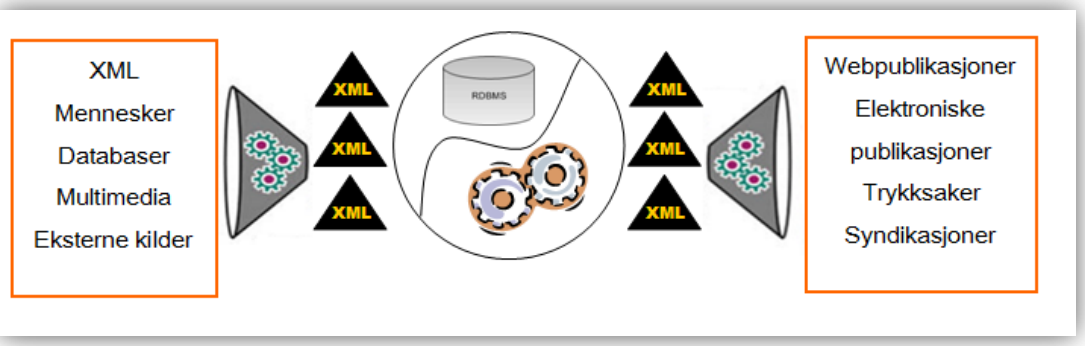
(Gould 1971) definerer informasjon som "the meaning that a human extracts from data by means of known conventions of the representation used." Informasjon har altså en kontekst, i motsetning til data. Definisjonen sier også at hva slags informasjon man får ut av data avhenger av vedkommende som tolker den – for eksempel RFC-1122, som beskriver TCP/IP-protokollen, vil gi forskjellig grad av informasjon til en nettverksprogrammerer og en bussjåfør. Dette innebærer at datamaskiner ikke – i hvert fall ikke i den formen de har i dag - er i stand til å lagre informasjon direkte. Dette synet støttes også av (Boiko 2005).

Innhold er en term som ikke er mye definert i litteraturen. (Boiko 2005) har denne definisjonen av innhold: "Content […] is information that you tag with data so that a computer can organize and systematize its collection, management and publishing". Med dette mener han at siden datamaskiner ikke er i stand til å lagre annet enn data, må vi også lagre metadata som hører til informasjonen, som inneholder data om hva slags informasjon vi har, hvordan den kan brukes og hvordan den skal presenteres til brukeren.

Betydningen av termen *innholdshåndteringssystem* definerer Boiko som "a system capable of organizing and systemizing collection, management and publishing of content". Under "collection" finner vi både skaping av nytt innhold av mennesker og innsamling av data fra ulike kilder, herunder XML-kilder med ulik oppbygning, databaser, multimedia og eksterne kilder. (se Figur 2-1) Dette steget omfatter også eventuell konvertering av data til et format innholdshåndteringssystemet kan håndtere, og aggregering av innholdet.

”Management”-delen av systemet består av fire deler: et sentralt repositorium for å lagre innholdet, som en relasjonell database eller en XML-database - gjerne kombinert med et fillager for binære filer; et administrasjonssystem som setter opp og konfigurerer systemet; en arbeidsflytdel som styrer arbeidsflyten i systemet (for eksempel skriving av utkast, godkjenning og publisering av dokumenter), og tilknytninger til eventuelle andre systemer i organisasjonen.

Å hente ut innhold og andre komponenter fra repositoriumet og sette det sammen på riktig måte for publisering er ”Publishing”-delen av systemets jobb. I dag er resultatet av publisering er ofte websider, men det bør også være mulig å produsere andre medier, for eksempel trykksaker.



Figur 2-1: Oversikt over komponentene i et innholdshåndteringssystem, basert på (Boiko 2005)

2.1.2 Relaterte systemer

Dette avsnittet gir en kort beskrivelse av noen av systemene som har lignende funksjonalitet som vi forestiller oss at DPG 2.0 skal ha. De har blitt utvalgt delvis fordi vi har jobbet med noen av dem tidligere, og delvis fordi de er mye brukt. Merk at noen av disse systemene kaller seg for innholdshåndteringssystemer, mens noen av dem kaller seg for portaler. Skillet mellom disse to begrepene er uklart, og det finnes ingen fastlagte definisjoner av hva som er det ene og hva som er det andre.

VerticalSite

VerticalSite er et kommersielt portalverktøy skrevet i Java, utviklet av det norske selskapet Enonic (Enonic). Programvaren tilbyr et komplett system for innholdshåndtering, og er basert på åpne industristandarder. Løsningen benyttes av flere store norske selskaper, blant annet Norsk Tipping, Tine og Gjensidige.

VerticalSite bruker XSLT-maler, noe som vi synes fungerer godt for å transformere data til innhold i riktig format. Applikasjonen benytter imidlertid også en egenutviklet XSLT-basert teknologi sammen med Java-klasser til å definere plugin-funksjonalitet. For oss virket dette vanskelig å forstå, vanskelig å bruke og som en ”feil” bruk av XSLT i henhold til XSLT-teknologiens formål.

Dette portalverktøyet bruker Java Content Repository (JSR-170) (Sun Microsystems) som underliggende persistensteknologi. Dette ser ut til å fungere godt, spesielt i forbindelse med XML-baserte data, siden Java Content Repository bruker XML som

internt lagringsformat og XPath som spørrespråk. Denne type teknologi kan være aktuell i den nye versjonen av DPG-en.

SiteVision
SiteVision er et annet kommersielt portalverktøy skrevet i Java, utviklet av det svenske selskapet Senselogic (Senselogic AB). Selskapet har Rikard Öberg i spissen, en kjent skikkelse i Open Source-miljøet. Systemet fokuserer på brukervennlighet og fleksibilitet, samtidig som det har mye avansert funksjonalitet. Applikasjonen benyttes av mange aktører, hovedsaklig fra Norge og Sverige, inkludert BaneTele, Levi Strauss og Biltema.

SiteVision benytter portlets (JSR-168) for å inkludere funksjonalitet på nettsider, og har en egen GUI-editor for å legge inn og plassere de ulike portlet-ene skrevet som et Java applet. Dette virker stort sett bra, og kan være et alternativ for oss når vi skal legge inn plugin-funksjonalitet i DPG-en.

Løsningen bruker den objektbaserte databasen db4o (db4objects) for interne data, og overlater til utviklerne av portlets å velge persistensteknologi selv. Applikasjonen benytter seg av Velocity (Velocity Apache Project) som språk for maler. Dette synes vi fungerer på en god måte, og et langt bedre alternativ enn å ha malene i JSPX-format, slik som DPG 1.0 har.

Plone
Plone er et Open Source innholdshåndteringssystem som er lisensiert under lisensen GNU General Public Licence (GNU) (GNU Project 2008). Det er basert på applikasjonsserveren Zope (Zope.org) og skrevet i programmeringsspråket Python.

Plone bruker sin egen teknologi for å inkludere funksjonalitet, kalt add-ons. Plone kommer med Zopes interne, objektbaserte database (ZODB) som standard persistensteknologi, men kan også settes opp med de fleste relasjonelle databaser.

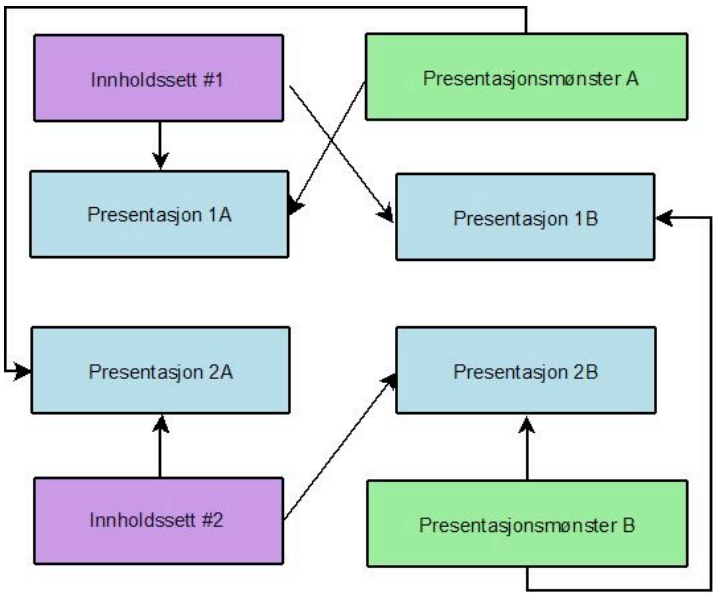
Systemet har sitt eget språk for maler. Dette innebærer at det kan spesialtilpasses applikasjonen, men også at utviklere som skal bruke systemet må lære seg et ekstra, proprietært språk for å kunne skrive maler i Plone. Dette vil vi prøve å unngå i utviklingen av den nye DPG-en så langt det lar seg gjøre.

2.2 Presentasjonsmønstre

2.2.1 Konsept

DPG-en er basert på konseptet om *presentasjonsmønstre*, en ide som ble fremlagt av Khalid Mughal i 2003 (Mughal 2003). Drivkraften bak ideen om presentasjonsmønstre var å promotere gjenbruk, både av innhold og av strukturen til nettsider. Siden JAFU-prosjektet er tett knyttet opp mot fjernundervisning, var gjenbruk av nettsider for kurs det mest fremtredende. På institutt for informatikk var situasjonen (og er enda, til en viss grad) at hver foreleser laget sine egne kurssider fra bunnen av, skrevet i HTML. Hver gang foreleseren skulle lage nye kurssider, tok de gjerne kurssidene fra et annet kurs, kopierte dem og byttet ut innholdet. Nettsider for kurs er stort sett strukturelt like: De har

en forside, et sted å poste meldinger, en side med kontaktinformasjon, en fremdriftsplan og andre elementer. Det er kun innholdet som er forskjellig, og gjerne også utseendet på sidene. Mughal så at denne strukturen kunne formaliseres i et presentasjonsmønster.



Figur 2-2: To presentasjonsmønstre og to sett med innhold kan gi fire forskjellige presentasjoner

En enkel måte å beskrive et presentasjonsmønster på, er at det er et regelsett for hvordan innholdet i en presentasjon skal *struktureres*, *renderes* og *navigeres i*. Til et presentasjonsmønster hører en eller flere presentasjoner. Hver av disse avgjør hva slags *innhold*, *layout* og *utseende* presentasjonen har, i tillegg til hvilket mønster den bruker. Med andre ord kan flere presentasjoner som deler samme presentasjonsmønster ha vidt forskjellig innhold, layout og utseende. Som vi ser i figur 2-2, kan både innhold og presentasjonsmønster gjenbrukes. Alt som skal til er at innholdet er strukturert i henhold til strukturen presentasjonsmønsteret dikterer.

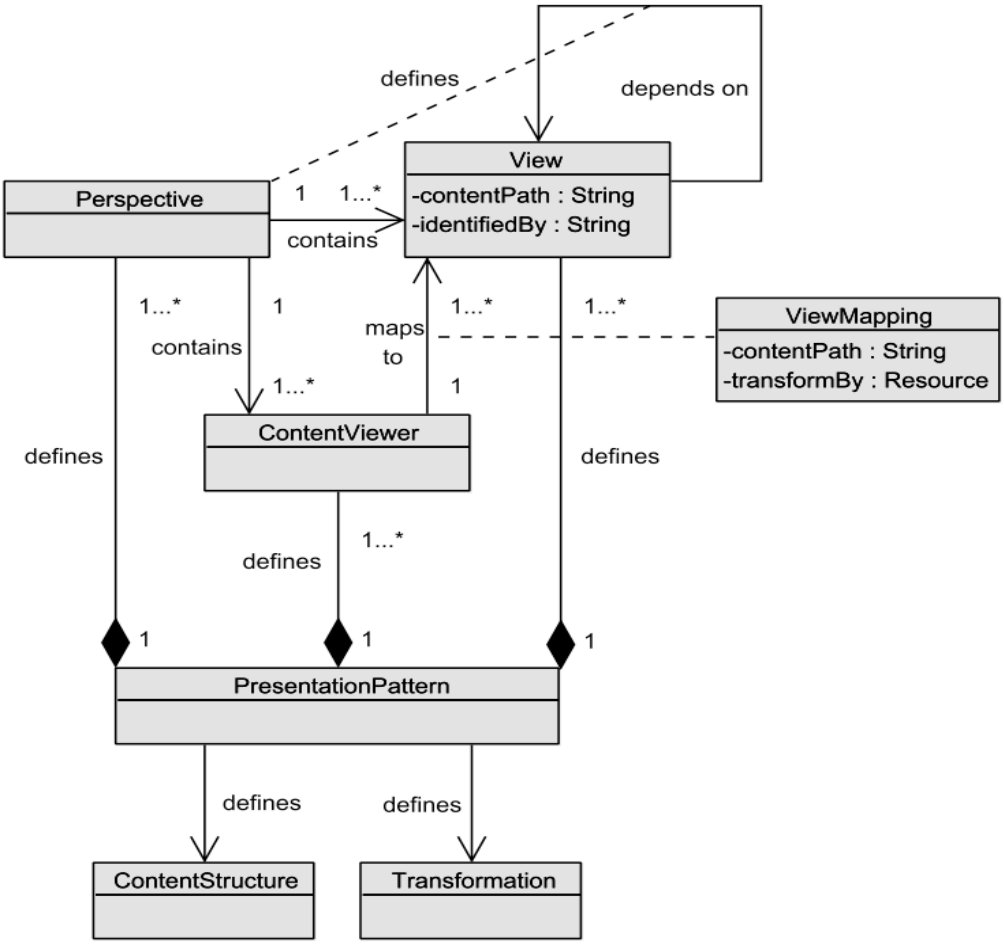
2.3 Presentasjonsmønstre i DPG 1.0

Både presentasjonsmønster- og presentasjonsspesifikasjonen ble begge definert ved hjelp av XML-dokumenter. Strukturen på disse XML-dokumentene ble validert ved hjelp av XML Schema, dette for å hindre at XML-dokumenter med ugyldig struktur blir lastet inn i systemet.

2.3.1 Presentasjonsmønsterspesifikasjonen

Presentasjonsmønsterspesifikasjonen består av tre hovedkomponenter. Figur 2-3 viser sammenhengen mellom følgende hovedkomponentene:

- View
- Content-viewer
- Perspective



Figur 2-3: Komponentene i presentasjonsmønsterspesifikasjonen (Espelid 2004)

View

Formålet med et view er å definere logiske navn til bestemte steder i datastrukturen. Det logiske navnet kan deretter brukes av de andre komponentene i mønsterspesifikasjonen. XPath-uttrykk brukes for å bestemme posisjonen til stedet informasjonen ligger.

I eksempel 2-1 ser man hvordan et view kan defineres. Først tildeler man view-et et logisk navn, `studentsView`. Deretter benyttes XPath for å få tak i alle personer av typen student. Til slutt oppgir man hvilket element som skal benyttes som unik id for å skille de forskjellige studentene fra hverandre.

```
<view name="studentsView">
    <content-path>//person-list[@type='student']</content-path>
    <identified-by>@personID</identified-by>
</view>
```

Eksempel 2-1: Definisjonen av et view

Content-Viewer
Dette komponentet bestemmer hvordan et view skal renderes. Den tar det logiske navnet til et view og knytter det opp mot en bestemt transformasjonsfil (XSLT). Det er mulig å opprette forskjellige content-viewere for ett og samme view, noe som gjør det mulig å rendre samme informasjon på forskjellige måter.

I eksempel 2-2 definerer man en content-viewer med navnet `listOfAllStudentsViewer`, som benytter seg av view-et `studentsView`, som er vist i eksempel 2-1. I attributten content-path, velger man hvilken node som skal være startnoden som skal brukes under XSLT-transformasjonen. Her oppgir man også hvilket XSLT-dokument som skal benyttes til selve transformasjonen.

```
<content-viewer name="listOfAllStudentsViewer">
    <view-mapping>
        <view-name>studentsView</view-name>
        <content-path>.</content-path>
        <transform-by>studentListTransformer.xslt</transform-by>
    </view-mapping>
</content-viewer>
```

Eksempel 2-2: Definisjonen av en content-viewer

Perspective
Et perspective gjør det mulig å gruppere relaterte views og content-viewers. Det er mulig å ha flere perspectives i en og samme presentasjon, som hver kan inneholde ulike kombinasjoner av views og content-viewers. Et perspective tilsvarer en bestemt HTML-side som kan vises i presentasjonen.

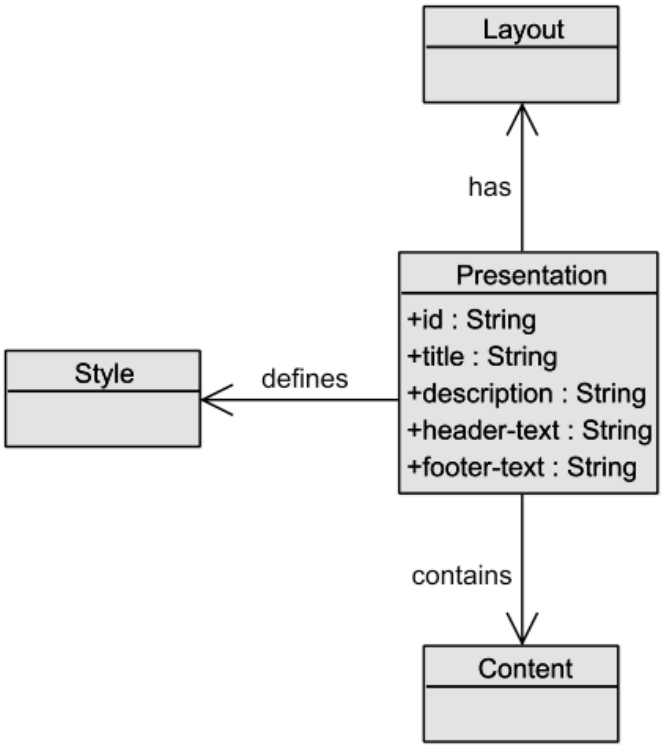
I eksempel 2-3 er det definert et perspektiv ved navn `studentList`. Den inneholder tre elementer. To av disse er views og det siste er en content-viewer. Hovedmålet med dette er å definere gruppering. Implementasjonen av mønsterspesifikasjonen i DPG 1.0 fraviker imidlertid betydelig fra hvordan dette er definert i (Espelid 2004).

```
<perspective name="studentList">
    <view-name>infoView</view-name>
    <view-name>contactView</view-name>
    <content-viewer-name> listOfAllStudentsViewer</content-viewer-name>
</perspective>
```

Eksempel 2-3: Definisjonen av et perspective

2.3.2 Presentasjonsspesifikasjonen
Figur 2-4 viser hvilke komponenter som inngår i en presentasjonsspesifikasjon. Som man ser er det tre hovedkomponenter:

- Content
- Layout
- Style



Figur 2-4: Komponentene som inngår i en presentasjonsspesifikasjon

Content
I denne komponenten oppgir man hvilke innholdsfiler (XML-dokumenter) som inngår i presentasjonen. Eksempel 2-4 viser et eksempel på dette:

```
<content>
    <file>exams.xml</file>
    <file>persons.xml</file>
    <file>curriculum.xml</file>
</content>
```

Eksempel 2-4: Definisjonen av content

Alle filer som inngår i presentasjonen må listes under taggen `<content>`. Dersom en innholdsfil ikke er med i listen, vil den heller ikke kunne benyttes i et view eller view-handler, og innholdet vil dermed ikke bli tilgjengelig for resten av systemet.

Layout
Layout definerer en kobling mellom et perspektiv og en layout-fil, i dette tilfelle en JSPX-side. Eksempel 2-5 viser hvordan dette defineres i spesifikasjonen ved hjelp av taggen `<layout-mapping>`:

```
<layout-mapping>
    <perspective-name>studentsPerspective</perspective-name>
    <layout-template>studentLayout.jspx</layout-template>
</layout-mapping>
```

Eksempel 2-5: Konfigurasjon av layout-mapping

Style
Hver presentasjon har muligheten til å definere et stilark (CSS). Dette gjør det mulig for presentasjoner å ha forskjellig utseende, selv om de benytter samme mønsterdefinisjon. viser hvordan man setter stilark. Eksempel 2-6 viser hvordan dette defineres.

```
<style>
    <style-sheet>students.css</style-sheet>
</style>
```

Eksempel 2-6: Konfigurasjon av style

2.4 DPG 1.0

2.4.1 Historikk

I begynnelsen av JAFU-prosjektet var det ingen verktøy på plass for å kunne holde fjernundervisningskurs. En sentral del av prosjektet ble – og er fremdeles - dermed å utvikle og tilpasse en samling av relaterte applikasjoner for å støtte gjennomføringen av slike kurs. Disse verktøyene ble gradvis utviklet og videreutviklet gjennom ulike master-, hovedfags- og doktorgradsoppgaver, prosjekter i fagene INF112 - Systemkonstruksjon og INF219 – Praktisk prosjekt i programmering og betalt arbeid. Ved oppstarten av denne oppgaven hører følgende verktøyene til JAFU-prosjektet (Se tabell 2-1).

System	Beskrivelse	Siste utførte arbeid/bidrag
Dynamic Presentation Generator (DPG)	Innholdshåndteringssystem (CMS)	Elektronisk kompendium-prosjektet (finansiert av SEVU) 2006-2007
Webucator 2.0	System for administrasjon av brukere	Masteroppgaven ”Webucator 2.0 – a courseadministration system” av Preben Solheim, fullført 2006
SYSTEKON	System for kontrollspørsmål og interaktive prøver	INF112-prosjekt vår 2007
MVNForum	Forumløsning	(ikke utviklet ved JAFU)

Tabell 2-1: Oversikt over verktøy tilhørende JAFU-prosjektet

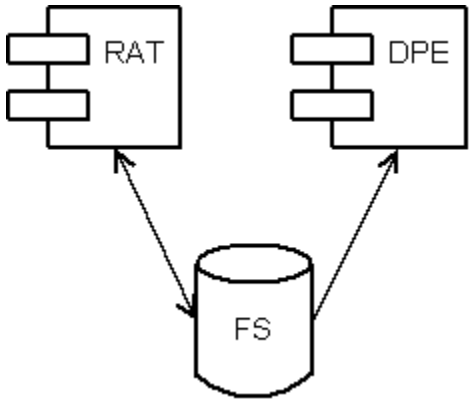
DPG 1.0 ble skrevet av Yngve Espelid i 2004 som en del av masteroppgaven hans, og ferdigstilt i 2004. Dette arbeidet er beskrevet i (Espelid 2004). Espelid baserte noe av arbeidet sitt på resultatene til flere andre hovedfagsoppgaver og doktorgradsavhandlinger skrevet ved instituttet, spesielt (Cruickshanks 2004) og (C. M. Berg 2004), som omhandlet lignende prosjekter kalt henholdsvis JPGen og PMM. Mer informasjon om disse prosjektene finnes i kapittel 2 av (Espelid 2004), som inneholder en sammenligning av de to prosjektene, og i de respektive oppgavene til Cruickshanks og Berg. All koden til Espelids DPG var imidlertid skrevet fra bunnen av, det var kun ideene fra de foregående systemene som ble brukt og videreutviklet i hans oppgave.

DPG 1.0 har gjennomgått en del vedlikeholdsarbeid etter Espelids oppgave (Espelid 2004). Sommeren 2005 foregikk det arbeid med utarbeidelse av systemdokumentasjon, retting av noen programfeil, refaktorering av deler av koden, og det ble påbegynt reimplementasjon av presentasjonslaget i MVC-rammeverket Spring MVC (The Spring Framework, SpringSource). Dette arbeidet ble imidlertid ikke videreført, da det senere det året ble bestemt at det skulle foretas en fullstendig reimplementasjon av DPG: DPG 2.0.

En modifisert versjon av DPG-systemet er også blitt tatt i bruk i andre sammenhenger enn JAFU-prosjektet. Senter for etter- og videreutdanning (SEVU), også ved Universitetet i Bergen, fattet interesse med verktøyet, og ville undersøke om det var mulig å bruke det til fjernundervisningskurs tilknyttet juridisk fakultet. Videreutviklingen av DPG-en startet for fullt i 2007 og ble ferdigstilt til kursstart høsten samme året. Kurset som ble avholdt ved høsten ble vellykket, og våren 2008 ble det avholdt to ytterligere juridiske nettkurs. Dermed fikk man ytterligere demonstrert nytteverdien med presentasjonsmønstre og generiske systemer, og både sterke og svake sider med den nåværende DPG-implementasjonen ble avdekket.

2.4.2 Overordnet arkitektur

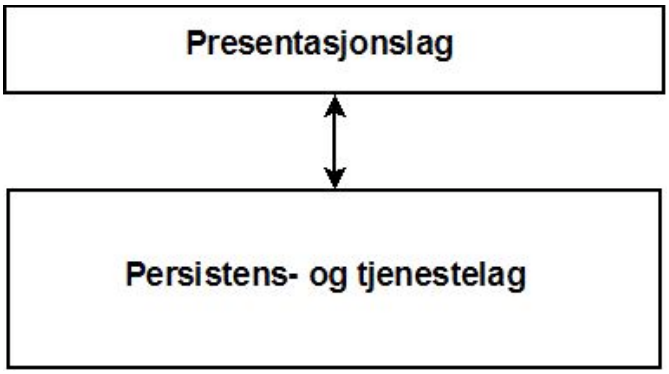
DPG 1.0 er delt inn i to hoveddeler (se Figur 2-5): Dynamic Presentation Engine (DPE) og Repository Administration Tool (RAT). Begge disse bruker et sentralt repositorium som datakilde. Dette repositoriet er en katalog på filsystemet med en forhåndsdefinert katalogstruktur. På grunn av problemer med kompilering av JSP-er og Tomcat og nedlasting av filer for brukere, må denne katalogen ligge inne i applikasjonens katalog i ”webapps”-katalogen til Tomcat.



Figur 2-5: Overordnet arkitektur for DPG 1.0

DPE er komponenten som er ansvarlig for å rendere innhold og vise det til vanlige brukere. Det er altså ikke mulig å manipulere innhold ved å gå gjennom DPE-en i DPG 1.0. RAT er komponenten som er ansvarlig for å tilby redigering av innhold, presentasjoner og presentasjonsmønstre, samt opprettelse av nye presentasjoner og presentasjonsmønstre. RAT kan også håndtere brukere (legge til, fjerne, gi rettigheter til), men denne funksjonaliteten er blitt lite brukt.

DPG 1.0 er delt opp i to lag (se Figur 2-6): Et presentasjonslag og et persistens- og tjenestelag. Presentasjonslaget har som ansvarsområde å vise brukeren riktige JSP-sider, å holde orden på hvor i presentasjonen brukeren befinner seg (tilstand), og å delegere hoveddelen av arbeidet til persistens- og tjenestelaget. Dette laget inneholder all forretnings- og persistenslogikken. Lagets ansvarsområder inkluderer blant annet å utføre create-, retrieve, -update,- og delete-operasjoner (CRUD) på presentasjoner, patterns og brukere, å parse XML-data for å bygge objekter og tilstandshåndtering.



Figur 2-6: Overordnet lagdeling i DPG 1.0

Presentasjonslaget til DPG-en bruker designmønsteret Front Controller (Fowler, Patterns of Enterprise Application Architecture 2003), både i DPE- og RAT-delen av applikasjonen. Begge disse ble implementert med en servlet som delegerte arbeid videre til resten av applikasjonen. Espelid benyttet ikke noe MVC-rammeverk til implementasjonen av presentasjonslaget. Presentasjonslaget til DPE er imidlertid nå

implementert i Spring MVC (The Spring Framework), som en del av en refaktorering foretatt etter Espelid var ferdig med oppgaven.

I persistens- og tjenestelaget til DPG 1.0 er designmønsteret Singleton (Gamma et al. 1994) mye brukt. Ved oppstart av applikasjonen lastes nemlig alle presentasjonsmønstre og alle presentasjoner inn i minnet, og en slik form for ressursbruk vil man naturligvis at kun skal skje en gang. Dette er i bunn og grunn en primitiv erstatning for mellomlagring, og er sannsynligvis gjort for å forbedre ytelsen på applikasjonen. Det er ikke brukt noen form for rammeverk i denne delen heller, verken til tjeneste- eller persistensdelen. Selve persisteringen av dataene skjer i form av flate XML-filer. Ingen form for transaksjonshåndtering er implementert (for eksempel i form av låsing).

2.4.3 Brukere og autentisering

DPG 1.0 opererer med tre grupper av brukere:

- `Readers`, som har tilgang til å lese informasjon
- `Publishers`, som har tilgang til å endre informasjon
- `Administrators`, som har tilgang til å endre struktur på informasjon (mønster), i tillegg til å administrere de andre brukerne og opprette nye presentasjoner og mønstre.

Tilgang for `readers` og `publishers` er på en per-presentasjon-basis. Det vil si at man blir definert som `reader` eller `publisher` i en presentasjon, og dermed ikke har tilgang til å vise de andre presentasjonene som finnes i systemet, med mindre man er definert som `reader` eller `publisher` i dem. `Administrators` har tilgang til å administrere alle presentasjoner som finnes i systemet. En `publisher` har alle rettigheter som en `reader` har i en gitt presentasjon, og `administrators` har alle rettigheter som en `publisher` har.

Autentiseringen foregår via Tomcat, satt opp med et såkalt JDBC Realm. (Apache Software Foundation 1999-2006) Denne er støtet opp under av en PostgreSQL-database som inneholder brukernavn og passord. Denne mekanismen sørger imidlertid bare for å finne ut om en bruker har tilgang til systemet i det hele tatt. Informasjonen om hvilken brukergruppe en bruker tilhører på en gitt presentasjon defineres på presentasjonsbasis i to filer som heter henholdsvis `READERS.XML` og `PUBLISHERS.XML`, og ligger i roten til katalogen til hver presentasjon. Disse har en enkel XML-struktur som inneholder brukernavnene til dem som har rettigheter som henholdsvis `reader` og `publisher`. Disse filene har ingen synkronisering med databasen, og må derfor oppdateres manuelt.

3 Analyse av DPG 1.0

Som tidligere nevnt har den nåværende implementasjonen av DPG-en blitt brukt i fjernundervisningssammenheng med stor suksess i flere semestre, men det har i løpet av denne tiden blitt klart at det finnes en rekke feil og mangler med systemet. Disse kommer fra erfaringer gjort av dem som har vært ansvarlige for fjernundervisningskursene, og andre studenter som har i en eller annen sammenheng har vært borti systemet. Leser man masteroppgaver og arbeidsrapporter som omhandler DPG-en, er det stort sett de samme problemene som nevnes.

Kandidaten har selv vært kursansvarlig et semester og fått føle problemene med DPG-systemet, spesielt med tanke på publisering av informasjon, på kroppen. I tillegg har undertegnede også vært med på å videreutvikle en versjon av DPG-en for bruk ved juridisk fakultet (Berg, *Elektronisk Kompendium*, 2006), der det var behov for en helt ny presentasjonsmønster- og presentasjonsspesifikasjon.

Dette kapittelet bygger derfor ikke bare på erfaringer fra andre, men også egne erfaringer under bruken av dette systemet. Etter å ha hatt rollen som både kursansvarlig og utvikler tilknyttet dette systemet, har kandidaten fått innblikk i hvordan det er å drifte systemet fra dag til dag og hvilke utfordringer som det byr på, i tillegg til oversikt over hvordan den underliggende arkitekturen og kodebasen ser ut.

3.1 Problemområder

I dette kapittelet vil de viktigste problemområdene bli belyst, og både positive og negative sider vil bli analysert. Analysen som gjøres i dette kapittelet vil danne grunnlaget for det nye systemet, og vil forhåpentligvis hindre at det nye systemet gjør de samme feilene.

3.1.1 Presentasjonsmotoren (DPE)

Sett med den vanlige brukers øyne, dvs. brukere som har rollen som `readers` (i fjernundervisningssammenheng er dette studenter), fungerer systemet greit. De tilbakemeldinger vi har fått fra studentene som har tatt fjernundervisningskurs de senere semestrene har stort sett vært gode. Hovedårsaken til dette er nok at den vanlige bruker ikke har flere rettigheter enn å lese av informasjon på nettsidene, og derfor aldri er i kontakt med administrasjonsverktøyet som brukes for å endre på innholdet.

3.1.2 Robusthet

Likevel er det en del problemer som har skapt en del nedetid og ekstraarbeid for dem som administrerer systemet. Disse problemene er en direkte følge av at systemet ikke er godt nok implementert og tilstrekkelig testet før det ble satt i drift. Systemet har lav robusthet og kaster regelmessig såkalte `NullPointerException's`, noe som fører til at brukeren får en kryptisk feilmelding i nettleseren. En del tid har blitt brukt for å finne ut hva som forårsaker disse feilene, uten at man har avdekket hva.

3.1.3 Systemytelse

Det er også avdekket problemer knyttet til bruken av systemminne. I forbindelse med videreutviklingen av DPG-en i det såkalte SEVU-prosjektet (Berg, *Elektronisk Kompendium*, 2006) ble det kjørt flere instanser av DPG-en samtidig på en og samme server. Man opplevde da at systemene etter noen ukers drift plutselig sluttet å fungere, og en omstart av tjener var nødvendig for å få dem i gang igjen. Etter noen uker til gjentok dette seg, og en ny omstart var nødvendig. Hyppigheten på problemet ble derimot kraftig redusert da man forsøkte å øke mengden med systemminne på server, noe som kan tyde på at det fins minnelekasjer i systemet.

Det er også høyst usikkert på hvor bra systemet skalerer etter hvert som antall brukere øker. Til nå har systemet blitt brukt i fjernundervisningssammenheng, der det som regel er et begrenset antall studenter per kurs (10-20 studenter i snitt). Dersom DPG-en skal benyttes i andre sammenhenger enn nettkurs der antall brukere øker dramatisk, kan problemene med bl.a. minnebruk og lav robusthet trolig bli så store at det ikke lar seg gjennomføre i praksis.

3.1.4 Sikkerhet og brukerhåndtering

Sikkerheten til systemet har lenge blitt kritisert for å være for dårlig. Karianne Berg og Kristian S. Løvik gjennomførte høsten 2006 en nærmere analyse av sikkerhetssituasjonen (Løvik og Berg, *Sikkerhetsproblemer i DPG 1.0*, 2006), der flere grove sikkerhetsbrudd ble avdekket. Bl.a. viste det seg at hvem som helst, ved å modifisere URL-er, kunne få tak i alt fra innholdsfiler, opplastede dokumenter og til og med viktige systemfiler. Lite ble gjort for å utbedre dette problemet, mest fordi det ville kreve omfattende refaktorering.

Brukerhåndteringen for DPG-en er i dag overlatt til et eksternt system, kalt Webucator. Dette systemet brukes for å administrere kurs, kursinstanser og brukere til disse. Listen over hvilke brukere som har rettigheter til hvilke nettkurs blir så manuelt eksportert som XML-filer, og importert inn i DPG-en. Selv om dette fungerer tilstrekkelig, er det en veldig slitsom og tidkrevende prosess. Webucator-systemet som har blitt brukt til nå er preget av å ikke være helt ferdigutviklet, og være vanskelig å installere og vedlikeholde. En ny versjon av Webucator-systemet har lenge vært under utvikling og denne vil bli brukt for brukerhåndtering i det nye DPG-systemet. For en mer grundigere gjennomgang av problemene knyttet til sikkerhet og brukerhåndtering, se (Løvik, 2008).

3.1.5 Kodekvalitet

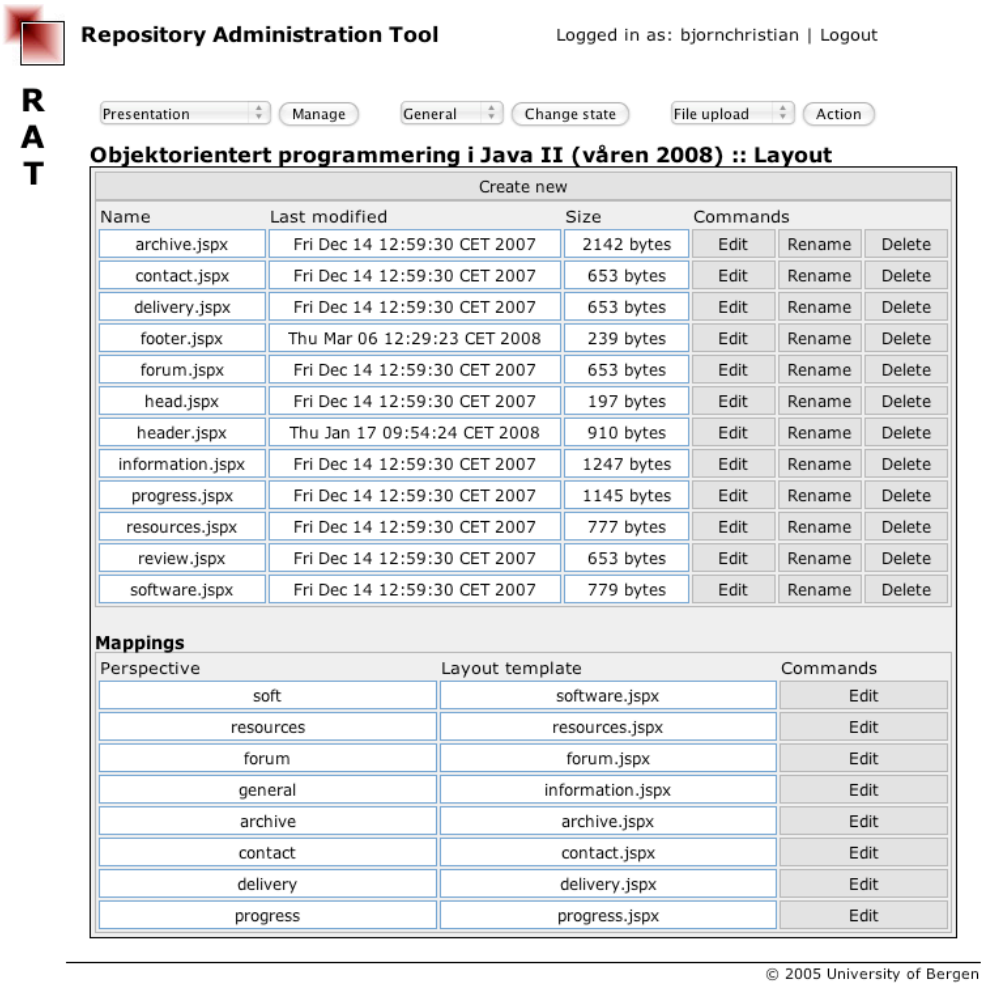
Kildekoden til DPG 1.0 innholder en rekke problemer som gjør den vanskelig å vedlikeholde og videreutvikle. Systemet bygger opprinnelig på en ren Servlet/JSP-arkitektur, uten bruk av tredjeparts rammeverk som for eksempel Struts/Spring (The Spring Framework, SpringSource). Årsaken til at slike rammeverk ikke ble brukt er usikkert, men en mulig årsak kan være fordi disse rammeverkene ikke ble gitt som del av undervisningen ved instituttet. Spring-rammeverket var også ganske ungt på den tiden, og hadde en mye svakere posisjon i markedet enn det har i dag.

Systemet lider også av en sterk mangel på enhetstester, slik at det er vanskelig å videreutvikle systemet uten at eksisterende kode slutter å fungere. Mange steder er det vanskelig for andre utviklere å sette seg inn i koden, da det også er mangel på gode kommentarer. Det fins heldigvis noe skriftlig dokumentasjon om systemet i en del masteroppgaver og prosjektoppgaver, noe som hjelper litt på forståelsen av systemet, deriblant (Mughal 2003), (Rossini og Liberati 2005) og (Espelid 2004).

Den nåværende versjonen av DPG-en har blitt forsøkt refaktorert til å benytte seg av Spring-rammeverket (Rossini og Liberati 2005). Dette ble aldri helt gjennomført, da utviklerne støtte på så mange problemer at det i stedet ble bestemt at en total omskriving av DPG-en var nødvendig.

3.1.6 Innholdshåndtering og administrasjon

De største problemene med DPG-en i dag er knyttet til innholdshåndtering og administrasjon av presentasjonene. Denne funksjonaliteten dekkes i dag av ett verktøy med navnet Repository Administration Tool (RAT), beskrevet i (Espelid 2004). Dette verktøyet er kun tilgjengelig for dem som har rollen som administrator, og har sin egen innloggingsside. Figur 3.1 gir et innblikk i hvordan grensesnittet i verktøyet er lagt opp.



Figur 3-1: Eksempel på hvordan grensesnittet i administrasjonsverktøyet i DPG 1.0 er lagt opp

Grensesnittet til dette verktøyet er høyst funksjonelt, men lite attraktivt og det kan være forvirrende å navigere mellom de forskjellige undersidene. Helst øverst finner man tre såkalte dropdown-bokser som lar bruker gå til andre steder i verktøyet. Dette er en nokså utradisjonell navigasjonsløsning og kan være forvirrende for bruker. Under menyen finner man som regel en liste eller skjema, avhengig av hva som er valgt.

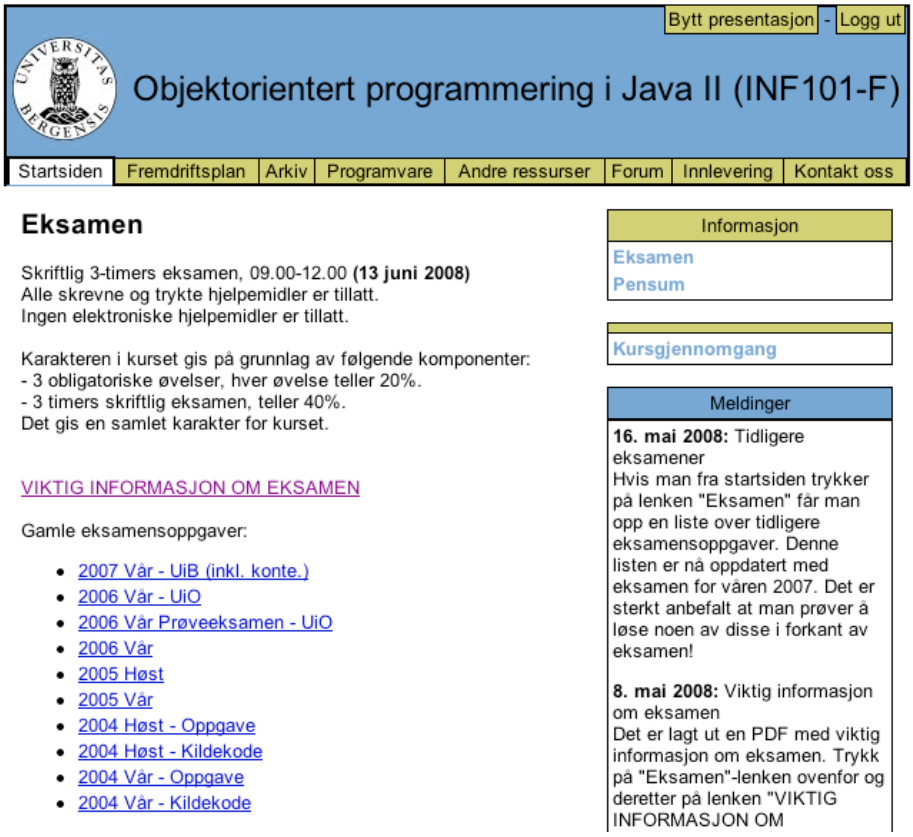
Selve navigasjonen skjer ved at bruker velger hvilken hovedaktivitet som skal utføres i dropdown-boksen lengst til venstre. I figur 3.1 har bruker valgt å redigere en bestemt presentasjon. Dette valget får konsekvenser på hvilke valg som er tilgjengelig i den midterste dropdown-boksen. I figuren har bruker valgt kategorien ”layout” fra denne menyen, men dessverre reflekterer ikke boksen hvilken kategori som er valgt. I stedet vises ”General”, den første valgmuligheten i dropdown-boksen.

Den siste dropdown-boksen lengst til høyre, er derimot ikke avhengig av det forrige menyvalget. Den inneholder en liste over aktiviteter (”actions”) som brukeren kan utføre for det valgte presentasjonen eller mønsteret, deriblant opplasting av ressurser eller håndtering av brukere.

Hele systemet fokuserer mye på filer fremfor konsepter, og man bør ha kjennskap til hvordan presentasjonsmønstrene er bygget opp for å kunne forstå hvilke filer som henger sammen på hvilken måte, og hvilke konsekvenser eventuelle endringer i de forskjellige filene fører til. Det er heller ikke noen måte å angre de handlinger som utføres, slik at den som benytter seg av dette verktøyet er nødt til å trå varsomt.

Et av problemene er at verktøyet egentlig dekker to brukerroller, `administrator` (en som skal endre på systemfiler, behandle brukere, etc.) og `publisher` (en som skal endre på innholdet i presentasjonene). Men brukergrensesnittet legger opp til manuell redigering av innholdsfiler, filer som er i XML-format. Mangelen på en såkalt WYSIWYG-editor (WYSIWYG Wikipedia) fører til at den som ønsker å endre på innholdet til en presentasjon, må ha god kjennskap til XML, XSLT og JSP-teknologiene. Det har vært en forutsetning at dem som er kursansvarlige har denne kunnskapen, i tillegg til å kunne nok om DPG-en og dens problemer, noe som gjør at det er vanskelig å finne kvalifisert personell til denne rollen.

Dette kan lett illustreres ved et eksempel. I figur 3-2 ser man startsiden på presentasjonen ”Objektorientert Programmering i Java II” fra nettkurset våren 2008. Den inneholder i utgangspunktet to `views`, et `view` med eksamensinformasjon og ett med de siste meldingene.

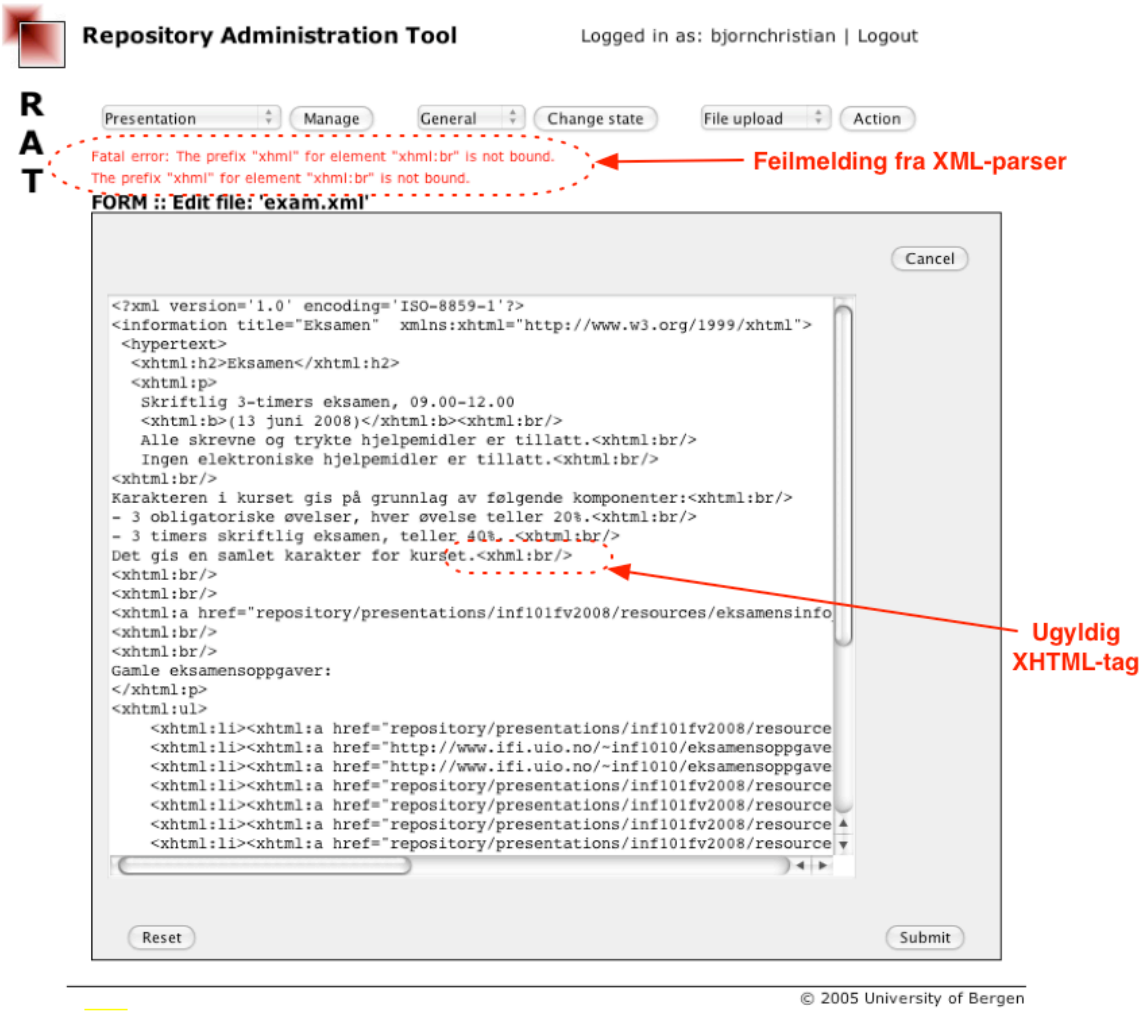


Figur 3-2: Rendring av eksamensinformasjonssiden i nettkurset INF101-F

Endre innhold

La oss nå si at kursansvarlig ønsker å endre på informasjon relatert til eksamen. Kursansvarlig logger seg da inn på administrasjonsverktøyet via dens på innloggingsside, velger listen over innholdsfiler og velger å redigere filen `exam.xml`.

Som figur 3-3 viser, får kursansvarlig opp et enkelt HTML-skjema med et stort tekstfelt med innholdet i filen. Selv om tekstinnholdet på eksamenssiden er nokså lite (en overskrift, noen paragrafer og en liste med lenker), blir XML-koden som tilsvarer dette nokså kompleks. For en utvikler som er godt vant med å lese XML-filer er dette ikke noe problem, men for den vanlige bruker virker dette utrolig skremmende. At man i dette tilfelle har XHTML nøstet inni en XML-fil, med namespace foran alle XHTML-elementer, gjør ikke situasjonen bedre.



Figur 3-3: Redigering av eksamensinformasjon ved hjelp av administrasjonsverktøyet RAT

Dersom man nå ønsker å legge til en lenke til en ny eksamensoppgave som er lastet opp på tjeneren, må man selv sette opp XML-koden som skal til, og selv sørge for at URL-en som kan skriver inn fungerer. Dersom brukeren er uheldig og skriver gal syntaks, vil systemet heldigvis hindre at ugyldig XML-kode blir lagret (noe som kan føre til problemer når siden skal rendres at motoren). Men som man ser i figur 3-3 er ikke feilmeldingene alltid så hjelpsomme. Det er tydelig at disse feilmeldingene blir generert av en XML-parser, og den snakker et språk som kun uviklere forstår. I dette tilfellet har bruker vært uaktsom og skrevet `</xhml:br>` fremfor det korrekte `</xhtml:br>` etter en paragraf.

Verktøyet har i utgangspunktet også muligheten til å redigere presentasjonsmønstre i tillegg til presentasjoner, men denne delen av verktøyet fungerer ikke skikkelig og er dermed svært begrenset. Slik det er nå, tilbyr verktøyet å opprette selve presentasjonsmønsterspesifikasjonen, deretter må bruker selv skrive og laste opp alle filene som kreves for et mønster (bl.a. layoutfiler). Dette har ført til at man har måtte utvikle nye mønstre ved hjelp av andre verktøy, for eksempel en XML-editor eller Integrated Development Enviroment (IDE) som har støtte for XML og XSLT.

En meget lei begrensning er at det ikke er noen enkel måte å opprette nye presentasjoner basert på eksisterende mønstre på. Dette er noe som absolutt burde være tilstede i verktøyet. Systemet kan kun opprette presentasjonsspesifikasjonen, uten innholds-, JSP- og stilark-filer. I stedet har kursansvarlig som regel valgt å sette opp nye presentasjoner manuelt i filsystemet. Det innebærer vanligvis at man kopierer presentasjonsmappen fra en eksisterende presentasjon, og deretter endrer bl.a. identifikatorer, navn, innhold og brukere for den nye presentasjonen. Dette er både tungvindt og det er lett å gjøre alvorlige feil som resulterer i at hele systemet blir ustabilt.

3.1.7 Presentasjonsmønstre

Presentasjonsmønsteret som dagens DPG bygger på er en klar demonstrasjon av at konseptet rundt presentasjonsmønstre ikke bare er mulig i teorien, men at det også lar seg realisere i praksis. Det er mulig å utvikle nye presentasjonsmønsterspesifikasjoner og opprette presentasjoner av disse, og mønsteret er fleksibelt nok til å takle en god del av den funksjonaliteten som trengs for å avholde nettkurs, samt en del av de andre mønsterforslag som er dukket opp i ettertid (se blant annet (Lervik Larsen 2006)) problemer som hindrer utviklingen av nye mønstre og presentasjoner.

Et av problemene er at det ofte er uklart hvordan de ulike elementene i en mønsterspesifikasjon, dvs. `view`, `content-viewer` og `perspective`, skal brukes i ulike kontekster. Det er med andre ord uklare grenser mellom disse elementene, noe som forvirrer dem som skal utvikle nye mønsterspesifikasjoner. Eksempel 3-1 gir et innblikk i hvor forvirrende et presentasjonsmønsterdokument kan være.

Eksempelet er en forenklet utgave av det presentasjonsmønsteret `coursePattern`, som har vært brukt til fjernundervisningen i dagens DPG i flere semestre. Det første man oppdager når man prøver å sette seg inn i dette eksempelet, er at implementasjonen av presentasjonsmønsterspesifikasjonen avviker noe fra det som er beskrevet i de tidligere masteroppgavene om temaet (spesielt (Espelid 2004)).

Det som skaper størst forvirring er hvordan `perspectives` (linje 34-48) bygges opp med `views` og `content-viewers`. Man ser at et tidligere definert `view` ved navn `allWeeklyAssignments` (linje 9-12) blir brukt i perspektivet `archive` (linje 35-46), men attributten `content-path` blir definert på nytt (linje 37), nå med en ny verdi, som i tillegg ikke er lovlig XPath-syntaks (XPath, W3C). Deretter blir content-vieweren `assignmentViewer` referert til (linje 39), også den med ny `content-path`-attributt (linje 40). I tillegg defineres det også hvilken transformasjonsfil som skal rendre resultatet til HTML (linje 41), til tross for at en transformasjonsfil allerede er definert tidligere (linje 24). Det er heller ingenting som forteller oss hvordan et vanlig `view` skal transformeres og rendres.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<presentation-pattern>
    <title>Course Pattern</title>
    <description>The Course Pattern is used for e-learing courses.</description>
    <content-structure>
        <schema>main.xsd</schema>
    </content-structure>
    <views>
        <view name="allWeeklyAssignments">
            <content-path>//assignment[@type='week']</content-path>
            <identified-by>title/text()</identified-by>
        </view>
        <view name="allCompulsoryAssignments">
            <content-path>//assignment[@type='compulsory']</content-path>
            <identified-by>title/text()</identified-by>
        </view>
        <!-- Øvrige views er utelatt for å forenkle eksempelet -->
      </views>
    <content-viewers>
        <content-viewer name="assignmentViewer">
            <view-mapping>
                <view-name>allWeeklyAssignments</view-name>
                <content-path>.</content-path>
                <transform-by>transformAssignment.xslt</transform-by>
            </view-mapping>
            <view-mapping>
                <view-name>allCompulsoryAssignments</view-name>
                <content-path>.</content-path>
                <transform-by>transformAssignment.xslt</transform-by>
            </view-mapping>
        </content-viewer>
        <!-- Øvrige content-viewers er utelatt for å forenkle eksempelet -->
    </content-viewers>
    <perspectives>
        <perspective name="archive" text="Arkiv">
            <view name="allWeeklyAssignments">
                <content-path>-</content-path>
            </view>
            <content-viewer name="assignmentViewer">
                <content-path>//hypertext[@label='archive']</content-path>
                <transform-by>hypertext.xslt</transform-by>
            </content-viewer>
            <view name="allCompulsoryAssignments">
                <content-path>-</content-path>
            </view>
        </perspective>
        <!-- Øvrige perspectives er utelatt for å forenkle eksempelet -->
    </perspectives>
</presentation-pattern>
```

Eksempel 3-1: Et eksempel på hvordan presentasjonsmønstre spesifiseres i DPG 1.0

Slike forvirrende detaljer skaper frustrasjon og forvirring for den som utvikler nye mønsterspesifikasjoner. Og det hjelper heller ikke at det lille som fins av informasjon om utvikling av presentasjonsmønstre for systemet, både er mangelfullt og er ikke i samsvar

med det som faktisk er implementert. Man risikerer dermed at spesifikasjonen inneholder feil eller mangler, som fører til at presentasjonsmotoren tolker den på gal måte, som igjen kan føre til systemsvikt.

Siden hver utvikler tolker presentasjonsmønstrene på sin egen måte, blir vedlikehold av eksisterende mønstre en stor utfordring. Og uten et felles forståelsesgrunnlag av hvordan presentasjonsmønstre skal spesifiseres på en korrekt måte, er det lite sannsynlig at det blir utarbeidet flere mønstre utover dem vi har i dag.

3.2 Utfordringer for det nye systemet

Som denne analysen har påpekt, er det flere alvorlige problemer med den eksisterende implementasjonen av DPG-en. En refaktorering av denne ble som nevnt forsøkt, men gitt opp og en total omskriving av systemet ble foreslått i stedet. Dette ser ut til å være den mest fornuftigste løsningen, og selv om det vil ta mye tid og krefter vil det høyst sannsynlig være verdt det på lengre sikt.

Dersom man skulle fortsette å videreutvikle systemet basert på den nåværende kodebasen, ville det trolig resultere i økte vedlikeholdskostnader og økende frustrasjon, ikke bare hos utviklerne, men også hos brukerne, som selvsagt blir påvirket av systemfeil og nedetid.

Det nye systemet står ovenfor flere store utfordringer som må overvinnes. Uklarheter knyttet til presentasjonsmønsterspesifikasjonen, lite brukervennlig innholdshåndtering og en rekke problemer med kildekode og systemytelse, er alle sentrale problemer som må løses. I det neste kapittelet blir målsetningene for det nye systemet lagt frem, og løsninger på disse problemene utarbeidet.

4 Introduksjon til det nye DPG-systemet

I dette kapittelet tas det utgangspunkt i de problemer og den lærdom som ble avdekket av analysen av det gamle systemet i det forrige kapittelet. De viktigste målsetningene for et nye systemet blir gjennomgått, før den overordnede strukturen til DPG 2.0 blir presentert. Dette kapittelet vil også introdusere et eksempel på en presentasjonsmønsterspesifikasjon som vil bli brukt i kommende kapitler for lettere å illustrere hvordan det nye systemet fungerer.

4.1 Målsetninger for det nye systemet

Følgende målsetninger, inndelt i ulike kategorier, er utarbeidet for det nye systemet:

4.1.1 Forbedret presentasjonsmønster

Det er svært viktig at det nye presentasjonsmønsteret er fleksibelt nok til å kunne danne grunnlag for en rekke forskjellige presentasjonsmønster- og presentasjonsspesifikasjoner. Utviklere av mønsterspesifikasjoner må få større frihet i forhold til hvordan nettsidene kan utformes, uten at den friheten blir for kostbar i form av for mange konfigurasjonsfiler og vanskelig syntaks.

Samtidig må syntaksen for disse spesifikasjonene være mer tydelig og logisk enn det dagens presentasjonsmønster er. De ulike elementene som er involvert må ha en klart definert funksjon og plassering slik at utviklere ikke misforstår hvordan elementene skal benyttes i ulike kontekster. Slike misforståelser fører til at utvikling av nye mønsterspesifikasjoner tar lengre tid og vedlikehold av eksisterende mønsterspesifikasjoner blir vanskeligere. Denne uklarheten og forvirringen rundt de forskjellige elementene i spesifikasjonene var det største problemet med det nåværende presentasjonsmønsteret, og det er derfor svært viktig at det nye presentasjonsmønsteret ikke viderefører dette problemet.

Det er også viktig at det nye presentasjonsmønsteret benytter de forskjellige teknologiene som er involvert på en bedre måte. For eksempel er det fullt mulig å utføre en god del prosessering av informasjon i XSLT fremfor i såkalte JSP-taglibs (Java Server Pages, SUN) eller i ren kode. XSLT har god innebygget støtte for både sortering og filtrering av elementer, har en klar og tydelig syntaks, og er kjent teknologi for de fleste webutviklere. Ved å la XSLT få en mer sentral rolle vil det trolig være lettere å utvikle nye presentasjonsmønster- og presentasjonsspesifikasjoner.

4.1.2 Enklere administrasjon av presentasjoner

Som avdekket i analysen av det eksisterende systemet er et av de største problemene knyttet til måten presentasjonsinnhold håndteres på. Kun personer med gode kunnskaper til systemet og teknologiene som benyttes (blant annet XML, XSLT og JSP) er kvalifisert til å endre på innholdet i en presentasjon. En mindre kyndig bruker kan lett være uheldig og endre en viktig systemfil på en slik måte at systemet blir ustabilt eller i verste fall slutter å fungere helt.

For å gjøre det enklere å redigere innholdet, bør alle innholdsfiler redigeres ved hjelp av vanlige HTML-skjema. Fremfor å gi bruker full tilgang til å redigere innholdsfilen direkte (og dermed eksponere innholdsfilens struktur), bør innholdsfilen først bli lest av systemet som så bygger opp et generisk HTML-skjema som presenterer den aktuelle innholdsfilen. Dette gjør at bruker kan fokusere fult og helt på selve innholdet, uten å måtte tenke på innholdsfilens interne struktur. Ansvaret for at innholdsfilene har riktig syntaks flyttes da fra bruker til systemet selv, noe som garanterer at innholdsfilen alltid har en korrekt struktur.

I tillegg bør det tas i bruk en såkalt WYSIWYG-editor (”What You See Is What You Get” (WYSIWYG Wikipedia)) i de felt der man ønsker litt rikere tekst enn det som er vanlig i et ordinært HTML-skjema. Dette kan for eksempel være tekstformatering som fet skrift, kursiv, lister, lenker og så videre. Altså de vanligste formateringsmulighetene som de fleste brukere er vandt med fra andre teksteditorer som for eksempel MS Word eller OpenOffice Writer. Dette er mye brukt i andre innholdshåndteringssystemer på nettet, og regnes som standard i de fleste moderne blogger og forumer.

4.1.3 Bedre sikkerhet

Det er ikke til å legge skjul på at sikkerheten i det eksisterende systemet ikke er god nok. Det er alt for lett for uvedkommende å få tak i viktige systemfiler eller upublisert innhold (for eksempel svarene på neste ukes oppgaver) ved å tukle med URL-er i nettleseren. Brukere som for eksempel offisielt kun hadde tilgang til presentasjon A og ikke til presentasjon B kunne likevel få tilgang til alle ressurser og innholdsfiler i begge presentasjonene. Årsaken til dette var at området der alle ressursene lå ikke var beskyttet på samme måte som resten av systemet.

For å tette disse sikkerhetshullene vil man i den neste versjonen av systemet ta i bruk et mer omfattende rammeverk for sikkerhet, nemlig Acegi Security (Acegi Security). Dette rammeverket passer veldig godt sammen med Spring Framework (The Spring Framework, SpringSource) og har all den funksjonaliteten som trengs for å løse de problemene som er nevnt.

4.1.4 Enklere brukerhåndtering

Brukerhåndtering i det eksisterende systemet er delegert til det eksterne verktøyet Webucator, som beskrevet i (Løvik 2008). Dette verktøyet lar de som er administrativt ansvarlig opprette kurs og legge brukere til disse. For hvert kurs eksporteres det så en brukerliste som må manuelt importeres inn i DPG-systemet.

Selv om dette fungerer forholdsvis greit, er det ønskelig å fjerne det siste manuelle steget fullstendig. Dette kan gjøres ved å la Webucator-verktøyet eksponere en web-service (Web Service, Wikipedia) som til en hver tid har oversikt over brukere og deres rettigheter. Sikkerhetsmodulen i den nye DPG-en kan da konfigureres til å kjøre spørringer direkte mot denne web-servicen, og manuell import av brukerinnstillinger er ikke lenger nødvendig. Da vil endringer som utføres i Webucator-verktøyet ha umiddelbar effekt i DPG-en.

4.1.5 Høyere kodekvalitet ved hjelp av tester

For å unngå at kodebasen blir vanskelig å forstå og videreutvikle, må det nye systemet ha en betydelig høyere kodekvalitet. En god måte å sikre dette på er å sørge for at det utarbeides enhetstester sammen med koden, gjerne ved hjelp av såkalt Test-driven Development (Koskela 2008). Ved å ha et omfattende sett med enhetstester har man et sikkerhetsnett som fanger opp kode som går i stykker under utviklingen. Da blir det også lettere for andre å sette seg inn i koden og videreutvikle kodebasen, siden enhetstestene også fungerer som teknisk dokumentasjon på hvordan koden er ment å fungere.

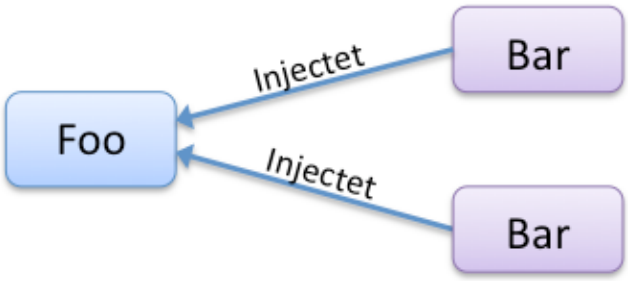
En ofte brukt måleteknikk for å sørge for at en kodebase er tilstrekkelig testet, er å bruke såkalte testdekningsverkøy (Test Coverage, (Koskela 2008)). Det fins flere slike verktøy basert på åpen kildekode som er gratis å ta i bruk, bl.a. Cobertura (Cobertura Coverage Tool) og EMMA (EMMA Java Code Coverage). Disse kan lett integreres enten direkte i IDE-et, eller i en Ant-byggefil. Rapportene som de produserer vil vise hvilke kodelinjer som er testet, og hvor det evt. testing er mangelfullt. Hvor stor testdekning som kreves for at man kan si at et prosjekt er tilstrekkelig testet fins det ikke noe fasitsvar på, da kvaliteten på testene er mer viktig fremfor kvantitet.

Det kan være utfordrende å teste komponenter som har mange avhengigheter til andre komponenter i systemet. For å unngå at enhetstestene blir integrasjonstester, må disse avhengighetene erstattes av såkalte mocks eller stubs (forskjellen mellom disse diskuteres i den velkjente artikkelen (Fowler, Mocks aren't Stubs)). Også her fins det flere gode åpne kildekodebibliotek, bl.a. EasyMock (EasyMock) og jMock (jMock - Lightweight Mock Object Library), som gjør det enkelt å eliminere disse avhengighetene under testing.

4.1.6 Bruk av moderne rammeverk og designteknikker

Siden dagens DPG ble implementert, har det dukket opp en rekke spennende Java-teknologier, og kanskje den viktigste av disse er Spring Framework (The Spring Framework, SpringSource). Dette rammeverket er basert på åpen kildekode og er en samling med svært nyttige hjelpeklasser som tilbyr løsninger på de mest vanligste problemene som Java-utviklere støter på. Mange ser på det som en mulig erstatning for Enterprise JavaBeans (Enterprise JavaBeans, SUN), og det gir en mye større frihet i hvordan man bygger opp større applikasjoner. Selve kjernen i dette rammeverket er det som kalles for en ”Inversion of Control”-modul (SpringSource, IoC Module). Denne modulen er ansvarlig for hele livssyklusen til de viktigste klassene i systemet, og sørger for at alle avhengigheter mellom klassene er oppfylt innen applikasjonen har startet opp og er klar til bruk.

Spring-rammeverket baserer seg mye på en teknikk kalt Dependency Injection. Dette var en videreføring av begrepet Inversion of Control og som Martin Fowler navnga i en artikkel i tidlig 2004 (Fowler, Dependency Injection). Vanligvis må klasser som er avhengig av andre klasser enten selv opprette objekter av disse klassene, eller selv skaffe seg de nødvendige referanser. Dette fører til at man skriver kode som er tett sammenkoblet og blir vanskelig å teste hver for seg.



Figur 4-1: Illustrasjon på hvordan Dependency Injection fungerer i praksis

Dependency injection eliminerer disse problemene ved at en tredjepart, i dette tilfelle Springs IoC-modul, sørger for at et objekt får tildelt referanser til sine avhengigheter under oppstarten av applikasjonen. Man slipper da å ha oppslagskode i klassene, noe som ofte innebærer en god del uhyggelige `try-catch`-blokker. Siden avhengighetene som regel er skjult bak et interface, øker også fleksibiliteten da det blir trivielt å bytte ut en avhengighet med en annen implementasjon. Dette er spesielt nyttig under testing, da alle avhengigheter kan erstattes av *mocks* eller *stubs* som oppfyller disse interfacene.

Som illustrert i figur 4-1, sørger Dependency Injection for at klassen `foo` blir tildelt referanser til sine avhengigheter, `bar`, fremfor å måtte skaffe referanser til disse avhengighetene på egenhånd. Springs IoC-modul sørger for at dette skjer på riktig måte.

```
<beans>
    <!-- Bean "foo" har to avhengigheter, "bar1" og "bar2" -->
    <bean id="foo" class="no.core.Foo">
        <property name="bar1" ref="bar1" />
        <property name="bar2" ref="bar2" />
    </bean>

    <!-- Disse to beansene brukes av "foo" -->
    <bean id="bar1" class="no.core.Bar" />
    <bean id="bar2" class="no.core.Bar" />
</beans>
```

Eksempel 4-1: Beans og avhengigheter defineres vanligvis i XML i Spring-rammeverket

Spring bruker primært XML-filer for å definere beans og avhengigheter mellom dem. I eksempel 4-1 ser man hvordan de tre beansene kan defineres og kobles til hverandre. Her blir avhengighetene til bean `foo` satt med såkalt `setter injection`, den mest brukte formen for dependency injection, der IoC-modulen vil lete etter en passende set-metoder i klassen `Foo` med samme navn som i `name`-attributtet i hver av `property`-taggene.

Dette rammeverket er et av de mest brukte i Java-verdenen, er svært godt dokumentert og det kommer stadig nye oppdateringer. Kandidaten har også selv brukt dette rammeverket

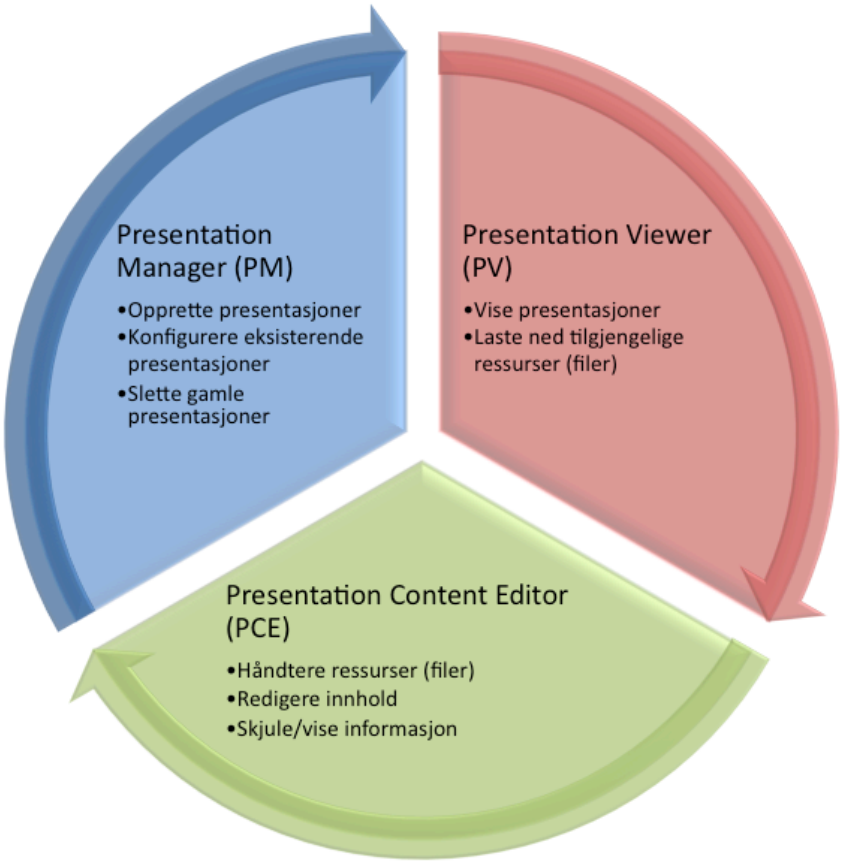
i flere tidligere prosjekter med gode resultater, og mener derfor det vil være ideelt å benytte Spring Framework i det nye systemet.

4.2 Oversikt over delsystemer i DPG 2.0

Det gamle systemet var delt inn i to delsystemer, DPE (Dynamic Presentation Engine) og RAT (Repository Administration Tool). Førstnevnte var ansvarlig for å rendre presentasjoner som HTML-nettsider, altså å gjøre innhold og ressurser tilgjengelige for brukere. Sistnevnte var et administasjonsverktøy som lot dem som var administrativt ansvarlige endre på innhold og innstillinger for presentasjoner og presentasjonsmønstre.

Man hadde i utgangspunktet tre brukertyper: `reader`, `publisher` og `admin`. Mens DPE-en var dedikert `readers`, måtte derimot brukere med rollene `publisher` eller `admin` begge benytte RAT-en for å gjøre endringer. Som det ble påpekt i analysen i forrige kapittel, skapte dette store problemer for `publishers`, da RAT-en ikke er ideell for redigering av selve innholdet i presentasjonene.

I det nye systemet er det ønskelig med en finere oppdeling av disse delsystemene. Denne inndelingen bør reflektere de tre brukerrollene som systemet skal håndtere slik at man ender opp med delsystemer som er skreddersydd for hver av de ulike rollene. Figur 4-2 viser hvilken inndeling som er foreslått for det nye systemet.



Figur 4-2: Oversikt over de ulike delsystemene og deres hovedoppgaver

Hvert delsystem er ment å tjene en bestemt brukerrolle på best mulig måte. Presentation Viewer, markert i det røde området, har som oppgave å gjøre informasjonen i presentasjonene tilgjengelige for `readers`. Presentation Content Viewer, markert i det grønne området, lar `publishers` endre på presentasjonsinnhold. Delsystemet, markert i det blå området, lar `administrators` konfigurere, slette og opprette presentasjoner.

Det nye systemet vil i all hovedsak bestå av tre delsystemer: Presentation Viewer (PV), Presentation Content Editor (PCE) og Presentation Manager (PM). Disse tre delsystemene er ment for henholdsvis brukerrollene `reader`, `publisher` og `admin`. Grovt sett kan man si at PV i det nye tilsvarer DPE i det gamle, mens RAT-en i det gamle er delt i to og blitt til i PCE og PM. Man ender da opp med tre delsystemer som er skreddersydd de tre rollene i systemet, noe som vil gjøre det lettere for de ulike brukerne å utføre sine oppgaver.

4.2.1 Presentation Viewer (PV)

Dette delsystemet er dedikert til `readers` tar seg av rendring av presentasjonene i systemet, på samme måte som det DPE-en gjorde i det gamle. Den tar i mot parametere fra brukeren, finner ut hvilken presentasjonen som skal vises, og konkret hvilke deler av presentasjonen som skal rendres. Presentation Viewer leser så inn den tilhørende presentasjonsspesifikasjonen og presentasjonsmønsterspesifikasjonen, og finner på den måten ut hvordan presentasjonen skal rendres. Før og under rendringsprosessen tas det høyre for sikkerhet og om deler av presentasjonen skal være skjult for bruker. Resultatet av rendringsprosessen er som regel et HTML-dokument, som blir vist i brukeres nettleser som en vanlig nettside.

4.2.2 Presentation Content Editor (PCE)

PCE er delsystemet som er dedikert til `publishers`, og som skal benyttes til å redigere innholdet i en presentasjon. Dette kan se på som en forenklet utgave av RAT-en, med et grensesnitt som er mye mer tilpasset innholdshåndtering. Her er det lagt stor vekt på brukervennlighet, slik at det skal være raskt og enkelt å endre på innholdet i en presentasjon. Det meste av innholdet redigeres ved hjelp av standard HTML-skjema, med WYSIWYG-editorer der det er nødvendig med rikere tekst.

Det er også her man finner funksjonalitet for å håndtere ressurser (filer) tilknyttet presentasjonene. I tillegg til tekst skal det også være mulig å laste opp forskjellige typer filer, som for eksempel dokumenter, bilder, videoer eller andre multimedia. Disse ressursene vil da bli tilgjengelige for readers i Presentation Viewer, enten som en helt enkel nedlastingslenke eller som elementer integrert i nettsiden (for eksempel en videospiller på nettsiden).

Det må også være mulig for en `publisher` å kunne skjule deler av presentasjonen slik at informasjon som enten ikke er relevant eller klar for publisering ikke blir rendret av Presentation Viewer. Editoren må også ha sikkerhetsfiltre som sørger for at kun `publishers` og `administrators` har muligheten til å redigere de presentasjoner de er publiser for.

4.2.3 Presentation Manager (PM)

Det tredje og siste delsystemet er ment for rollen `admin` og er det kraftigste verktøyet DPG-systemet. Her har man muligheten til å administrere alle presentasjoner i hele systemet, og endre på viktige systeminnstillinger.

Den viktigste funksjonen dette verktøyet tilbyr er muligheten til å opprette nye presentasjoner, ikke bare basert på tilgjengelige mønstre, men også basert på eksisterende presentasjoner. I begge tilfellene skal presentasjonene kunne opprettes ved hjelp av en enkel veiviser, og det skal ikke være noe behov til manuell redigering av systemfiler slik det var i det gamle systemet. Det skal også være mulig å slette presentasjoner som det ikke lenger er behov for.

Man må også kunne konfigurere de forskjellige presentasjonene dersom det er nødvendig. For eksempel bør det være mulig å overstyre innstillinger som presentasjonen får gjennom mønsteret, slik at hver presentasjon kan skreddersys til det formålet den skal brukes i. Denne fleksibiliteten gjør presentasjonsmønstrene mer fleksible, og hindrer at det unødvendig utvikles mange nesten like presentasjonsmønstre. Mer detaljer om denne overstyringen blir diskutert i kapittelet om det nye presentasjonsmønsteret senere i oppgaven.

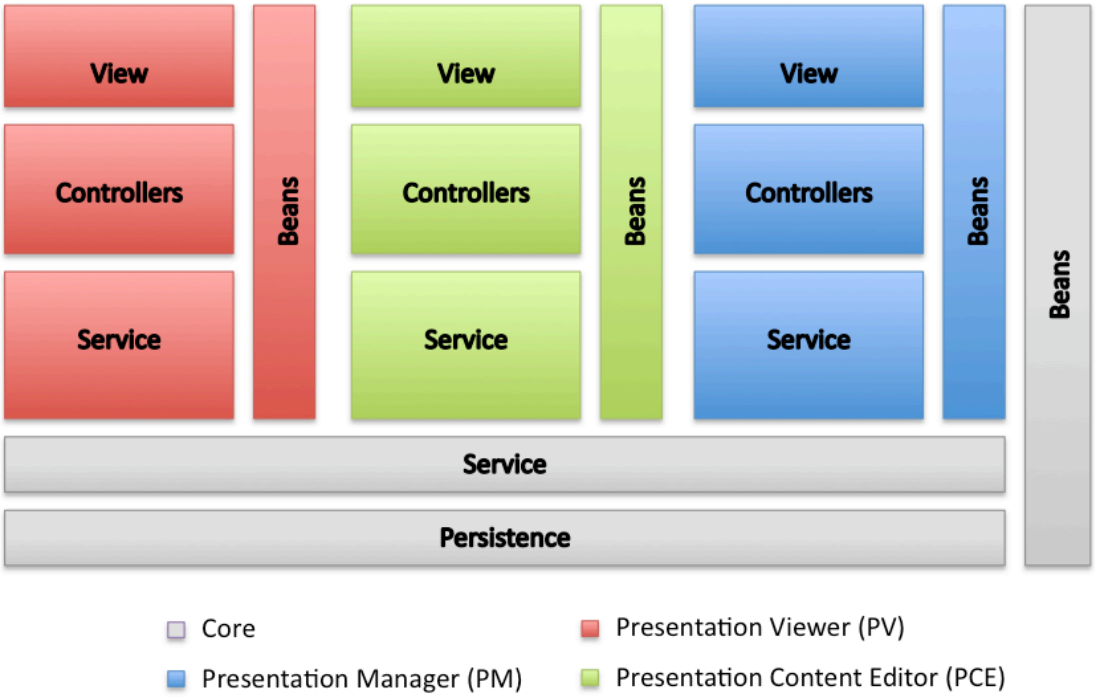
4.2.4 Øvrige delsystemer

I tillegg til disse tre viktige delsystemene er det behov for en påloggingsside og en slags velkomst eller startside, som er det første alle som er brukere av DPG 2.0 møter. Ved å tilby felles pålogging og startside for alle brukerne av systemet, og gjøre det enkelt for brukerne å bytte mellom de ulike delsystemene, hindrer man i at systemet oppleves som tre separate fremfor ett integrert, koherent system.

Innholdet som vises på startsiden etter at bruker har logget seg inn må reflektere de rettigheter som den brukeren har i systemet. Det vil si at dersom brukeren for eksempel ikke er administrator, skal ikke hyperlenken til Presentation Manager vises på siden. Samtidig må listen over presentasjoner også kun inneholde de presentasjoner som den brukeren har tilgang til å se på i Presentation Viewer.

4.3 Overordnet laginndeling

Det er viktig at det nye systemet har en fornuftig og oversiktlig laginndeling. Uten det kan det fort skape problemer for dem som skal sette seg inn og videreutvikle systemet senere. Laginndelingen bør naturlig nok reflektere den inndelingen i de tre delsystemene, noe figur 4-3 illustrerer.



Figur 4-3: Overordnet oversikt over hvordan laginndelingen for DPG 2.0 er lagt opp

Laginndelingen er basert på den tradisjonelle måten å strukturere webapplikasjoner på. I bunn, i `core`, finner man persistenslaget, der lesing og skriving til og fra databasen utføres, som regel ved hjelp av designstrategien Data Access Object (DAO, Core J2EE Patterns, SUN). Over der igjen har man et servicelag som benytter domeneobjektene (i `beans`-pakken) og realiserer ulike use-cases gjennom grovkornede operasjoner.

Hver av delsystemene har sine egne spesialiserte service-klasser, som igjen benytter seg service-laget i `core`. Disse servicene kan ha behov for mer spesialiserte domeneobjekter enn det som er tilgjengelig i `core`, og hvert delsystem har derfor sin egen lokale `beans`-lag. Servicene er laget for å tilby grovere metoder for klassene i `controller`-laget, som utfører disse metodene i forbindelse med en http-forespørsel fra klienten.

Spring-rammeverket tilbyr sitt eget Model-View-Controller (MVC)-lag (Spring MVC, SpringSource), og har en rekke hjelpeklasser for å enklere utvikle controller-klasser. På toppen av hvert delsystem finner man `view`-laget, et lag som hovedsakelig realiseres ved hjelp av Java ServerPages (Java Server Pages, SUN).

4.4 Eksempel: Nettside for fjernundervisning

For lettere å kunne forklare de ulike konseptene i de kommende kapitlene, kan det være bra å ha et konkret, gjennomgående eksempel å forholde seg til. Siden DPG-en så langt har vært brukt til fjernundervisningskurs i Java-programmering, vil det være naturlig å utvikle en nettside rettet mot nettopp dette. Hvert kapittel vil bygge videre på dette eksempelet. Til slutt vil eksempelet bestå av en mer eller mindre komplett

presentasjonsmønsterspesifikasjon og presentasjonsspesifikasjon, som kan brukes som utgangspunkt for videre utvikling.

4.4.1 Strukturen på nettsidene

For å komme opp med en god struktur på kurssidene, tok man utgangspunkt i de eksisterende nettkurssidene. Disse sidene har stort sett fungert greit, men har fått kritikk for noe rotete oppsett, som fører til at det kan være vanskelig å finne frem til informasjonen. Etter å ha diskutert dette på en del møter, kom man frem til at kurssidene skulle bestå av følgende nettsider:

- **Startside**
  Dette er den første siden som møter brukeren når man kommer til siden. Den skal inneholde en kort velkomstmelding til nytte for nye brukere, og en liste over de siste meldingene som er lagt ut.

- **Fremdriftsplan**
  Fremdriftsplanen er den viktigste informasjonskilden for studentene, siden den inneholder en oversikt over alt som kurset inneholder. Studenten kan se kursets progresjon uke for uke, og klikke på en bestemt uke for å se mer detaljer om hva som skjer. For hver uke kan det legges ut informasjon om hvilke deler av pensum det skal arbeides med, hvilke oppgaver som skal løses og forskjellige ressurser som studentene kan laste ned og skrive ut.

- **Kursinformasjon**
  Fra denne siden vil studentene finne ut mer om hva kurset går ut på, hvilken litteratur som er på pensum, og hvordan eksamen er lagt opp. Det vil også være informasjon om hvordan man leverer inn obligatoriske oppgaver, samt en lise over hvem som er forelesere i kurset.

- **Ressurser**
  Her fins en oversikt over diverse ressurser som kan være nyttig for studentene gjennom kurset. Det kan være informasjon om nyttig tilleggsliteratur eller artikler som er relevant for kurset. Lenker til programvare, som kompilatorer og kildekodeverktøy, er også tilgjengelig her.

- **Kontakt oss**
  Denne siden forteller studenten hvem som er ansvarlig for kurset, både faglig og administrativt, og hvordan man raskt kan komme i kontakt med dem dersom det skulle dukke opp spørsmål underveis.

5 Analyse av tidligere forslag til presentasjonsmønster

Konseptet presentasjonsmønstrer er selve kjernen i den nåværende implementasjonen av DPG-systemet. Som tidligere avdekket i analysen av systemet er det rom for en rekke forbedringer i måten presentasjonsmønstrene defineres og brukes av systemet. I tiden etter at DPG 1.0 ble satt i drift, har det blitt gjort en del arbeid knyttet til utvikling av nye presentasjonsmønstre innenfor den nåværende implementasjonen (Lervik Larsen 2006), i tillegg til forslag til helt nye mulige implementasjoner (Rossini og Liberati 2005).

Vi vil gjennomgå det mest omfattende forslaget til ny presentasjonsmønster så langt og gjennomføre en kort analyse av de positive og negative sidene ved dette forslaget. Med analysen av dette forslaget som utgangspunkt, samt erfaringer høstet fra den nåværende implementasjonen av presentasjonsmønstre, vil det bli utarbeidet en ny implementasjon av presentasjonsmønsterkonseptet. Denne nye implementasjonen vil bli fundamentet som de tre delsystemene, Presentation Viewer, Presentation Content Editor og Presentation Manager, bygger på.

Før vi begynner, kan det være fornuftig å få på plass noen enkle definisjoner som kommer til å bli brukt kontinuerlig gjennom dette kapittelet. Med begrepet *presentasjonsmønsterspesifikasjon* mener man da måten et konkret presentasjonsmønster, eksempelvis `coursePattern`, defineres på. En presentasjon som benytter seg av en presentasjonsmønsterspesifikasjon, er definert ved hjelp av en presentasjonsspesifikasjon. Når teksten bruker begrepet *presentasjons-mønster*, omtaler man da hele konseptet, altså presentasjons-mønsterspesifikasjon og presentasjonsspesifikasjon under ett.

5.1 Utgangspunktet for det nye presentasjonsmønsteret

Det mest omfattende arbeidet innen presentasjonsmønstre siden DPG 1.0 ble utført av Alessandro Rossini og Graziano Liberati som en oppgave i kurset INF219, 2006. Dette arbeidet resulterte i et nokså detaljert utkast til ny implementasjon av presentasjonsmønster, som vi kort vil gå gjennom her. For en grundigere gjennomgang, se den endelige rapporten (Rossini og Liberati 2005).

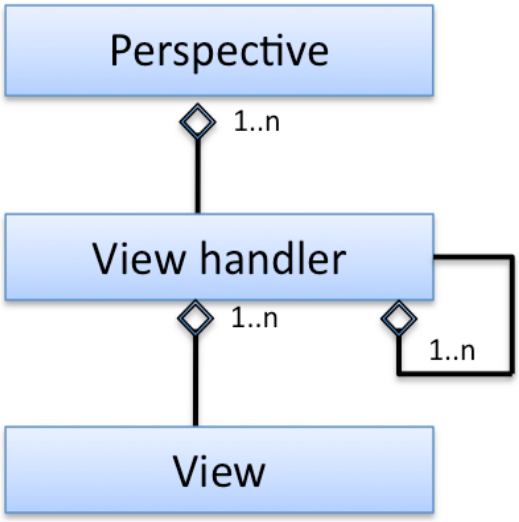
I tillegg bidro de med en del nye tanker ideer rundt presentasjonsmønstre, bl.a. konseptene *spesifikasjon* og *instansiering*. Man kan se på presentasjons-mønsterspesifikasjonen som spesifikasjonen av strukturert innhold for en applikasjon, og presentasjonsspesifikasjonen som en instans av innholdet til en bestemt applikasjon. Som i den tidligere implementasjonen, ble hovedideen bak presentasjonsmønstre ivaretatt gjennom ”specify once and user several times” altså gjenbruken av mønstre i flere presentasjoner.

5.1.1 Forslaget til ny presentasjonsmønsterspesifikasjon

Presentasjonsmønsterspesifikasjonen lar en utvikle bestemme hvilken struktur innholdet i en presentasjon skal ha. I likhet med den gamle implementasjonen av presentasjonsmønsterspesifikasjonen består den av tre hovedelementer:

- Views
- View Handlers
- Perspectives

Forholdet mellom de forskjellige elementene er illustrer i figur 5.1. Denne strukturen er nokså lik strukturen mellom `perspectives`, `content-handlers` og `views` i den gamle spesifikasjonen (Espelid 2004), men måten de tre elementene skal benyttes er betydelig endret.



Figur 5-1: Oversikt over de ulike elementene i presentasjonsmønsterspesifikasjonen

Views
Et `view` er ment som en abstraksjon mellom persistenslaget og resten av applikasjonen. Disse er som regel knyttet direkte til en bestemt entitetstype, slik at man får en 1-til-1-relasjoner mellom `view` og entiteter. Et view defineres ved hjelp av en unik identifikator, samt en liste over hvilke `fields` det inneholder, som vist i eksempel 5-1. I dette tilfelle representerer viewet en student som har et studentnummer, fullt navn og et gitt antall studiepoeng.

```
<views>
    <view
        id="student">
        <fields>
            <field>studentNumber</field>
            <field>fullName</field>
            <field>studyPoints</field>
        </fields>
    </view>
</views>
```

Eksempel 5-1: Hvordan et view kan defineres i spesifikasjonen

View Handlers
Som navnet tilsier, er hovedoppgaven til dette elementet å håndtere views på ulike måter. Funksjonaliteten som tilbys er bl.a. endringer i måten informasjonen er strukturert på, filtrering av elementer basert på datomatching eller brukerinput, eller sortering av elementene etter bestemte regler. Et view kan assosieres med flere forskjellige view handlers, noe som gjør at samme informasjon kan transformeres og presenteres på forskjellige måter i ulike sammenhenger. Dette gir stor grad av fleksibilitet i utviklingen av presentasjonsmønsterspesifikasjoner.

```
<view-handlers>
    <view-handler
        id="studentList">
        <associated-views>
            <associated-view>student</associated-view>
        </associated-views>
        <fields>
            <field>student.fullName</field>
            <field>student.studyPoints</field>
        </fields>
        <filters>
            <filter>
                <field>student.studyPoints</field>
                <value>30</value>
            </filter>
        </filters>
        <transformations>
            <transformation>sortStudents</transformation>
        </transformations>
    </view-handler>
</view-handlers>
```

Eksempel 5-2: Hvordan en view handler kan defineres

I eksempel 5-2 har vi definert en `view-handler` for viewet `student` definert tidligere. Deretter velger vi å kun ta med de angitte feltene som er listet under `fields`-taggen i blokken etter. Kun felter som listes her blir tilgjengelige for dem som ønsker å benytte denne view-handleren. Ved hjelp av et filter begrenses listen med studenter til de studentene som har et bestemt antall studiepoeng. Til slutt sorteres listen ved hjelp av en transformasjon, som i en tenk implementasjon trolig vil svare til et XSLT-dokument.

Perspectives
Et `perspective` representerer innholdet til en bestemt nettside som en presentasjonsspesifikasjon kan rendre å forskjellige måter. Strukturen på dette innholdet blir definert ved hjelp av et XSD-dokument. Siden dem som utvikler presentasjonsmønsterspesifikasjonen og dem som presentasjonsspesifikasjonen ofte er forskjellige personer, blir dette XSD-dokumentet svært viktig slik at begge parter er enige om denne strukturen. Et perspective bygges opp av en eller flere view-handlers, som igjen kan ha avhengigheter til hverandre. Det er da mulig å lage regler for hvilke view-handlers som skal vises avhengig av hvilke lenker bruker har klikket på innenfor samme nettside. Eksempel 5-3 gir et innblikk i hvordan et perspektiv kan defineres.

```
<perspectives>
    <perspective id="studentPerspective">
        <description>A list of students</description>
        <schema>studentListXSD</schema>
        <composition>
            <content>studentList</content>
        </composition>
    </perspective>
</perspectives>
```

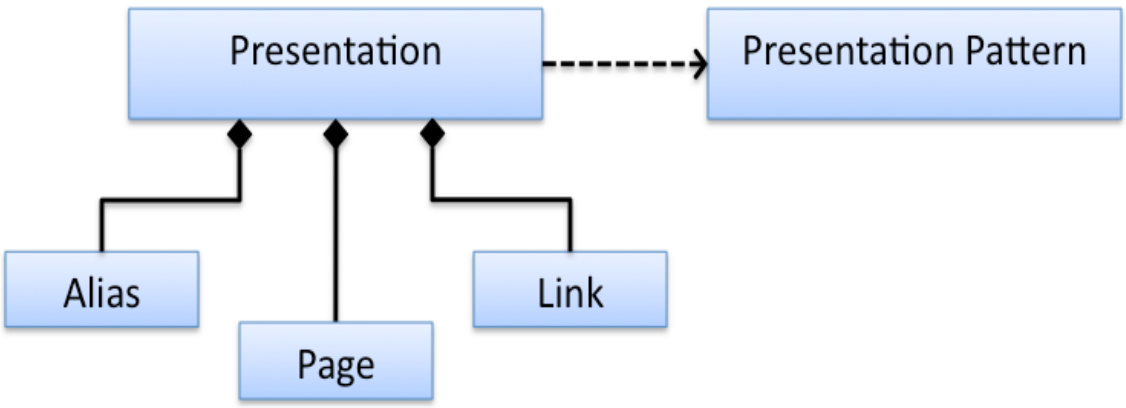
Eksempel 5-3: Et perspective som inneholder view-handleren `studentList`

5.1.2 Forslaget til ny presentasjonsspesifikasjon

Presentasjonsspesifikasjonen definerer man mappingen mellom views og de faktiske datakildene. Her blir det også bestemt hvordan det strukturerte innholdet som presentasjonsmønsterspesifikasjonen definerer skal rendres for klienten som ønsker tilgang til informasjonen. Denne spesifikasjonen består av følgende tre elementer:

- Alias
- Page
- Link

Hver presentasjonsspesifikasjon angir først hvilken presentasjonsmønsterspesifikasjon som skal benyttes. I tillegg gis presentasjonen en unik tittel og en beskrivelse av hva presentasjonen inneholder. Figur 5-2 viser en oversikt over de ulike elementene i presentasjonsspesifikasjonen, forholdet mellom de for skjellige elementene og hvordan presentasjonsspesifikasjonen benytter seg av presentasjonsmønsterspesifikasjonen.



Figur 5-2: Oversikt over de ulike elementene i presentasjonsspesifikasjonen

Alias

Et `alias` definerer en mapping mellom et `view` og konkrete entiteter i persistenslaget. Denne kan benytte seg av forskjellige typer datakilder, eksempelvis en tabell i en database eller i en XML-fil. Når et bestemt view er valgt og datakildene definert, starter man mappingen mellom innholdet i datakilden og de forskjellige feltene i viewet. I eksempel 5-4 har vi mappet et av feltet `fullName` i viewet `student` til kolonnen `nameOfStudent` i `studentTable` i en database.

```
<aliases>
    <alias name="student" kind="DB" url="db:studentDatabase:8023">
        <attributes>
            <attribute name="fullName"/>
            <mapping>
                <entity>studentTable</entity>
                <attribute>nameOfStudent</attribute>
            </mapping>
            <!-- mapping for other fields... -->
        </attributes>
    </alias>
</aliases>
```

Eksempel 5-4: Viser hvordan et `alias`-element kan defineres

Page

En `page` bestemmer hvordan informasjonen definert av et `perspective` skal rendres. Mer konkret definerer man en binding mellom et perspective og en bestemt rendringsteknologi. Ved å knytte samme perspectives til ulike pages kan man rendre samme informasjon på forskjellige måter (som for eksempel HTML, WML, XML, eller lignende teknologier). Her bestemmer man også hvilke sider som skal være startsiden, og om siden for øyeblikket er tilgjengelig for klientene eller ikke. Det `page`-elementet som har attributten `default` satt til `true` vil bli brukt som startside, mens attributten `available` bestemmer om en nettside skal være tilgjengelig for klientene.

I eksempel 5-5 har vi definert en nettside som inneholder en liste over studenter, med informasjonen som finnes i `studentPerspective`. Nettsiden rendres i dette tilfelle til HTML ved hjelp av en JSPX-side.

```
<pages>
    <page
        id="studentPage"
        default="false"
        description="A page contaning students"
        available="true">
        <perspective>studentPerspective</perspective>
        <layout>studentLayout.jspx</layout>
    </page>
</pages>
```

Eksempel 5-5: Viser hvordan et page-element kan defineres i spesifikasjonen

Link
Dette elementet lar oss sette opp navigasjon mellom de forskjellige sidene. En link angis på samme måter som kanter i en rettet graf, altså med en kilde og en destinasjon. Ved hjelp av denne strukturen kan systemet automatisk bygge opp navigasjonselementer på sidene. I eksempel 5-6 defineres navigasjonsmulighetene brukeren har fra siden `studentPage`. Dersom denne siden renders som en HTML-side, vil det finnes HTML-lenker til `startPage` og `resourcePage` i en meny på nettsiden.

```
<links>
    <link sourcePage="studentPage">
        <destination-pages>
            <page>startPage</page>
            <page>resourcePage</page>
        </destination-pages>
    </link>
</links>
```

Eksempel 5-6: Viser hvordan navigasjonen mellom nettsidene kan defineres i spesifikasjonen

5.1.2 Problemer med dette forslaget

Sammenliknet med den gamle implementasjonen av presentasjonsmønsteret er det nye forslaget beskrevet i (Rossini og Liberati 2005) en kraftig forbedring. Først og fremst er det mye mer tydelig hva oppgavene til de forskjellige elementene er, og på måten de benyttes på i spesifikasjonsdokumentene. Det er lagt stor vekt på fleksibilitet, og forslaget er ikke bundet opp til noen konkret teknologi, foruten XML-teknologien som spesifikasjonsdokumentene er skrevet i. Til tross for dette, er det flere potensielle problemer med dette forslaget som gjør at man kan benytte det uten videre til DPG 2.0.

Problemer med transformering
Forslaget som presenteres er på enkelte steder altfor generisk, noe som skaper usikkerhet rundt hvordan ting skal implementeres i praksis. Spesielt gjelder dette forholdet mellom `perspectives` og `pages`. Et perspective, som beskrevet tidligere, representerer et stort XML-dokument som inneholder all informasjonen som skal presenteres på en side. En page derimot, er ansvarlig for å knytte denne informasjonen til en bestemt rendringsteknologi, eksempelvis ved hjelp av JSPX. Problemet med dette er at JSPX ikke egner seg til å transformere XML til HTML, som er presentasjonsformat som er ønskelig i websammenheng. Her bør man heller benytte XSLT, en teknologi spesielt egnet for transformering av XML.

Et annet problem er at det legges opp til at en view-handler kan hente informasjon fra flere enn ett view om gangen. En transformasjonsfil er som regel beregnet for å transformere ett og ett XML-tre om gangen, mens man i dette forslaget forventer at flere XML-dokumenter skal takles om gangen. I tillegg er det mulig å liste flere transformasjonsfiler, som trolig betyr at man kan kjede XSLT-transformasjoner etter hverandre.

Selv om dette i få tilfeller kan være nyttig, blir det vanskelig for presentasjonsmønsterutviklere å holde oversikten over strukturen på XML-dokumentet etter hver transformasjon. Man må holde oversikt over strukturen på resultatet fra forrige transformasjon, og deretter sørge for at XSLT-filen som kommer etter er implementert i henhold til resultatet av den forrige. Dette blir fort svært uoversiktlig, selv etter bare to eller tre serietransformasjoner, og det kan derfor stilles spørsmål med hvor nyttig denne funksjonaliteten er i forhold til hvor mye kompleksitet som introduseres i systemet.

Komplekse XML-dokument
Det er også en ulempe at alt innholdet i et perspective samles i ett XML-dokument. Dette dokumentet kan bli ganske stort, og inneholde informasjon fra en rekke forskjellige view-handlers. Den som deretter skal rendre dette til et annet format må da kjenne til strukturen til hvert enkelt view, og implementere rendringsprosessen for alle sammen i ett og samme XSLT-dokument. Dette kan fort bli uoversiktlig og vanskelig å håndtere. Dersom man i stedet konverterte XML-innholdet fra hver view-handler hver for seg, i separate transformasjoner, for så og sette det sammen igjen i en page, vil det bli mye lettere å håndtere for utviklerne.

Stort fokus på XML Schemas
Både i det gamle systemet, og til en viss grad i det nye forslaget, var det stor fokus på bruken av XML Schemas (XML Schema, W3C) for å definere strukturen på informasjonen som skal rendres. Selv om slike skjemadefinisjoner kan være svært nyttige for å sørge for at XML-dokumentene har korrekt innhold, er disse skjemaene tidkrevende å utvikle og vedlikeholde. I det gamle systemet måtte man redigere XML-filer direkte for å endre på innholdet, og man brukte da XML Schemas for å hindre at innholdsdokumentene inneholdt gal struktur. I det nye systemet vil man utvikle et fullverdig redigeringsverktøy som gjør det umulig for bruker å tukle med strukturen og dermed eliminerer man muligheten for at strukturen til XML-dokumentene blir ugyldig. Dermed er det heller ikke lenger behov for å definere XML Schemas for innholdsfilene, noe som gjør situasjonen for utviklere betydelig enklere.

Ingen fungerende prototype å vise til
Forslaget bærer noe preg av å ha blitt designet ”på papiret”, uten at man har forsøkt å implementere det i praksis. Ved ikke å utvikle en prototype sammen med spesifikasjonen, er det lett for å ta med funksjonalitet som i praksis vil være unødvendig eller overflødig, og som vil kunne implementeres på en mye bedre måte. Et eksempel på dette finner man i definisjonen av view-handlers. All filtrering og sortering som man ser i eksempel 5-2 kan man enkelt gjøre i en og samme XSLT-transformasjon. XSLT er en meget kraftig teknologi som sammen med XPath kan utføre en mengde forskjellige operasjoner, bl.a. sortering og filtrering, i tillegg til selve transformeringen. Denne teknologien har ikke blitt utnyttet til det fulle verken i det gamle systemet og heller ikke i dette forslaget.

For generisk spesifikasjon
Både i den tidligere implementasjonen av presentasjonsmønstre (Espelid 2004) og i forslaget til (Rossini og Liberati 2005) ble det lagt vekt på at spesifikasjonene skulle være så generiske så mulig, dvs. at det skulle ikke være knyttet opp til konkrete teknologier

(eksempelvis XSLT/HTML) eller kontekster (web, desktop-applikasjon, pdf-generator, osv.). Dette førte til flere ulemper enn fordeler, blant annet ved at selve spesifikasjonene lett blir for abstrakte og upresise. Mulighetene til å dra nytte av spesifikke teknologier som er tilgjengelige innenfor et bestemt formål (som for eksempel web) blir da svekket.

En annen fare ved å la spesifikasjonen være for generisk, er at man risikerer å måtte utvikle et eget domenespesifikt språk (Fowler, Domain Spesific Language) som oversetter mellom den generiske spesifikasjonen og det konkrete domenet som presentasjonsmønstrene skal benyttes i. Da er det heller bedre at man bestemmer seg for å lage en implementasjon av presentasjonsmønster som er rettet mot ett bestemt formål. Dette vil gjøre det mye lettere for dem som skal utvikle presentasjonsmønstre for nettsider, siden spesifikasjonen da er skreddersydd webutvikling.

Problemer med navigasjon mellom nettsidene

`Link`-elementet slik det presenteres i presentasjonsspesifikasjonen gjør det mulig å skreddersy navigasjonen mellom de ulike sidene. Det er for eksempel fullt mulig å utvikle en kompleks graf for navigasjonen, og dermed lage forskjellige veivisere med ulike stier og løkker. Problemet er at i websammenheng blir denne implementasjonen for enkel. Når man benytter slike kompliserte veivisere er det også behov for overføring av tilstand fra en side til en annen. Som regel vil navigasjonsmuligheten endre seg basert på hvilke valg bruker gjør på forrige side. Det er i så fall behov for mer kompliserte regler for hvordan navigasjonen skal foregå, og den enkle grafimplementasjon som blir foreslått blir da for simpel.

Problemer knyttet til persistens

I forslaget legger man opp til at systemet skal kunne hente informasjon fra en rekke forskjellige informasjonskilder, deriblant databaser og XML-dokumenter. Det er også foreslått en enkel 1-1-mapping mellom entiteter i disse informasjonskildene og `views` i presentasjonsmønsterspesifikasjonen. I utgangspunktet er dette en veldig fleksibel løsning, som gjør det mulig å lage et system som aggregerer informasjon fra eksisterende systemer. Men det er flere problemstillinger knyttet til hvordan dette skal foregå i praksis.

Et av problemene med alias-elementet er at kun en enkel 1-1-mapping er mulig. Da vil man for eksempel ikke kunne hente informasjon fra mer enn én databasetabell om gangen. I praksis er det ønskelig å hente informasjon fra flere databasetabeller via en SQL-spørring, noe som ikke er mulig i forslaget. Det er også tenkelig at man ønsker å sette sammen informasjon fra forskjellige databaser, eller fra totalt forskjellige typer datakilder, i ett og samme view. Da blir strukturen som er foreslått for enkel og begrensende.

Det er heller ikke tatt hensyn til hvordan disse datakildene skal aksesseres, foruten selve databaseadressen eller filbanen til XML-dokumentene. Som regel er slike datakilder beskyttet på forskjellige måter. Databaser beskyttes gjerne med et enkelt brukernavn og passord. XML-dokumenter kan være filer som ligger lokalt på samme maskin som DPG-systemet, men de kan også ligge på en nettverksdisk eller det kan ligge på en URL tilknyttet en web service som genererer XML-dokumenter. Da kan det være snakk om en

rekke forskjellige sikkerhetsinnretninger som systemet må kjenne til, for eksempel HTTP Basic, Form-basert innlogging eller andre mer kompliserte former for autentisering.

Å *lese* data fra kildene og transformere data etter behov vil trolig ha ingen innvirkning på datakildene. Det er først når man ønsker å *skrive* data til datakildene at ting kan gå galt. Det kan ikke anses som trygt å endre på data i, for eksempel, en database tilhørende et eksternt system så lenge presentasjonsutviklere ikke har god kjennskap til hvilke dataformat og regler som gjelder for det eksterne systemet. En tryggere løsning vil være å la DPG-systemet lese og skrive til eksterne datakilder gjennom et egnet API, slik at selve skrivingen til databasen gjøres av det eksterne systemet og ikke DPG-en. Så lenge de eksterne systemene implementerer dette API-et vil DPG-systemet ikke trenge å vite noe om hvordan de eksterne systemene lagrer sine data. Den store ulempen er selvsagt at man må implementere API-et i alle eksterne systemer man ønsker at DPG-systemet skal kommunisere med, noe som trolig er urealistisk.

En bedre og mer realistisk løsning ville være å la DPG 2.0 ha sin eget persistenslag. Da unngår man disse problemene, da systemet vil ha alle rettigheter for å opprette, slette og modifisere innholdet i de forskjellige presentasjonene. Systemet vil da bli lettere å administrere med tanke på å ta sikkerhetskopi av data. Man vil heller ikke lenger være avhengig av en rekke eksterne informasjonskilder, noe som er bra for stabiliteten og tilgjengeligheten til systemet. Den store ulempen med dette er selvsagt at informasjonen fra de eksterne systemene ikke kan brukes direkte, men må kopieres inn i dette persistenslaget. Men stiller man dette opp mot alle de problemene beskrevet ovenfor, fremstår dette likevel som en bedre løsning.

5.2 Målsetninger for det nye presentasjonsmønsteret

På bakgrunn av de analyser som er gjort både av det eksisterende presentasjonsmønsteret og forslaget til det nye, er det utarbeidet følgende målsetninger for den nye implementasjonen av presentasjonsmønsteret:

- Alle spesifikasjonselementene skal ha en lettfattelig og intuitiv deklarasjon, og ha klart definerte roller. Dette er viktig for å unngå forvirring og dermed feil bruk av elementene.

- Det må foreligge bedre samsvar mellom spesifikasjonene og den tilhørende dokumentasjonen. Dette er svært viktig for vedlikehold og videreutvikling av presentasjonsmønstrene.

- Mengden XML som må skrives for hånd må begrenses for å redusere tiden det tar å utvikle og vedlikeholde mønstre. Overflødige elementer og attributter bør unngås der det er mulig.

- Det bør baseres på konseptet `spesifikasjon/instansiering`, som ble introdusert i forslaget ovenfor. For å gjøre det enklere for systemet å opprette nye presentasjoner, samt minimere manuell konfigurasjonen for brukeren, bør

presentasjonsmønsterspesifikasjonen inneholde alt som trengs for å opprette en presentasjon.

- Ikke alle presentasjoner ønsker å bruke presentasjonsmønsteret på akkurat samme måte. Derfor bør det være mulig for en presentasjon å overkjøre en del av konfigurasjonen som den får fra mønsterspesifikasjonen slik at presentasjonen kan skreddersys til sitt bruksområde.

- Man bør eliminere behovet for manuell mapping mellom entiteter og elementer i presentasjonene, og heller satse på å ha et selvstendig persistenslag innebygget i systemet. Dette vil redusere tiden det tar å opprette nye presentasjoner betraktelig, og vil forenkle presentasjonsspesifikasjonen betydelig.

- Siden systemet selv tar seg av opprettelsen og vedlikehold av innholdsfilene i persistenslaget, er det ikke lenger behov for utviklerne å opprette XML Schemas, som sparer mye tid under utviklingen av nye presentasjonsmønstre.

- Det må være bedre støtte for å presentere samme innhold på forskjellige måter i en og samme presentasjon. Som utvikler av dynamiske nettsider er det ofte behov for å presentere en og samme informasjon på forskjellige måter, avhengig av hvor på nettsiden informasjonen skal plasseres. Da unngår man at innhold dupliseres, og man gir utviklerne større frihet under design av de ulike nettsidene.

- Det bør være enklere å definere navigasjonen mellom de forskjellige nettsidene. For å unngå at man implementerer en kompleks, men likevel svært begrenset graf som tidligere foreslått, bør man heller satse på en enkel, flat navigasjonsstruktur. Dette er den mest naturligste og trolig vanligste strukturen for en nettside, hvor man har en sterkt koblet graf (Graph Connectivity, Wikipedia) der alle page-elementene er tilgjengelig fra alle de øvrige page-elementene. Da vil det ikke lenger være behov for å definere navigasjon i mønsterspesifikasjonen, siden systemet enkelt han opprette denne navigasjonsmodellen automatisk.

6 Presentasjonsmønstre i DPG 2.0

Ved hjelp av alle de tidligere analysene og de målsetninger som er satt, er det utviklet et nytt og utbedret forslag til implementasjonen av presentasjonsmønster for DPG 2.0. Navnene på en del av elementene er hentet fra tidligere implementasjoner og forslag, men betydningen og funksjonen til disse er endret betraktelig. En del elementer fra det forslaget beskrevet i (Rossini og Liberati 2005), som for eksempel `alias` og `link`, er utelatt i henhold til de målsetninger som ble satt.

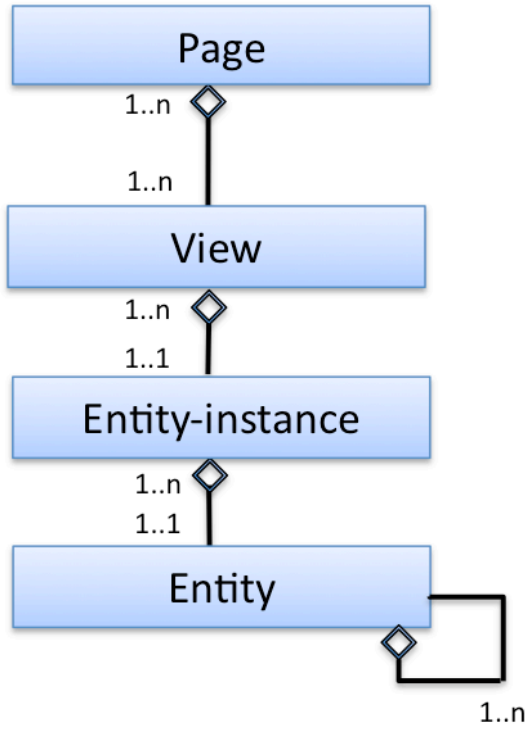
I dette kapittelet går vi grundig gjennom de forskjellige konseptene i den nye implementasjonen av presentasjonsmønstre. En solid forståelse av disse konseptene er nødvendig for å forstå hva som ligger til grunn for de forskjellige delsystemene som DPG 2.0 består av. Først tar vi for oss den nye presentasjonsmønsterspesifikasjonen, deretter den nye presentasjonspesifikasjonen og til slutt kommer en kort oppsummering av hva som er oppnådd med den nye implementasjonen.

6.1 Den nye presentasjonsmønsterspesifikasjonen

I tillegg til elementene `title` og `description`, består den nye presentasjonsmønster-spesifikasjonen av følgende hovedelementer:

- Entities
- Entity-instances
- Views
- Pages

Figur 6-1 viser en oversikt over de forskjellige elementene i spesifikasjonen og relasjonene mellom dem. Som vi ser av figuren, består et `page`-element av en eller flere `views`, og et `view` kan forekomme i en eller flere `pages`. En `entity instance` kan forekomme i en eller flere `views`, men et `view` kan derimot kun inneholde èn entity instance. En `entity instance` kan igjen inneholde kun èn `entity`, mens en `entity` kan forekomme i flere `entity instances`, i tillegg til å nøstes inni en annen entitet. Formålet til hver av disse fire elementene vil bli gjennomgått i dette avsnittet.



Figur 6-1: Oversikt over de forskjellige elementene i en presentasjonsmønsterspesifikasjon

6.1.1 Entities

Siden systemet nå skal ha sitt eget persistenslag, har mønsterutviklerne nå full kontroll over strukturen til informasjonen i et mønster. Denne strukturen defineres ved hjelp av elementet `entity`. Hver `entity` tildeles en unik identifikator, samt en liste over de feltene som hører til denne entiteten.

```
<!-- Entity used for general information -->
<entity id="informationEntity">
    <fields>
        <field type="string" required="true">title</field>
        <field type="xhtml">content</field>
    </fields>
</entity>
```

Eksempel 6-1: Hvordan entiteter deklareres i presentasjonsmønsterspesifikasjonen

I eksempel 6-1 ser man hvordan entiteter kan deklareres. I dette tilfelle deklarerer man en entitet kalt `informationEntity`, som består av to felt, `tittel` og `content`. Det første feltet er en vanlig streng uten formatering, mens det andre feltet er ment for å håndtere en større mengde tekst med formateringsmuligheter.

Hvert felt har minimum et navn og en attributt som indikerer hvilken felttype det er snakk om. Tabell 6-1 inneholder en oversikt over de forskjellige felttypene som er støttet og hvilken funksjonalitet de tilbyr.

Det er også mulig å bruke entiteter man selv har definert som felttype. Man kan altså nøste entiteter inni hverandre, noe som gjør spesifikasjonen svært fleksibel. Det fins to felttyper som muliggjør slik nøsting; `entity` og `entity-list`. Den første av dem gjør det mulig å nøste en konkret entitet under en annen. I dette tilfelle vil Presentation Content Editor tilby funksjonalitet for å redigere denne nøstede entiteten. Sistnevnte representerer en liste med entiteter av en bestemt type. Hver gang felttypen `entity-list` dukker opp, vil Presentation Content Editor automatisk tilby funksjonalitet for å legge til, redigere og slette entiteter i listen.

Felttype	Beskrivelse
string	Brukes til enkle strenger som ikke trenger spesiell behandling av systemet. Denne typen kan brukes i kombinasjon med attributten `required` satt til `true`, som vil hindre at brukere unnlater å fylle inn dette feltet i editoren. Man kan også legge til attributten `allowedValues`, for å begrense hvilke lovlige verdier tekstfeltet kan inneholde. Se eksempel 6-2, der denne attributten er benyttet.
xhtml	Benyttes til rik tekst der man blander inn XHTML-tagger sammen med innhold. Det gjør det mulig for bruker å legge inn formatering, som for eksempel uthevet skrift, punktlister og farger på tekst. Denne kan kombineres med attributten `required="true"`, som vil hindre at brukere unnlater å fylle inn dette feltet i editoren.
date	Brukes når man ønsker en enkel dato, som vil redigeres ved hjelp av en datovelger i Presentation Content Viewer. Attributten `required` er ikke støttet for denne felttypen.
file	Denne felttypen benyttes når man vil gjøre filer tilgjengelige for nedlasting. Feltet vil da inneholde en referanse til en fil i persistenslaget, og Presentation Content Viewer vil da kunne tilby skjemaelementer for filopplasting. Attributten `required` er ikke støttet for denne felttypen.
plugin	Som navnet tilsier, brukes dette i forbindelse med plugins, og må kombineres med attributt som angir hvilken pluginkonfigurasjon som skal benyttes. Se (Ingvaldsen 2008) for mer informasjon om bruken av plugins. Attributten `required` er ikke støttet for denne felttypen.
entity	Brukes for å nøste en entitet inni en annen. Attributten `ref-id` som angir entitetstypen som skal nøstes, er påkrevd i dette tilfelle. Attributten `required` er ikke støttet for denne felttypen.
entity-list	Brukes for å nøste en *liste* over entitet inni en annen entitet. Attributten `ref-id` som angir entitetstypen er påkrevd også her. Attributten `required` er ikke støttet for denne felttypen.

Tabell 6-1: Oversikt over felttyper og bruksområdet for hver av dem

Det er også mulig å nøste entiteter i flere nivåer, og i teorien kan man ha et uendelig antall nivåer. Dette gjør det mulig å for eksempel ha en liste over personer, der hver person har hver sin liste over sine favorittalbum, og under hver album finner man en liste over sanger, osv. Men flere enn tre til fire nivåer vil nok gjøre det vanskeligere å utvikle

og vedlikeholde mønsteret. Trolig vil en til tre nivåer være mer realistisk i de fleste presentasjonsmønsterspesifikasjonene.

```
<entity id="messageEntity">
    <fields>
        <field type="string" required="true">topic</field>
        <field type="xhtml">body</field>
        <field type="string" allowedValues="green;yellow;red">level</field>
    </fields>
</entity>

<entity id="messageListEntity">
    <fields>
        <field type="entity-list" ref-id="messageEntity">messages</field>
    </fields>
</entity>
```

ss

Eksempel 6-2: Eksempel på nøsting av entiteter

I eksempel 6-2 ser vi hvordan man kan benytte felttypen entity-list for å nøste en entitet inni en annen. Her ser vi hvordan nøsting av entiteter kan foregå. Entiteten `messageEntity` består av tre felter; en overskrift til tema for meldingen og en meldingskropp (formatert med XHTML). Det siste feltet indikerer viktigheten av meldingen, som kan avgjøre hvordan meldingen vises på nettsiden. Her brukes den valgfrie attributten `allowedValues`, som vil føre til at feltet vises som en dropdown-boks med de angitte verdiene i Presentation Content Editor i stedet for et vanlig tekstfelt. Denne entiteten benyttes deretter av entiteten `messageListEntity`, som inneholder en liste over meldinger.

6.1.2 Entity-Instances

Det er viktig å få frem at entiteter kun definerer en *struktur* på hvordan informasjonen skal se ut, litt på samme måte som en klasse i et objektorientert programmeringsspråk definerer hvilke feltvariabler som fremtidige objekter skal ha. Selve instansieringen av innholdsfilene skjer i elementet `entity-instance`, som vist i eksempel 6-3.

```
<entity-instance id="messageListEntityInstance">
    <entity>messageListEntity</entity>
</entity-instance>
```

Eksempel 6-3: Informasjonsinstanser av entiteter definert i presentasjonsmønsterspesifikasjonen

Når systemet skal opprette en ny presentasjon basert på en eksisterende presentasjonsmønsterspesifikasjon, spiller elementet `entity-instance` en avgjørende rolle. Det er på bakgrunn av disse elementene at innholdsfilene for presentasjonen opprettes. For hver `entity-instance` opprettes det et XML-dokument som har samme struktur som den tilhørende entiteten, og innholdet i hvert av disse kan nå endres på ved hjelp av Presentation Content Editor. Hver gang brukeren

velger å gjøre endringer i informasjonen på nettsidene, vil altså disse endringene bli skrevet til disse innholdsfilene. Og hver gang nettsidene skal rendres igjen av Presentation Viewer, vil systemet lese fra disse innholdsfilene.

Eksempel 6-4 viser hvordan et slikt innholdsdokument kan se ut. Her ser vi hvordan innholdsfilen til entity-instance `messageListEntityInstance` kan se ut, etter at en bruker har lagt inn to meldinger i listen. Innholdsfilene kan fort bli ganske omfattende, men det er ikke noe problem i praksis da strukturen genereres og modifiseres av DPG-systemet uten noe behov for manuell redigering.

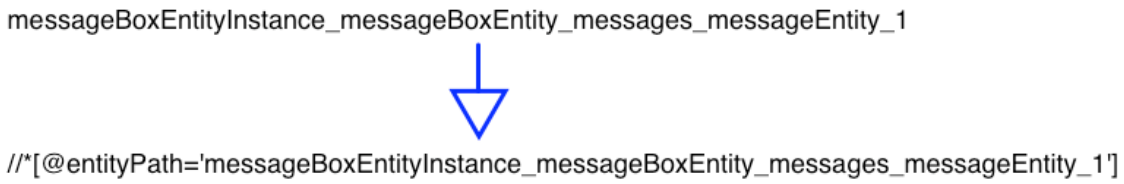
```
<messageListEntity  type="rootEntity" entityPath="messageListEntityInstance_messageListEntity"
                    createdBy="user123" updatedBy="user123"
                    createdOn="18/06/2008 15:35" updatedOn="19/06/2008 14:14">
    <messages type="entity-list"
              entityPath="messageListEntityInstance_messageListEntity_messages_messageEntity">
        <messageEntity type="subEntity"
                       entityPath="messageListEntityInstance_messageListEntity_messages_messageEntity_1"
                       createdBy="user123" updatedBy="user123"
                       createdOn="19/06/2008 14:14" updatedOn="19/06/2008 14:14">
            <topic type="string">Velkommen</topic>
            <body type="xhtml">
                <![CDATA[<p>Velkommen til nettsidene for kurset!</p>]]>
            </body>
            <level type="string" allowedValues="green;yellow;red">green</level>
        </messageEntity>
        <messageEntity type="subEntity"
            entityPath="messageListEntityInstance_messageListEntity_messages_messageEntity_2"
            createdBy="user123" updatedBy="user123"
            createdOn="19/06/2008 14:14" updatedOn="19/06/2008 14:14">
            <topic type="string">Eksamenstidspunkt er endret</topic>
            <body type="xhtml">
                <![CDATA[<p>Eksamen er flyttet til <b>onsdag 10/07, kl 9.00</b>!</p>]]>
            </body>
            <level type="string" allowedValues="green;yellow;red">red</level>
        </messageEntity>
        <!-- ... flere messageEntity-elementer ... -->
    </messages>
</messageListEntity>
```

Eksempel 6-4: Et eksempel på hvordan innholdsfilen til en `entity-instance` kan se ut

I tillegg til å holde på selve *innholdet*, har innholdsfilene også en del metainformasjon i form av XML-attributter. Denne metainformasjonen består først og fremst av informasjon om strukturen til de forskjellige feltene for hver av entitetene som dokumentet kan inneholde. I utgangspunktet virker denne type informasjon unødvendig, siden systemet kan få tak i dette ved å se på strukturen til hver av `entity`-elementene som denne `entity-instance` er koblet til. Det ble gjort en del forsøk på dette hvor systemet hele tiden måtte ”slå opp” i entitetene hver gang det var behov for å avgjøre strukturen på entitetene og felttypene i innholdsfilen. Men dette viste seg fort å være lite effektivt og førte til mye unødvendig kode, med det resultat at man heller valgte å angi både navn på entiteter og felttyper direkte i innholdsfilen. På den måten unngår man å ha

kostbar oppslagskode flere steder i koden, i tillegg til at all metainformasjon om informasjonen er lettere tilgjengelig for XSLT-parseren.

Hver av entitetene i innholdsfilene har også en del metainformasjon. For hver entitet er det angitt hvilken rolle og plassering den har i innholdsdokumentet, for eksempel om det er snakk om en entitet på toppnivå, eller en nøstet entitet (subentitet). Attributten `entityPath` indikerer hvilken plassering denne entiteten har i innholdsdokumentet. En `entityPath` minner en del om XPath og har omtrent samme funksjon, men inneholder ikke problematiske tegn som /, [eller], som gjør det lettere å sende verdien av en `entityPath` som `get`-parametere i URL-er. En `entityPath` kan enkelt oversettes til XPath når det skulle være nødvendig, noe systemet gjør automatisk etter behov.



Figur 6-2: Hvordan en `entityPath` oversettes til XPath

Figur 6.2 viser hvordan en `entityPath` til en melding i en liste over meldinger blir oversatt til XPath. Den aktuelle `entityPath` i figuren peker på en bestemt `messageEntity` (med id 1), i en liste over meldinger (`messages`). Disse meldingene hører til under entiteten messageBoxEntity, som er definert i innholdsdokumentet `messageBoxEntityInstance` (XML). Denne blir så tolket og oversatt til et XPath-uttrykk (XPath, W3C) som gjør det enkelt for resten av systemet å lokalisere informasjonen som skal hentes frem ved hjelp av ordinære XML og XSLT-operasjoner.

I de fleste innholdshåndteringssystemer er det ønskelig å holde oversikt over når informasjon sist ble endret, og av hvem. For hver entitet er det derfor også angitt når denne entiteten ble opprettet, når den sist ble endret, og hvem (dvs. hvilken bruker) som evt. utførte opprettelsen eller endringene. Presentation Content Editor vil sørge for at denne informasjonen oppdateres hver gang innholdet i en `entity-instance` endrer seg. Denne metainformasjonen er tilgjengelig under rendring av nettsidene, noe som gjør det enkelt å hente ut og presentere når informasjonen ble endret og hvilken bruker som utførte endringen. Et eksempel på hvor dette kan være nyttig kan være i en liste over innlegg i en blogg, hvor man kan for hvert innlegg kan vise når dette ble opprettet og hvilken skribent som har skrevet innlegget.

6.1.3 Views

Elementet `entity-instance` inneholder ingen informasjon om hvordan innholdet skal rendres til HTML av Presentation Viewer. For å angi hvordan informasjonen skal rendres, benytter man elementet `view`. Et `view` er ganske enkelt en kobling mellom en entity `entity-instance` og en transformasjon (filnavnet til et XSLT-dokument, der filtypen *.xslt er utelatt), samt en kort beskrivelse av hva dette `view`-et vil rendre.

Beskrivelsen vil være synlig for administrator i Presentation Manager slik at det er lettere å identifisere de forskjellige `view`-ene uten å måtte se på selve presentasjonsmønsterspesifikasjonen.

Som angitt i figur 6-1 er det mulig å koble flere `views` til en og samme `entity-instance`. Hensikten med dette er at det skal være mulig å rendre denne informasjonen på forskjellige måter i en og samme presentasjon. Eksempel 6-5 viser hvor nyttig denne funksjonaliteten er. Her har vi definert to `views`, det første knytter `messageListEntityInstance` opp mot en transformasjon som kun rendrer de siste 5 meldingene. Det andre `view`-et benytter samme `entity-instance`, men bruker en transformasjon som tar med alle meldingene i innholdsfilen.

```
<views>
    <view id="latestMessagesView">
        <description>Inneholder en liste over de siste 5 meldingene</description>
        <entity-instance>messageListEntityInstance</entity-instance>
        <transformation>latestMessagesTransformer</transformation>
    </view>

    <view id="allMessagesView">
        <description>Inneholder en liste over ALLE meldingene</description>
        <entity-instance>messageListEntityInstance</entity-instance>
        <transformation>allMessagesTransformer</transformation>
    </view>
</views>
```

Eksempel 6-5: Hvordan `views` kan deklareres i presentasjonsmønsterspesifikasjonen

Disse to `view`-ene kan man plassere på forskjellige nettsider i presentasjonen. For eksempel kan man plassere `latestMessageView` på startsiden, slik at brukerne alltid ser de siste meldingene når de kommer til nettstedet, og samtidig plassere `allMessagesView` på en nettside for tidligere meldinger.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" version="1.0" encoding="UTF-8" indent="yes"/>

    <xsl:template match="messageListEntity">
        <h1 class="mainHeading">Siste 5 meldinger</h1>
        <!-- Gå gjennom hver melding i listen -->
        <xsl:for-each select="messages/messageEntity">

            <!-- Sorter elementer etter år, måned, dag og time -->
            <xsl:sort select="substring(@createdOn,7,4)"  order="descending"/>
            <xsl:sort select="substring(@createdOn,4,2)"  order="descending"/>
            <xsl:sort select="substring(@createdOn,1,2)"  order="descending"/>
            <xsl:sort select="substring(@createdOn,12,5)" order="descending"/>

            <!-- Ta med kun de siste 5 meldinger -->
            <xsl:if test="position()&lt;=5">

                <!-- Vis tittel, innhold, publiseringsdato og forfatter -->
                <h3><xsl:value-of select="title"/></h3>
                <p><xsl:value-of select="content" disable-output-escaping="yes" /></p>
                <span class="readmore"><xsl:value-of select="@createdBy" /></span>
                <span class="date"><xsl:value-of select="@createdOn" /></span>

            </xsl:if>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

Eksempel 6-6: Hvordan en transformasjon for et `view` kan se ut

Transformasjoner defineres ved hjelp av XSLT og har som hovedoppgave å transformere innholdet i dokumentene til XHTML. I tillegg tilbyr XSLT funksjonalitet for både sortering og filtrering, noe som gjør det unødvendig å spesifisere dette i presentasjonsmønsterspesifikasjonen, slik det ble foreslått i forslaget til (Rossini og Liberati 2005). Eksempel 6-6 viser hvordan en slik transformasjon kan se ut. Denne transformasjonen sorterer alle meldingene i listen etter opprettelsesdato og filtrerer vekk alle meldingene unntatt de fem nyeste. Resultatet av transformasjonen er en XHTML-snutt, som sammen med CSS dikterer hvordan meldingslisten skal se ut på nettsiden.

6.1.4 Pages

Det siste elementet i presentasjonsmønsterspesifikasjonen er `pages`, og dets formål er å definere de forskjellige nettsidene som en presentasjon skal inneholde. En `page` består i all hovedsak av en `page template` og et eller flere `views`.

```xml
<page id="startPage" default="true">
    <description>Startside med en kort beskrivelse og de viktigste meldingene</description>
    <composition>
        <view alwaysVisible="true">welcomeInfoView</view>
        <view alwaysVisible="true">forumView</view>
        <view defaultView="true">latestMessagesView</view>
        <view>allMessagesView</view>
    </composition>
    <page-template>startPageTemplate</page-template>
</page>
```

Eksempel 6-7: Hvordan et `page`-element kan deklareres i spesifikasjonen

I eksempel 6-7 ser man hvordan et `page`-element kan deklareres i presentasjonsmønsterspesifikasjonen. Hvert element inneholder en kort beskrivelse, som er synlig i Presentation Manager for lettere å identifisere de forskjellige sidene. Deretter følger et `composition`-element som inneholder en liste over alle `views` som skal være på denne siden, samt regler, angitt som attributter, for hvordan de ulike views-ene skal oppføre seg. Til slutt angir man hvilken `page template` som skal benyttes.

Man må alltid angi hvilket av `view`-ene som skal være såkalt `default`, altså hvilket view som skal vises dersom ikke noe annet view er valgt av bruker. Dette gjøres ved å angi attributten `defaultView="true",` og kun ett `view` per `page`-element kan ha denne attributten. I utgangspunktet vil kun ett `view` vises på nettsiden om gangen, avhengig av brukerens forespørsel. Men ved å angi attributten `alwaysVisible="true"` vil et `view` alltid bli rendret når denne siden vises, uavhengig av hvilket `view` bruker har forespurt. Denne funksjonaliteten er nyttig når man har informasjon som alltid skal være synlig på en side. I eksempel 6.7 vil `welcomeInfoView` og `forumView` alltid være synlig på nettsiden, mens bruker kan bytte mellom de to øvrige `view`-ene `latestMessagesView` og `allMessagesView`.

På samme måte som en `entity-instance` kan benyttes i ett eller flere `views`, kan et `view` benyttes i en eller flere pages. Da har man igjen mange av de samme mulighetene ved at man kan plassere et og samme `view` på forskjellige nettsider i en presentasjon.

6.2 Oppbygging av en nettside

I dette avsnittet vil vi gå gjennom de forskjellige byggeklossene som inngår i prosessen rundt rendringen av en nettside (`page`) definert i presentasjonsmønsteret. Forståelsen av de ulike teknologiene som benyttes og hvordan de samarbeider er svært viktig for å forstå hvordan de øvrige delsystemene fungerer. Hvordan selve rendringsprosessen blir implementert i det nye systemet skal vi se på i et senere kapittel.

6.2.1 Page templates

Hovedansvaret til en `page template` er å plassere de ulike `views` som er tilknyttet en bestemt `page`. Ved å ha forskjellig `page template` på hver av nettsidene kan man plassere ett og samme `view` på forskjellige måter, gjerne i forskjellige HTML-elementer med unike CSS-klasser, som kan føre til at bruker ser ett og samme `view` på totalt forskjellige måter. Denne kraftige funksjonaliteten er noe utviklere av presentasjonsmønstre kan dra stor nytte av.

```
<!-- Følgende views skal vises i hovedområdet -->
<div id="mainContentArea">
    #if( $welcomeInfoView )
        $welcomeInfoView
        <hr />
    #end

    #if( $latestMessagesView )
        ${latestMessagesView}
    #end

    #if( $allMessagesView )
        ${allMessagesView}
    #end
</div>

<!-- Følgende views skal plasseres i høyre kolonne på nettsiden -->
<div id="sidebar">
    #if( $forumView )
        ${forumView}
    #end
</div>
```

Eksempel 6-8: Hvordan en `page template` for en `page` kan settes opp

En `page template` for en nettside defineres ved hjelp av Velocity Templates (Velocity Apache Project). Eksempel 6-8 viser hvordan et slikt dokument for en nettside kan se ut. Her plasserer man de ulike `view`-elementene på nettsiden. Hver av variablene, markert med `$`-tegn, svarer til et bestemt `view` på nettsiden. Når dette dokumentet blir tolket av systemet, blir hver av disse variablene byttet ut med XHTML-snutten fra transformasjonen av hver av `view`-ene. Merk bruken av den innebygde `#if`-makroen for å sjekke om view-et er tilgjengelig, da det er mulig at dette viewet er skjult (`disabled`) i Presentation Manager.

I utgangspunktet ble de foreslått å benytte XSLT også her, men man fant fort ut at dette ikke var den rette teknologien å benytte i dette tilfellet. Problemet er at XSLT kun arbeider effektivt med ett XML-tre om gangen, mens man i dette tilfellet sitter med en mengde XHTML-snutter som skal settes sammen til ett stort XHTML-dokument. Velocity Templates er en enkel teknologi som er lett å lære og er ideell for å plassere resultatet av de forskjellige `view`-transformasjonene på nettsiden.

En viktig funksjonalitet som bruken av disse templatene fører med seg, er at utviklerne har muligheter til å bestemme hvordan de forskjellige XHTML-snuttene skal plasseres i forhold til hverandre. Ofte er det for eksempel ønskelig å plassere resultatet av to av `view`-ene på nettsiden i en og samme span- eller div-element, slik at de kan plasseres på en bestemt måte ved hjelp av CSS. Dette gjør det enkelt å sette opp de mest brukte teknikkene for CSS-layout, som for eksempel to eller tre kolonners layout.

6.2.2 Master template

Alle `page`-elementer har sin egen `page template`, som hver er ansvarlig for å plassere resultatet av sine `views`. Men man har fortsatt ikke generert et fullstendig gyldig XHTML-dokument. Man trenger også tagger for <html>, <head>, <body> osv.

For å unngå at man må duplisere denne informasjonen i alle layoutfilene, har alle presentasjoner en såkalt `master template`, også er implementert som en Velocity Template (Se eksempel 6-9). Denne inneholder HTML som skal være på toppen og i bunnen av hver eneste side, uavhengig av hvilken side som er valgt. Her setter man opp strukturen som er felles for alle nettsidene i presentasjonen, og plasserer viktige menyer som `main menu`, `view meny` og `navigation bar` (forklares senere i kapittelet).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
        <meta http-equiv="pragma" content="no-cache"/>
        <meta http-equiv="cache-control" content="no-cache"/>
        <meta http-equiv="expires" content="0"/>
        <title>${presentationTitle}</title>

        <!-- Lenke til presentasjonens stilark blir satt inn her -->
        ${stylesheetLink}
    </head>
    <body>
            <!-- Plassering av navigasjonsbaren -->
            <div id="navigationBar">
                ${navigationBarContent}
            </div>

            <div id="header">
                <!-- Presentasjonstittel og beskrivelse er tilgjengelig for master template -->
                <h1 id="logo">${presentationTitle}</h1>
                <h2 id="slogan">${presentationDescription}</h2>

                <!-- Plassering av hovedmenyen -->
                <div id="mainMenu">
                    ${mainMenuContent}
                </div>
            </div>

            <div id="content">
                <!-- Plassering av view-menyen -->
                <div id="sidebar">
                    ${viewMenuContent}
                </div>

                <!-- Resultatet fra pageTemplate kommer her -->
                <div id="mainPageContents">
                    ${pageContents}
                </div>

                <div id="footer">
                    <center>&copy; 2008, Universitetet i Bergen</center>
                </div>
            </div>
    </body>
</html>
```

Eksempel 6-9: Hvordan en master template kan se ut

6.2.3 Cascading Style Sheet (CSS)

Hvert presentasjonsmønster har også et tilhørende `stylesheet`, eller stilark. CSS (Cascading Style Sheet, W3C), som er det siste leddet i rendringsprosessen. Ved å angi `class`-attributter på HTML-elementene som genereres i transformasjonene eller i Velocity-templatene, kan utviklerne utnytte CSS-teknologien til det fulle. Stilarkene har stor innvirkning på hvordan den endelige nettsiden vil se ut for brukeren, og vil derfor trolig være det komponentet som dem som utvikler nye presentasjoner oftest vil overkjøre. Se eksempel 6-10 for et kort utdrag av et typisk CSS-stilark for et presentasjonsmønster.

```
/* header */
#header {
    position: relative;
    height: 85px;
    background: #000 url( getPatternResource.htm?patternId=coursePatternV2&type=MULTIMEDIA&fileId=headerbg.gif ) repeat-x 0% 100%
}

#header h1#logo {
    position: absolute;
    margin: 0;
    padding: 0;
    font: bolder 4.1em 'Trebuchet MS', Arial, Sans-serif;
    letter-spacing: -2px;
    text-transform: uppercase;
    top: 0;
    left: 5px;
}
```

Eksempel 6-10: Her ser vi et kort utdrag av et CSS-stilark for et presentasjonsmønster

6.2.4 Bruk av Javascript

Det er også fullt mulig å inkludere JavaScript i presentasjonene, alt fra enkle små script på noen få linjer, til omfattende JavaScript-rammeverk. Lenken til disse scriptene legger man til i HTML-headeren i `master template`. Bruk av JavaScript kan bidra til å gjøre nettsidene mer interaktive og brukervennlige.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
        <meta http-equiv="pragma" content="no-cache"/>
        <meta http-equiv="cache-control" content="no-cache"/>
        <meta http-equiv="expires" content="0"/>

        <title>${presentationTitle}</title>

        ${stylesheetLink}

        <script language="javascript" type="text/javascript"
            src="../getPatternResource.htm?patternId=coursePattern&type=JAVASCRIPT&fileId=jquery-1.2.3.js"></script>
        <script language="javascript" type="text/javascript"
            src="../getPatternResource.htm?patternId=coursePattern&type=JAVASCRIPT&fileId=ui.accordion.js"></script>
        <script language="javascript" type="text/javascript"
            src="../getPatternResource.htm?patternId=coursePattern&type=JAVASCRIPT&fileId=jquery.dimensions.js"></script>
        <script language="javascript" type="text/javascript"
            src="../getPatternResource.htm?patternId=coursePattern&type=JAVASCRIPT&fileId=thickbox.js"></script>

    </head>
```

Eksempel 6-11: Utdrag fra headeren til en master template der JavaScript er benyttet

I eksempel 6-11 ser vi hvordan man kan legge inn slike script i headeren til `master template`. Det er ingenting i veien for at en presentasjonsmønsterutvikler kan ta i bruk et AJAX-rammeverk (AJAX, Wikipedia) eller JavaScript-bibliotek, som for eksempel jQuery (jQuery JavaScript Library.) eller Dojo (DOJO Toolkit), og bruke dette til å lage mange forskjellige effekter på nettsidene. Her er det kun kreativiteten som setter grenser.

6.2.5 Menyer

I `master template` må man også plassere tre viktige elementer; `navigation bar`, `main menu` og `view menu`. Disse tre elementene svarer til henholdsvis variablene `$navigationBarContent`, `$mainMenuContent` og `$viewMenuContent` i `master template`, og presentasjonsmønsterutvikler er nødt til å plassere disse for at en presentasjon skal fungere som den skal.

Alle de tre menyene genereres av systemet i XML-format, og det forventes at alle presentasjonsmønstre inneholder tilsvarende XSLT-dokumenter for å transformere disse til XHTML som kan rendres i presentasjonen. På denne måten har en presentasjonsmønsterutvikler full kontroll over hvordan de forskjellige menyene skal vises på nettsidene. Vi vil her gå gjennom hvordan disse tre menyene er bygget opp og hvordan disse rendres.

Navigation bar

Denne menyen inneholder lenker som knytter en presentasjon sammen med resten av systemet. XML-dokumentet som genereres av systemet ser man i eksempel 6-12. Den første lenken, `editor-link`, lar brukere med rollen `publisher` gå direkte til Presentation Content Editor og redigere innholdet i den aktive presentasjonen. Den andre lenken, `change-presentation-link`, lar en bruker gå tilbake til presentasjonsoversikten og dermed muligheten til å velge en annen presentasjon. Den siste lenken, `logout-link`, lar en bruker logge ut av systemet.

```
<navbar-links>
    <editor-link>../pce/content/listContent.htm?pid=inf101h08</editor-link>
    <change-presentation-link>../lobby/presentations.htm</change-presentation-link>
    <logout-link>../j_acegi_logout</logout-link>
</navbar-links>
```

Eksempel 6-12: XML for `navigation bar` generert av systemet

For å transformere denne menyen til XHTML, må alle presentasjonsmønstre inneholder transformasjonsdokumentet `navBarTransformer.xslt` (Se eksempel 6-13). Her har man full frihet til å rendre disse lenkene slik man vil, og er et ideelt sted å plassere forskjellige `id`- eller `class`-attributter som kan dekoreres i CSS. Merk at for å unngå at lenken til Presentation Content Editor vises for brukere som ikke har `publisher`-rettigheter for den aktuelle presentasjonen, bør man bruke en `xsl:if`-tag før denne lenken i XSLT-dokumentet, demonstrert i eksempelet. Her har man valgt å vise lenkene som vanlige XHTML-lenker med en vertikal strek mellom dem.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" version="1.0" encoding="UTF-8" indent="yes"/>

    <xsl:param name="isPublisher" select="'isPublisher'"/>

    <xsl:template match="navbar-links">

        <xsl:if test="$isPublisher='true'">
            <xsl:element name="a">
                <xsl:attribute name="href">
                    <xsl:value-of select="editor-link"/>
                </xsl:attribute>
                Rediger innhold
            </xsl:element>
            |
        </xsl:if>
        <xsl:element name="a">
            <xsl:attribute name="href">
                <xsl:value-of select="change-presentation-link"/>
            </xsl:attribute>
            Bytt presentasjon
        </xsl:element>
        |
        <xsl:element name="a">
            <xsl:attribute name="href">
                <xsl:value-of select="logout-link"/>
            </xsl:attribute>
            Logg av
        </xsl:element>
    </xsl:template>

</xsl:stylesheet>
```

Eksempel 6-13: XSLT-dokumentet `navBarTransformer` transformerer menyen

Main menu

Denne menyen inneholder lenker til de forskjellige `pages` som finnes i presentasjonen, og er den viktigste menyen for å navigere en presentasjon. Eksempel 6-14 viser hvordan det genererte XML-dokumentet for en presentasjon med tre `pages` kan se ut. Hvert element i listen har en `url`-attributt som inneholder lenken til den aktuelle siden, og verdien til hvert element er tittelen på siden. I tillegg blir den aktive nettsiden, den som er valgt for øyeblikket, markert med en `selected`-attributt. Merk at `pages` som er deaktiverte er ikke med i denne listen (mer om deaktivering senere i kapittelet). Dette gjør det mulig å rendre lenken til den aktive nettsiden på en annen måte enn de øvrige lenkene.

```
<pages>
    <page selected="true" url="presentation.htm?pid=inf101h08&amp;amp;page=startPage">Startsiden</page>
    <page url="presentation.htm?pid=inf101h08&amp;amp;page=progressPlanPage">Fremdriftsplan</page>
    <page url="presentation.htm?pid=inf101h08&amp;amp;page=courseInfoPage">Kursinformasjon</page>
</pages>
```

Eksempel 6-14: XML for `main menu` generert av systemet

I eksempel 6-15 ser vi hvordan et transformasjonsdokument for `main menu` for kan se ut. Her blir lenkene rendret som en enkel unummerert liste. Den aktive nettsiden blir markert med en `id`-attributten satt til `current`, som igjen kan fanges opp av CSS og

rendre denne lenken annerledes. Igjen har presentasjonsmønsterutvikler full kontroll over hvordan disse lenkene skal rendres.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" version="1.0" encoding="UTF-8" indent="yes"/>

    <xsl:template match="pages">
        <ul>
            <xsl:apply-templates/>
        </ul>
    </xsl:template>

    <xsl:template match="page[@selected='true']">

        <xsl:element name="li">
            <xsl:attribute name="id">current</xsl:attribute>
            <xsl:element name="a">
                <xsl:attribute name="href">
                    <xsl:value-of select="@url"/>
                </xsl:attribute>
                <span>
                    <xsl:value-of select="."/>
                </span>
            </xsl:element>
        </xsl:element>

    </xsl:template>

    <xsl:template match="page">
        <xsl:element name="li">
            <xsl:element name="a">
                <xsl:attribute name="href">
                    <xsl:value-of select="@url"/>
                </xsl:attribute>
                <span>
                    <xsl:value-of select="."/>
                </span>
            </xsl:element>
        </xsl:element>
    </xsl:template>

</xsl:stylesheet>
```

Eksempel 6-15: XSLT-dokumentet `mainMenuTransformer` transformerer hovedmenyen

View menu

Den siste menyen inneholder lenker til `views` for den aktive nettsiden, og på den måten kan en bruker bla gjennom de forskjellige `view`-ene. Merk at views som har blitt markert som `alwaysVisible` eller er deaktiverte (mer om deaktivering senere i kapittelet) ikke er med på denne menyen. XML-dokumentet som genereres (se eksempel 6-16) har samme struktur som `main menu`.

```
<views>
    <view selected="true"
          url="presentation.htm?pid=inf101h08&amp;amp;page=startPage&amp;amp;view=latestMessagesView">Siste m
    <view url="presentation.htm?pid=inf101h08&amp;amp;page=startPage&amp;amp;view=allMessagesView">Meldingsar
</views>
```

Eksempel 6-16: XML for `view menu` generert av systemet

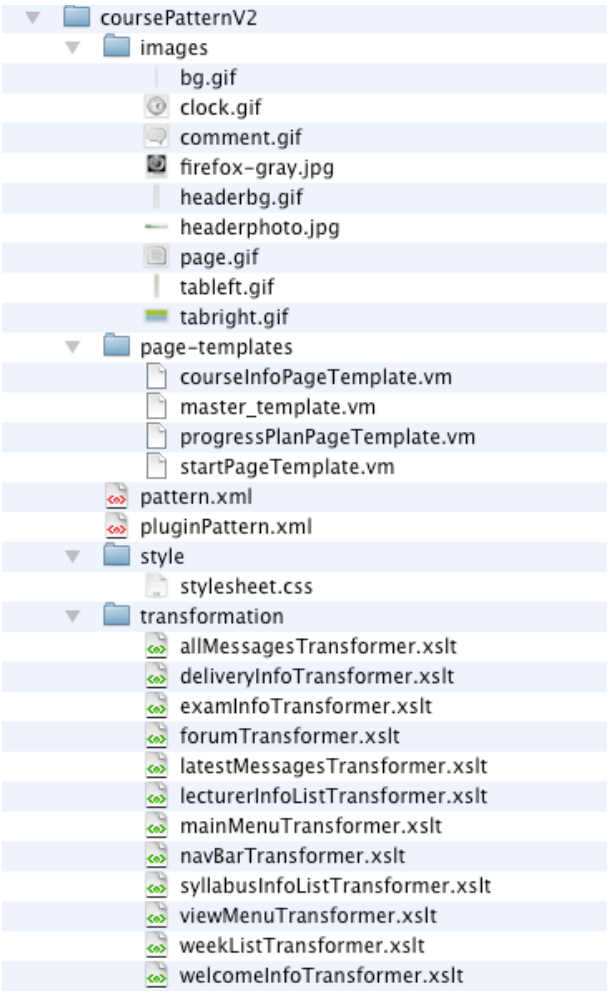
Et eksempel på en transformasjonsfil for denne menyen ser man i eksempel 6-17. Denne har veldig lik struktur som transformasjonsfilen til `main menu`, men med andre CSS-attributter som gjør at den vil se veldig annerledes ut når den vises på nettsiden.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" version="1.0" encoding="UTF-8" indent="yes"/>

    <xsl:template match="views">
        <h1>Meny</h1>
        <ul class="sidemenu">
            <xsl:apply-templates/>
        </ul>
    </xsl:template>

    <xsl:template match="view[@selected='true']">
        <li>
            <xsl:element name="a">
                <xsl:attribute name="href">
                    <xsl:value-of select="@url"/>
                </xsl:attribute>
                <xsl:attribute name="class">selectedView</xsl:attribute>
                <xsl:value-of select="."/>
            </xsl:element>
        </li>
    </xsl:template>

    <xsl:template match="view">
        <li>
            <xsl:element name="a">
                <xsl:attribute name="href">
                    <xsl:value-of select="@url"/>
                </xsl:attribute>
                <xsl:value-of select="."/>
            </xsl:element>
        </li>
    </xsl:template>

</xsl:stylesheet>
```

Eksempel 6-17: XSLT-dokument `viewMenuTransformer` transformerer `view menu`

I figur 6-3 ser man en oversikt over de forskjellige filene som inngår i et presentasjonsmønster. I tillegg til selve presentasjonsmønsterspesifikasjonen, lagret i `pattern.xml`, har man mappene `images`, `page templates`, `style`, `resources` og `transformation`. Bildemappen inneholder som regel bilder som hører til CSS-stilarket og som brukes av `master template` (for visning av bannere, bakgrunnsbilder, logoer, og lignende).

Alle `page templates`, inkludert `master template`, har fått sin egen mappe, og det samme har CSS-stilarket. Den siste mappen inneholder alle transformasjoner, en for hver av `views`-ene, i tillegg til transformasjonsdokumentene for de tre menyene. XML-dokumentet `pluginPattern.xml` brukes for å konfigurere ulike multimediaplugins i presentasjonsmønsteret, og en nærmere beskrivelse av dette finner man i (Ingvaldsen 2008)



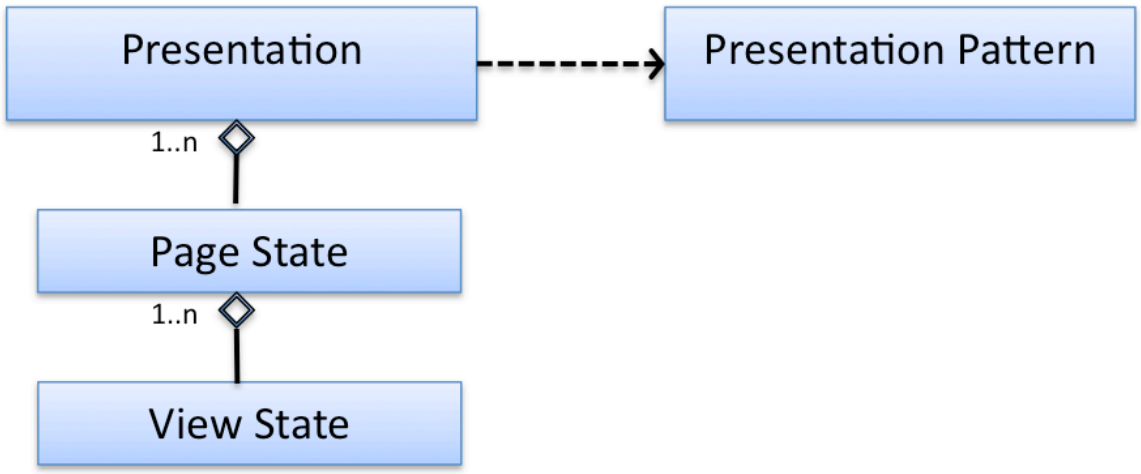
Figur 6-3: Oversikt over de forskjellige filene i en presentasjonsmønsterspesifikasjon

6.3 Den nye presentasjonsspesifikasjonen

Den nye presentasjonsspesifikasjonen inneholder metainformasjon knyttet til instanser av presentasjonsmønstre, altså såkalte *presentasjoner*. I motsetning til presentasjonsspesifikasjonen man finner i forslaget til (Rossini og Liberati 2005) er det ikke lenger behov for manuell opprettelse og redigering av innhold i denne.

En viktig målsetning for det nye systemet er at det skal være svært enkelt å opprette nye presentasjoner basert på eksisterende mønstre. Alt man trenger av spesifikasjonsdokumenter, transformasjoner og HTML-dokument er allerede tilgjengelig som en del av presentasjonsmønsteret. Som et resultat av dette er presentasjonsspesifikasjonen mindre omfattende en presentasjonsmønsterspesifikasjonen. Den består av følgende to hovedelementer:

- Page State
- View State



Figur 6-4: Oversikt over de ulike elementene i presentasjonsspesifikasjonen

Oversikt over de ulike elementene og forholdet mellom disse er illustrert i figur 6-4. En presentasjon består av en eller flere `page state`-elementer, ett element for hver `page` i presentasjonsmønsterspesifikasjonen. Tilsvarende finner man et eller flere `view state`-element for hvert `view` i henhold til presentasjonsmønsterspesifikasjonen. I tillegg til disse elementene har spesifikasjonen en del enkle elementer, som `presentation-pattern`, `title`, `description` og `user-group-id`.

I eksempel 6-18 ser vi hvordan en presentasjonsspesifikasjon kan se ut. I dette tilfelle benytter man et mønster som består av to `pages` som hver har en `views`, og presentasjonsspesifikasjonen holder tilstandsinformasjon om disse. Det første man angir er hvilken presentasjonsmønsterspesifikasjon denne presentasjonen skal benytte seg av. Deretter kommer en tittel og beskrivelse av presentasjonen. I elementet `user-group-id` er det angir identifikatoren til en gruppe med de brukere som skal ha tilgang til denne presentasjonen. Hvordan brukerhåndtering og sikkerhet blir håndtert i det nye systemet kan man lese i detalj om i (Løvik 2008).

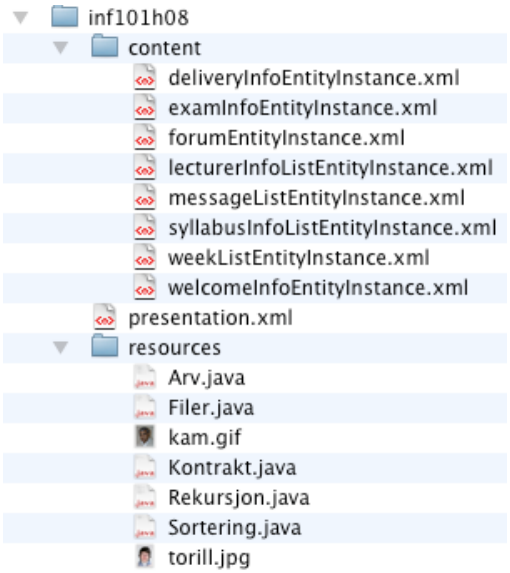
Deretter defineres tilstanden til de forskjellige `pages` og `views` som er angitt i presentasjonsmønsterspesifikasjonen. For hver page defineres elementet `page-state` sammen med en rekke attributter, deriblant attributten `id` som angir hvilken page definert i presentasjonsmønsterspesifikasjonen det er snakk om. Dersom det er enkelte nettsider i mønsteret som det ikke er behov for, kan disse deaktiveres ved at man setter attributten `enabled` til `false`. Da vil nettsiden ikke være synlig for de vanlige brukerne.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<specification>

    <!-- Angir hvilket mønster som skal benyttes -->
    <presentation-pattern>coursePattern</presentation-pattern>

    <!-- Tittel, beskrivelse og brukergruppe til presentasjonen -->
    <title>INF101</title>
    <description>Videregående programming i Java</description>
    <user-group-id>5</user-group-id>

    <!-- Tilstanden til de forskjellige nettsidene i presentasjonen -->
    <presentation-state>

        <!-- Tilstanden til startPage og alle views som er assosisert med denne siden -->
        <page-state id="startPage" enabled="true" default="true" label="Startsiden">
            <view-state enabled="true" label="Informasjon">welcomeInfoView</view-state>
            <view-state enabled="true" label="Siste meldinger">latestMessagesView</view-state>
            <view-state enabled="true" label="Meldingsarkiv">allMessagesView</view-state>
            <view-state enabled="true" label="Foruminformasjon">forumView</view-state>
        </page-state>

        <!-- Tilstanden til courseInfoPage og alle views som er assosisert med denne siden -->
        <page-state id="courseInfoPage" enabled="true" label="Kursinformasjon">
            <view-state enabled="true" label="Pensum">syllabusInfoListView</view-state>
            <view-state enabled="true" label="Eksamen">examInfoView</view-state>
            <view-state enabled="false" label="Innlevering">deliveryInfoView</view-state>
            <view-state enabled="true" label="Kursansvarlige">lecturerInfoListView</view-state>
        </page-state>

    </presentation-state>

</specification>
```

Eksempel 6-18: Her ser vi hvordan en presentasjonsspesifikasjon kan se ut

Hvilken nettside som skal være såkalt `default`, altså den siden som først møter brukerne rett etter pålogging, er allerede angitt i presentasjonsmønsterspesifikasjonen. Men denne innstillingen er det mulig å overstyre når man oppretter presentasjoner av mønsteret ved å angi attributten `default="true"`. Dette kan være en svært nyttig og i enkelte tilfeller nødvendig funksjonalitet, spesielt dersom man velger å slå av nettsiden som er satt som `default` i presentasjonsmønsterspesifikasjonen.

Den siste attributten som må angis er `label`, og teksten i denne vil dukke opp som navn på lenker til de forskjellige nettsiden i hovedmenyen. I utgangspunktet kunne disse navnene blitt angitt i presentasjonsmønsterspesifikasjonen fremfor i presentasjonsspesifikasjonen. Men dette ville være like fleksibelt fordi man da låser mønsteret til ett bestemt språk, et problem man unngår ved å angi lenketeksten for hver enkelt presentasjonsinstans. Har man for eksempel utviklet et presentasjonsmønster for nettkurs, er det da mulig å benytte samme presentasjonsmønsterspesifikasjon for presentasjoner på både norsk og engelsk.



Figur 6-5: Oversikt over de forskjellige filene tilhørende en presentasjon

I figur 6-5 ser man en oversikt over de forskjellige filene som inngår i en presentasjon. I tillegg til selve presentasjonsspesifikasjonen, `presentation.xml`, har man innholdsfiler (`content`) og ressurser (`resources`). For hver `entity instance` i presentasjonsmønsteret har man et tilsvarende innholdsdokument i XML-format. Ressursmappen inneholder alle filer som lastes opp til presentasjonen.

6.3.1 Overkjøring av presentasjonsmønsterspesifikasjonen

En svært nyttig funksjonalitet som den nye implementasjonen av presentasjonsmønstre tillater, er at presentasjoner kan velge å *overkjøre* forskjellige komponenter som er den del av presentasjonsmønsterspesifikasjonen. Dersom man ikke er tilfreds med måten en del av disse komponentene, kan man hente ut en kopi av disse, gjøre de nødvendige endringene man ønsker, og lagre disse i presentasjonen. Når Presentation Viewer laster inn presentasjonen, vil den velge de endrede komponentene fremfor de som ligger i mønsterspesifikasjonen.

Dette vil kunne redusere det såkalte dupliseringsproblemet som DPG 1.0 var så plaget av, der en presentasjonsutvikler har funnet en presentasjonsmønsterspesifikasjon som løser 80-90% av det man ønsker, men man er ikke fornøyd med de siste 10-20%. Kanskje er man ikke helt fornøyd med måten menyene er lagt opp, og man ønsker å ha en spesiell logo øverst på hver nettside. Da ender man ofte opp med å duplisere *hele* presentasjonsmønsterspesifikasjonen, for så å gjøre de små endringene som skal til. Da sitter man plutselig med to nesten identiske spesifikasjoner, noe som øker vedlikeholdsmengden og skaper forvirring for nye utviklere som ønsker å benytte en av disse to spesifikasjonene, men vet ikke nøyaktig hvilken. Slik duplisering er kun fornuftig når det er *markant forskjell* mellom spesifikasjonene. Ved å tilby overkjøring av komponenter i spesifikasjonen vil man kunne unngå dette.

Komponentene som kan overkjøres er `transformations` (XSLT-dokument), `page templates` og `master templates` (Velocity Templates) og `stylesheets` (CSS). Informasjon om hvilke komponenter som er overkjørt for de ulike presentasjonene er lagret direkte i persistenslaget, og krever ingen konfigurasjon i XML-dokumentet for presentasjonsspesifikasjonen. Mer om hvordan overkjøring foregår vil bli diskutert i kapittelet om Presentation Viewer og Presentation Manager.

6.4 Hva som er oppnådd

I den nye implementasjonen av presentasjonsmønstre har vi løst mange av de problemer som har blitt diskutert i de tidligere analysene av DPG 1.0 og i (Rossini og Liberati 2005). Man har også introdusert helt nye konsepter og teknologier, som åpner for mange nye og spennende muligheter som mønsterutviklerne kan dra stor nytte av. Her følger en kort oppsummering av de viktigste fordelene med den nye implementasjonen.

6.4.1 Bedre utnyttelse av teknologier

Personer som ønsker å utvikle nye eller modifisere eksisterende presentasjonsmønstre må ha kjennskap til en rekke webteknologier, blant annet XHTML, XML, XSLT, CSS, og grunnleggende kunnskaper om Velocity Templates. Selv om dette kan virke omfattende, er det lite sannsynlig at det blir et problem, da dette er velkjente teknologier som de fleste webutviklere er komfortable med. Og her ligger en av styrkene til det nye mønsteret, nemlig det at man bruker velkjente og velprøvde teknologier, og utnytter de muligheter som de tilbyr. Samspillet mellom de forskjellige teknologiene gjør at presentasjonsmønsterutviklerne har mye større frihet i hvordan man designer nettsidene enn det som var mulig tidligere.

6.4.2 Mer tydeligere og veldefinerte spesifikasjoner

Strukturen på presentasjonsmønster- og presentasjonspesifikasjonen er kraftig forbedret, og det er mye mer tydelig hvilken hensikt med de forskjellige elementene (`entities`, `views`, `pages`, osv.) i spesifikasjonene har. Dette vil gjøre det lettere for presentasjonsmønsterutviklere å lese og forstå presentasjonsmønstre utviklet av andre, samt selv utvikle nye mønstre. Mulighetene for overkjøring av komponentene man finner i presentasjonsmønsterspesifikasjonen vil også redusere problemet med duplisering av nesten like presentasjonsmønstre. Mengden XML som må skrives er også redusert, og det er ikke lenger noe behov for XML Schemas.

6.4.3 Enklere å opprette nye presentasjoner

Som en bonus er det ikke lenger nødvendig å sette opp presentasjonsspesifikasjonen manuelt, da denne inneholder detaljer som DPG-systemet selv kan generere. Dette kan sees på som en videreføring av ideen om spesifikasjon og instansiering fra (Rossini og Liberati 2005), og man trenger ikke lenger være kvalifisert utvikler for å opprette presentasjoner fra mønstre. Systemet kan enkelt generere nye presentasjoner, og kjennskap til de forskjellige teknologiene er kun nødvendig i tilfeller der man ønsker å overkjøre komponenter fra presentasjonsmønsterspesifikasjonen.

6.4.4 Integrert persistensløsning

Det nye DPG-systemet vil ha sitt eget persistenslag, som inneholder all data som lagres i presentasjonene, noe som eliminerer behovet for manuell mapping mellom presentasjonsmønstre og ulike eksterne datakilder (og dermed også problemer og utfordringer knyttet til disse). I den nye presentasjonsmønsterspesifikasjonen har utviklerne full kontroll på strukturen på informasjonen som lagres, og de trenger ikke tenke på hvordan informasjonen faktisk lagres. Spesifikasjonen tilbyr de mest vanligste datastrukturene som trengs på nettsider (blant annet strenger, datoer, XHTML-felt og filer) i form av en enkel og intuitiv syntaks basert på `entities`, `fields`, og `entity-instances`. Det er relativt enkelt å legge til flere entitetstyper dersom det skulle være behov for dette i fremtiden. Dette involverer utvidelse av en abstrakt klasse, samt noen små endringer i koden som håndterer hver av felttypene i motoren.

7 Presentation Viewer (PV)

Hovedoppgaven til Presentation Viewer, selve rendringsmotoren i systemet, er å rendre presentasjoner og gjøre dem tilgjengelig for brukergruppen `readers`. Rent konseptuelt fungerer den på samme måte som DPE-en i DPG 1.0 (Espelid 2004), men selve rendringsprosessen er selvsagt tilpasset den nye implementasjonen av presentasjonsmønstre. I dette kapittelet vil vi se på hvordan Presentation Viewer gjennomfører rendringen, hva de viktigste komponentene i denne prosessen er og hvordan disse samarbeider. Vi vil også se på et mindre delsystem, kalt lobby. Til slutt følger en kort oversikt over de viktigste egenskapene til Presentation Viewer og hvilke forbedringer som er gjort i forhold til presentasjonsmotoren i DPG 1.0.

7.1 Oversikt over de viktigste komponentene

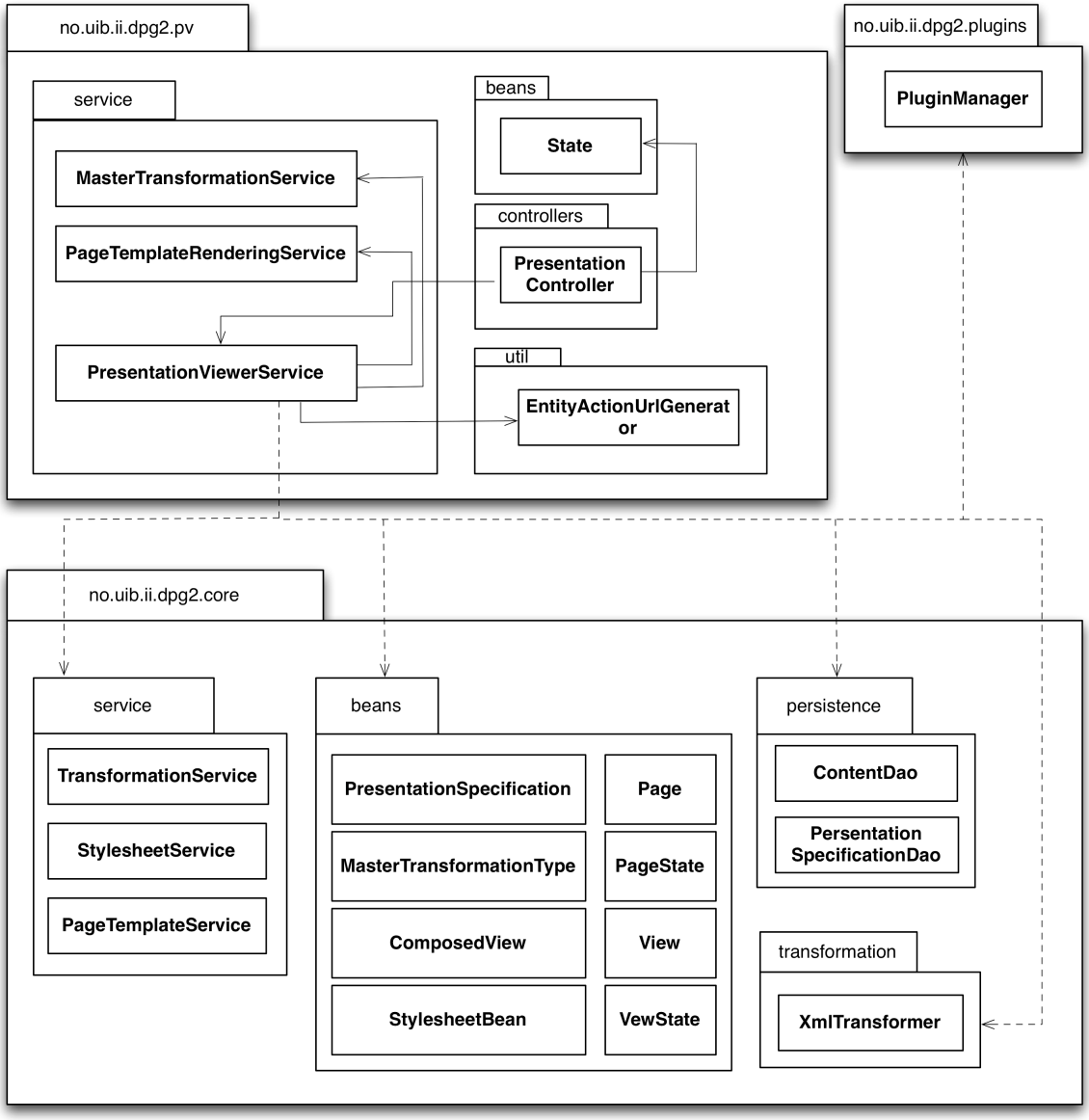
I figur 7-1 ser vi en oversikt over de viktigste pakkene og klassene til Presentation Viewer. Hver av de tre delsystemene (Presentation Viewer, Presentation Content Editor og Presentation Manager) er delt inn i hver sine pakker, og klassene til Presentation Viewer er `no.uib.ii.dpg2.pv`. Internt er denne pakken igjen delt inn i mindre underpakker; `controllers`, `service`, `beans`, og `util`.

Presentation Viewer har kun en eneste controller-klasse i web-laget. Det er denne klassen som tar i mot HTTP-forespørsler fra brukerens nettleser og undersøker hvilken nettside som bruker spør etter ved å se på eventuelle parametere som er vedlagt forespørselen. Disse parameterne blir innkapslet i en enkel bean kalt `State` fra `beans`-pakken. Denne klassen utfører ikke selve rendringen selv, da dens hovedoppgave er prosessering av http-forespørsler.

I stedet delegeres dette arbeidet til `PresentationViewerService` i service-pakken. Denne klassen har det overordnete ansvaret for rendringsprosessen, og det er her man finner algoritmen som Presentation Viewer baserer seg på. Vi skal se nærmere på denne algoritmen senere i dette kapitelet. Alle service-klassene er plassert bak et interface slik at man kan bytte ut implementasjonen av hele eller deler av algoritmen uten at det kreves endringer hos klassene som benytter disse. Dette vil nok være en stor fordel når DPG 2 videreutvikles i årene som kommer.

Enkelte presentasjonsmønstre benytter seg av ulike `plugins`, utvidelsesmoduler som legger til ekstra funksjonalitet utover det som allerede er støttet i DPG 2. Derfor må `PresentationViewerService` ha en relasjon til klassen `PluginManager`, som har oversikt over hvilke plugins som er installert og sørger for at de blir kjørt i henhold til spesifikasjonen. Hvordan plugins håndteres i DPG 2.0 kan man lese mer om i (Ingvaldsen 2008).

`PresentationViewerService` vil, i de tilfellene presentasjonsmønsteret ønsker en tettere integrasjon med Presentation Content Editor, også benytte hjelpeklassen `EntityActionUrlGenerator`. Denne klassen er plassert i pakken `util` og vi vil se nærmere på denne senere i dette kapitelet.



Figur 7-1: Oversikt over de viktigste pakkene og klassene i Presentation Viewer

`PresentationViewerService` har flere avhengigheter til klasser plassert i pakken `core`, som består av felles pakker og klasser som benyttes av alle de tre delsystemene. Igjen er avhengighetene mellom pakkene implementert ved hjelp av `interfaces`, slik at lagene kan byttes ut med andre implementasjoner etter behov.

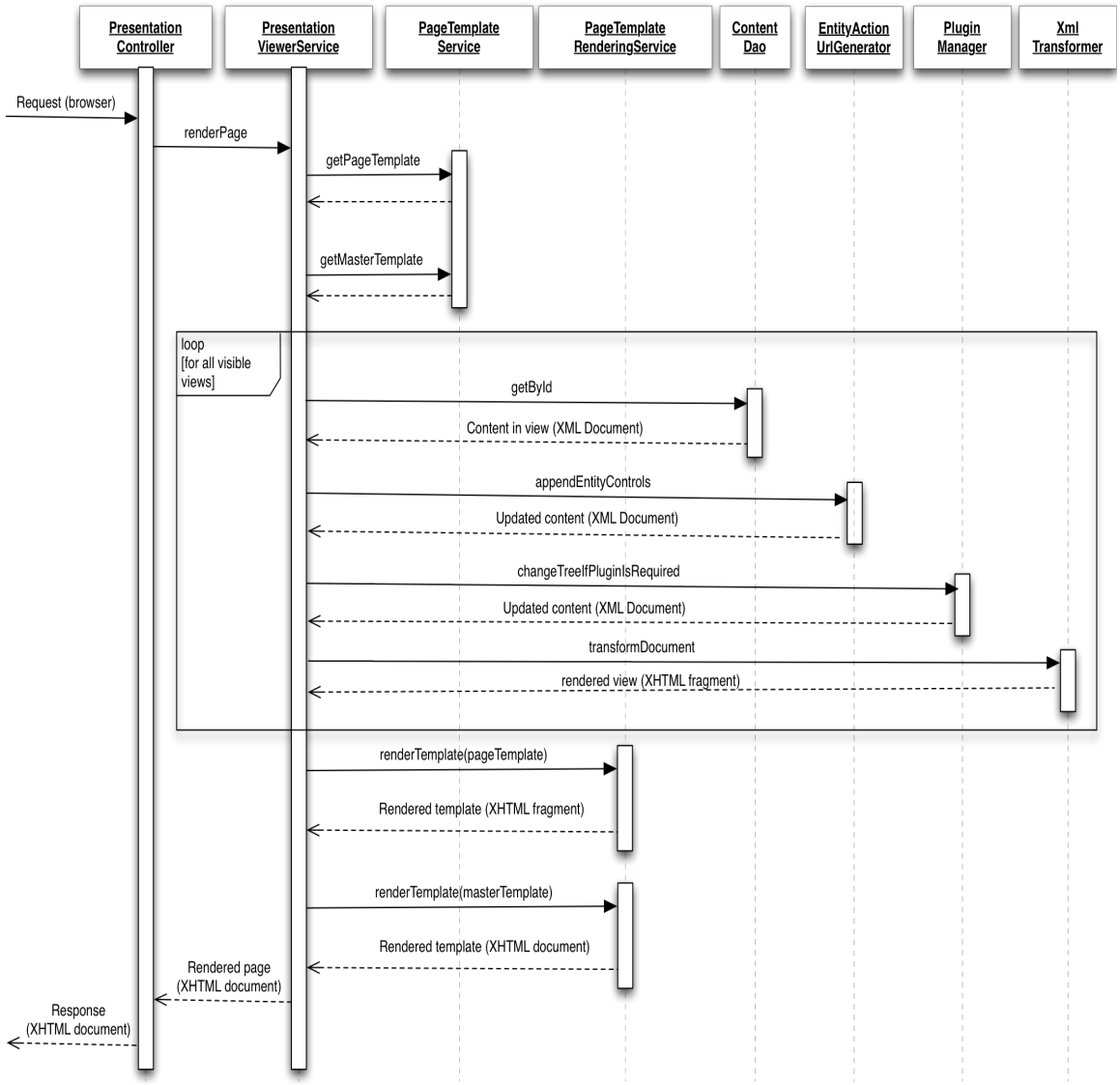
Som vi ser av diagrammet, brukes blant annet flere ulike klasser i service-pakken for å hente ut de forskjellige komponentene (`page templates`, `transformations`, `stylesheets`) som et presentasjonsmønster består av. Selve innholdet til de forskjellige `views` hentes fra `ContentDao` i persistence-pakken, før alt transformeres til HTML ved hjelp av hjelpeklassen `XmlTransformer` i `transformation`-pakken.

7.2 Gjennomgang av rendringsprosessen

Vi skal nå se på hvordan selve rendringsprosessen fungerer. Merk at for å forstå hvorfor rendringsprosessen er slik den er, er det viktig å se på hvordan det nye presentasjonsmønsteret er implementert kapittel 6 i denne oppgaven.

7.2.1 Oversikt over prosessen

I sekvensdiagrammet (figur 7-2) har vi valgt ut de mest sentrale klassene fra pakkediagrammet og fokusert på hvordan Presentation Viewer tar i mot en forespørsel fra bruker, prosesserer denne og returnerer en ferdig rendret HTML-side. Dette diagrammet gir et godt bilde av hvilken algoritme som ligger bak rendringen av nettsidene, og hvilke klasser som utgjør de viktigste komponentene av denne algoritmen.



Figur 7-2: Dette sekvensdiagrammet viser den viktigste prosessen i Presentation Viewer

Det hele begynner med at klassen `PresentationController` får inn en http-forespørsel fra brukerens nettleser. Denne forespørselen inneholder, som minimum, en identifikator som angir hvilken presentasjon som brukeren ønsker tilgang til. Hvis ikke denne er satt, kan ikke Presentation Viewer rendre noe, og brukeren vil i stedet bli vist en liste over tilgjengelige presentasjoner å velge mellom. HTTP-forespørselen kan også inneholde identifikatoren til hvilken `page` og hvilket `view` som skal rendres, men dette er ikke påkrevd. Hvis kun presentasjonsidentifikatoren er satt, vil Presentation Viewer finne ut hvilken `page` som er satt som *default*, altså hvilken side som skal vises dersom ingen spesifikk `page` er angitt i forespørselen. Det samme vil skje for hvilket `view` som skal vises på den valgte `page`, dersom dette heller ikke er angitt i forespørselen.

Dersom forespørselen skulle inneholde ugyldig data, for eksempel at bruker vil se et `view` som ikke hører til den angitte `page`, eller man forsøker å få rendret et `view` eller en `page` som er deaktivert i Presentation Manager, vil man i stedet få opp en feilmelding der brukeren får forklart hvorfor forespørselen ikke ble godtatt. Grensesnittet i DPG 2.0 vil ta hensyn til hvilke deler av presentasjonene som er deaktivert, og vil ikke vise lenker til disse, slik at de vanlige brukerne vil trolig ikke få opp slike feilmeldinger, så fremt de ikke tukler med http-parameterne manuelt.

Når først en forespørsel er mottatt og validert av `PresentationController`, sendes parameterne i forespørselen videre til `PresentationViewerService`. Som beskrevet tidligere, har denne klassen det overordnete ansvaret for selve algoritmen som utfører rendringen av nettsidene. Etter å ha funnet frem mønsterspesifikasjonen til presentasjonen den skal rendre, sender den forespørsler til klassen `PageTemplateService` for å få tak i de nødvendige `page templates`, først til den `page` som skal rendres, deretter til `master template` for hele presentasjonen. Disse to templatene vil bli brukt for å rendre det endelige HTML-dokumentet som skal returneres til brukeren.

Deretter vil klassen finne ut hvilke `views` på den aktuelle siden som skal rendres. Her må det tas hensyn til hvilke `views` som er deaktivert og sørge for at de ikke blir rendret. Hvert aktivt `view` rendres så en etter en. Det første som gjøres inne i løkken er å få tak i innholdsdokumentet til den `entity-instance` som `view`-et er assosiert med i henhold til spesifikasjonen. Klassen `ContentDao` vil da utføre spørringer mot persistenslaget og returnere dette innholdet i form av et XML-dokument.

`PresentationViewerService` vil også gi `EntityActionUrlGenerator` muligheten til å endre på innholdsdokumentet i den aktuelle presentasjonen. Denne klassen vil kun ha noen effekt for presentasjoner som benytter seg av funksjonaliteten for en tettere integrasjon med delsystemet Presentation Content Editor. Hva dette går ut på blir forklart i kapittel 8, som er dedikert til dette delsystemet.

Før innholdsdokumentet transformeres til HTML, vil `PluginManager` få muligheten til å kjøre eventuelle plugins som er assosiert med den aktuelle presentasjonen. `PluginManager` blir oversendt hele innholdsdokumentet og vil se på hvilke plugins

som er installert, og kjøre dem i tur og orden. Et plugin har da muligheten til å oppdatere innholdsdokumentet, for eksempel legge inn XHTML-elementer for multimedia eller lignende. Mer om hvordan `PluginManager` og plugins fungerer i praksis kan leses i (Ingvaldsen 2008).

Til slutt vil innholdsdokumentet transformeres fra XML til XHTML ved hjelp av klassen XmlTransformer. XSLT-dokumentet som benyttes til transformasjonen hentes fra `transformation`-elementet i presentasjonsmønsterspesifikasjonen for det aktuelle `view`-et. Når dette er utført, har man en liten bit av det endelige XHTML-dokumentet (et såkalt *XHTML fragment*), som `PresentationViewerService` lagrer midlertidig.

Når så alle `views` er ferdig transformert, vil `PresentationViewerService` kontakte `PageTemplateService` for å få rendret `page`-elementet, ved hjelp av den `page template` som ble hentet tidligere. Alle XHTML-fragmentene fra rendringen av hvert `view` vil bli sendt med som parameter til denne operasjonen. Resultatet av denne prosessen er et nytt, større XHTML-fragment bestående av innholdet i hvert view, plassert i XHTML-fragmentet slik som det er angitt i den aktuelle `page template`.

Deretter `PageTemplateService` kontaktet på nytt, denne gangen for å rendre den endelige nettsiden, ved hjelp av `master template`. Her blir de forskjellige XHTML-fragmentene, deriblant de tre menyene (`navigation bar`, `main menu` og `view menu`), samt selve sideinnholdet (XHTML-fragmentet som den forrige operasjonen returnerte) plassert på nettsiden, i henhold til det som er angitt i `master template`.

7.2.2 Resultatet av rendringsprosessen

Resultatet av denne operasjonen er et fullstendig XHTML-dokument som returneres til brukeren og vises i brukerens nettleser. Figur 7-3 viser et eksempel på hva som til slutt kan bli visst i brukerens nettleser. Her har utvikleren av presentasjonsmønsterspesifikasjonen brukt en del tid på utseende, og utarbeidet et CSS-stilark sammen med litt grafikk slik at presentasjonen skal få et mer profesjonelt utseende. Helt øverst på nettsiden, i midten, er `navigation bar` plassert. Deretter kommer en overskrift, plassert i `master template`, og ved siden av der igjen er `main menu` plassert. I venstre kolonne finner vi `view menu`, som for øyeblikket lar bruker velge mellom to forskjellige `views` på siden. I midten og i høyre kolonne finner vi resultatet av tre views; i dette tilfelle `welcomeInfoView`, `latestMessagesView` og `forumInfoView`.

Her ser man fordelen ved at hver `page` har sin egen layout, nemlig at man kan plassere hvert enkelt view på nettsiden akkurat hvor man vil. I dette tilfelle er `forumInfoView` plassert i høyre kolonne. Dette `view`-et vil alltid være synlig i denne kolonnen, avhengig av hvilke andre views som er valgt, fordi den har attributten *alwaysVisible* satt til *true* i denne presentasjonen. Brukeren kan derfor kun bla mellom `view`-ene `latestMessagesView` (med attributtet *label* satt til ”Siste meldinger”) og `allMessagesView` (med attributtet *label* satt til ”Meldingsarkiv”).

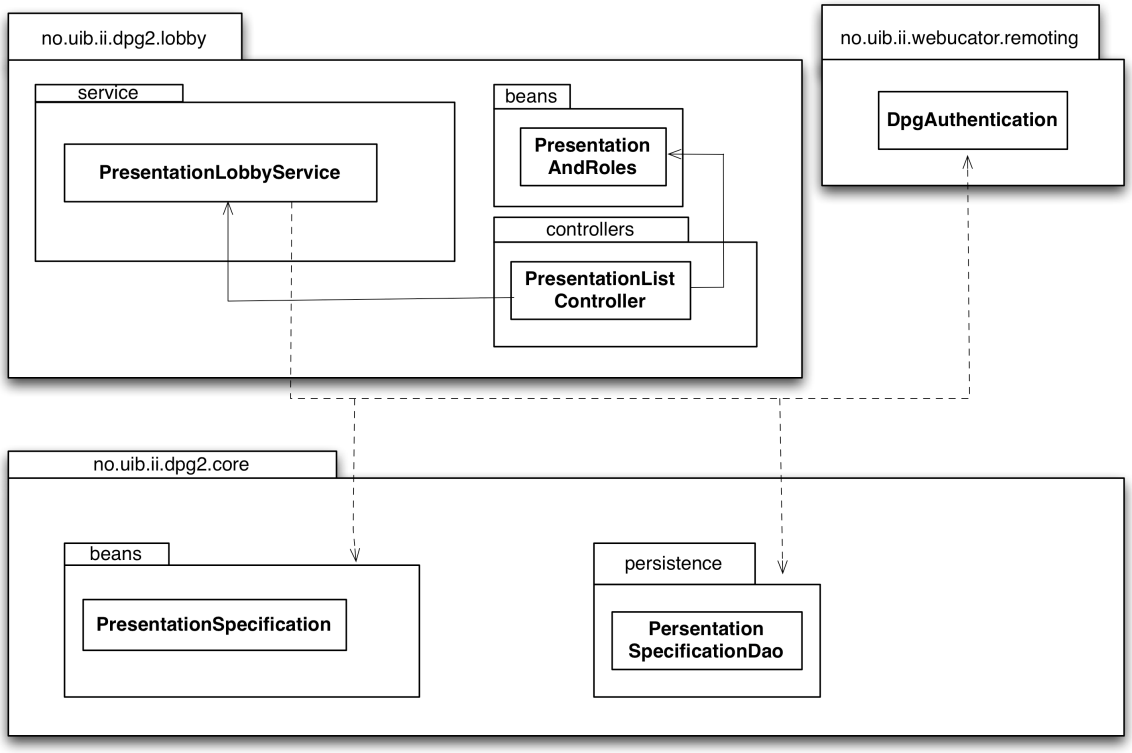
`welcomeInfoView` er også satt som *alwaysVisible*, og vil derfor alltid bli visst øverst uavhengig av hvilket av de to øvrige `views` som bruker har valgt.

Figur 7-3: Hvordan en nettside i en presentasjon kan se ut i Presentation Viewer

7.3 Lobby

Nå som det første av de tre viktigste delsystemene er gjennomgått, kan det være fornuftig å se på det noe mindre delsystemet som knytter disse tre sammen. Som det ble nevnt tidligere i oppgaven, vil man hindre at brukeren opplever at DPG 2.0 består av tre tilsynelatende separate systemer. Det er bedre for brukeropplevelsen dersom DPG 2.0 føles som ett sammenhengene system som hjelper brukeren i å navigere mellom de underliggende tre delsystemene.

Derfor har man valgt å legge til et mindre delsystem, kalt lobby, som fungerer som binneleddet mellom de tre delsystemene, og er som regel det første som møter brukeren når man entrer systemet. Lobby består av en påloggingsside, som sørger for at kun autoriserte brukere får tilgang til systemet, samt en oversiktside som viser alle presentasjonene en bruker har tilgang til. En oversikt over de viktigste pakkene og klassene som er involvert finner man i figur 7-4. Som i de øvrige delsystemene, er det brukt interfaces for å gjøre det lettere å dele opp systemet og skifte ut implementasjoner etter behov. Legg spesielt merke til klassen `DpgAuthentication` som danner grunnlaget for kommunikasjonen med det selvstendige Webucator 3.0.

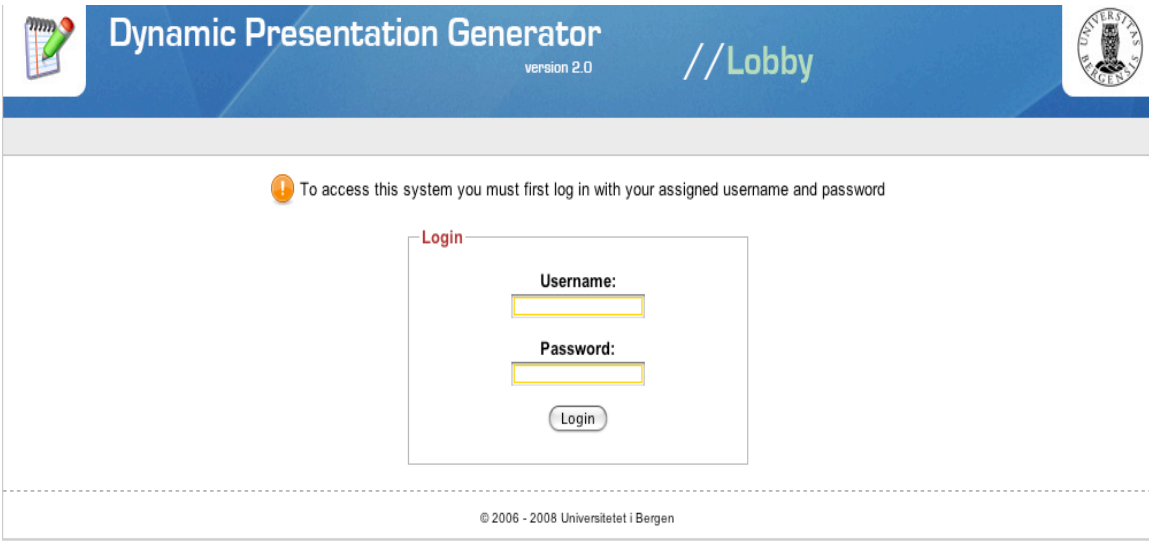


Figur 7-4: Oversikt over de viktigste pakkene og klassene for lobby

Pakken lobby er strukturert på tilsvarende måte som de andre tre delsystemene, dvs. delt inn i controllers, service og beans. I controllers-pakken finner vi `PresentationListController`, som har som oppgave å finne ut hvilke presentasjoner som brukeren har tilgang til og sende denne listen videre til JSP-siden som viser denne for bruker. Her må det tas hensyn til hvilke rettigheter brukeren har, slik at ikke presentasjoner som brukeren ikke har tilgang til dukker opp i denne listen.

`PresentationLobbyService`, som `PresentationListController` benytter seg av, hjelper til med å finne ut brukerens rettigheter. Det benyttes en såkalt `Web Service`-løsning mellom DPG 2.0 og brukerhåndteringssystemet Webucator 3.0 under påloggingen. Etter pålogging har hver bruker et tilhørende `DpgAuthentication`-

objekt som inneholder alle brukerens rettigheter i systemet. PresentationLobbyService vil da undersøke dette objektet og finne ut hvilke presentasjoner som skal vises.

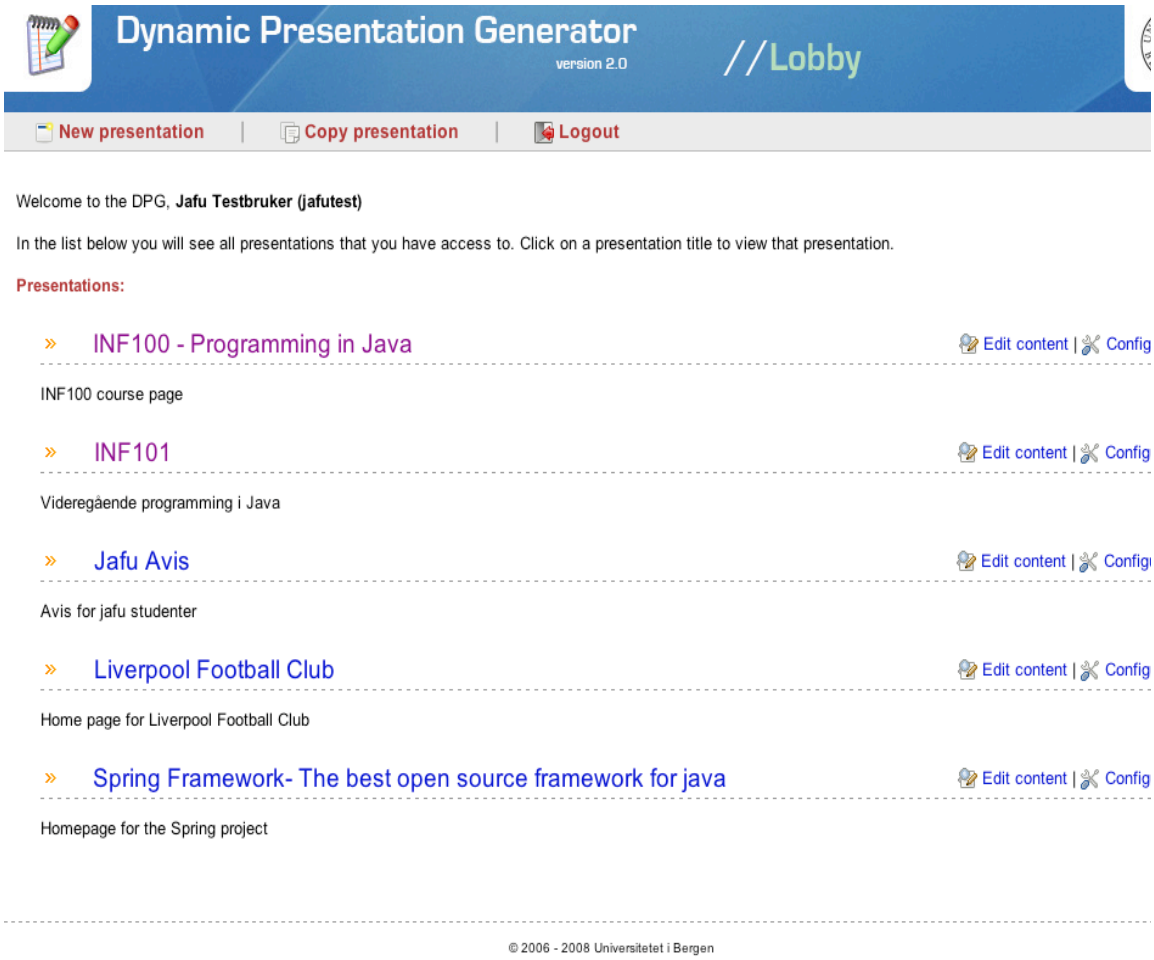


Eksempel 7-1: Felles innloggingsside for alle brukere av DPG 2.0

I eksempel 7-1 ser vi hvordan innloggingssiden ser ut. Innloggingsfunksjonaliteten håndteres av Acegi, som er integrert i Spring-rammeverket, og bruker eksisterende klasser. En detaljert beskrivelse av hvordan dette fungerer er beskrevet i (Løvik, 2008). Alle de tre brukertypene, readers, publishers og administrators, benytter alle den samme innloggingssiden, uavhengig av hvilket delsystem de ønsker tilgang til. Dette bidrar igjen til å skape en sammenheng mellom de tre systemene og en bedre brukeropplevelse.

Etter innlogging får brukeren opp en oversikt over alle presentasjonene som han eller henne har tilgang til (se eksempel 7-2). Informasjonen som finnes på denne nettsiden reflekterer brukerens rettigheter, og brukerens rolle er avgjørende for om enkelte lenker dukker opp eller ikke. For eksempel vil menyvalgene for `New presentation` og `Copy Presentation` er kun tilgjengelig dersom bruker har rollen som administrator. Samtidig vil lenkene i høyre kolonne i listen over presentasjoner også påvirkes av disse rettighetene.

Lenken `Edit content` (som leder bruker til Presentation Content Editor) vil kun være tilgjengelig dersom brukeren har rollen publisher for den aktuelle presentasjonen. Og lenken `configure` (som leder brukeren til Presentation Manager) vil kun være tilgjengelig dersom brukeren har rollen som administrator.



Eksempel 7-2: Grensesnittet reflekterer brukerens rettigheter til hver av presentasjonene

7.4 Hva som er oppnådd

Denne nye rendringsmotoren i DPG 2.0 er på mange måter betydelig bedre enn den gamle DPE-motoren i DPG 1.0. De viktigste fordelene blir oppsummert her:

7.4.1 Bedre organisering av kildekode

Den nye motoren har en mye mer fornuftig inndeling i pakker og klasser enn det den gamle hadde. Samtidig er det brukt flere gode `design patterns`, eksempelvis `Strategy`, `Factory` og `Proxy Pattern`. Disse er alle nærmere forklart i den kjente boken (Gamma et al, 1994), Dette er ofte en direkte følge av at man benytter Spring-rammeverket og løsningene der. Dette sørger for at kildekoden blir mer fleksibel og lettere å forstå, noe som er viktig for dem som skal vedlikeholde og videreutvikle denne motoren.

7.4.2 Støtte for plugins

Et viktig poeng er at motorens funksjonalitet lett kan utvides fordi den har støtte for såkalte plugins. Dette gjør at man i fremtiden kan legge til ekstra funksjonalitet etter behov, uten at dette krever endringer i kildekoden til selve motoren. Mer om hva som er mulig i forhold til plugins kan man se i (Ingvaldsen 2008).

7.4.3 God integrasjon med lobby

Ved å ha en felles innloggingsside og liste over presentasjoner som startsiden til alle brukergruppene, oppnår man en bedre brukeropplevelse som gjør at bruker ikke opplever DPG 2.0 som tre separate systemer. Lobby vil også sørge for at lenker til deler av systemet som bruker ikke har tilgang, blir skjult. Dette gjør at grensesnittet for de vanlige brukerne (de som har rollen `readers`) får et mye enklere og lettfattelig grensesnitt enn for eksempel administratorer. Dette er i tråd med et `design pattern` for sikkerhet kalt `Limited Access`, som er nærmere beskrevet i (Schumacher, et al. 2006).

8 Presentation Content Editor (PCE)

I dette kapittelet vil vi se nærmere på den nye verkøyet for redigering av presentasjonsinnhold, Presentation Content Editor. Dette verktøyet tjener i utgangspunktet det samme formålet som det Repository Administration Tool (RAT) gjorde i DPG 1.0, men er mye mer fokusert inn mot innholdshåndtering og skreddersydd brukerrollen `publisher`. De viktigste forbedringene er mye større brukervennlighet, mulighet for tettere integrasjon med Presentation Viewer, samt en mer modulær struktur som gjør det lettere å utvide verktøyet etter behov.

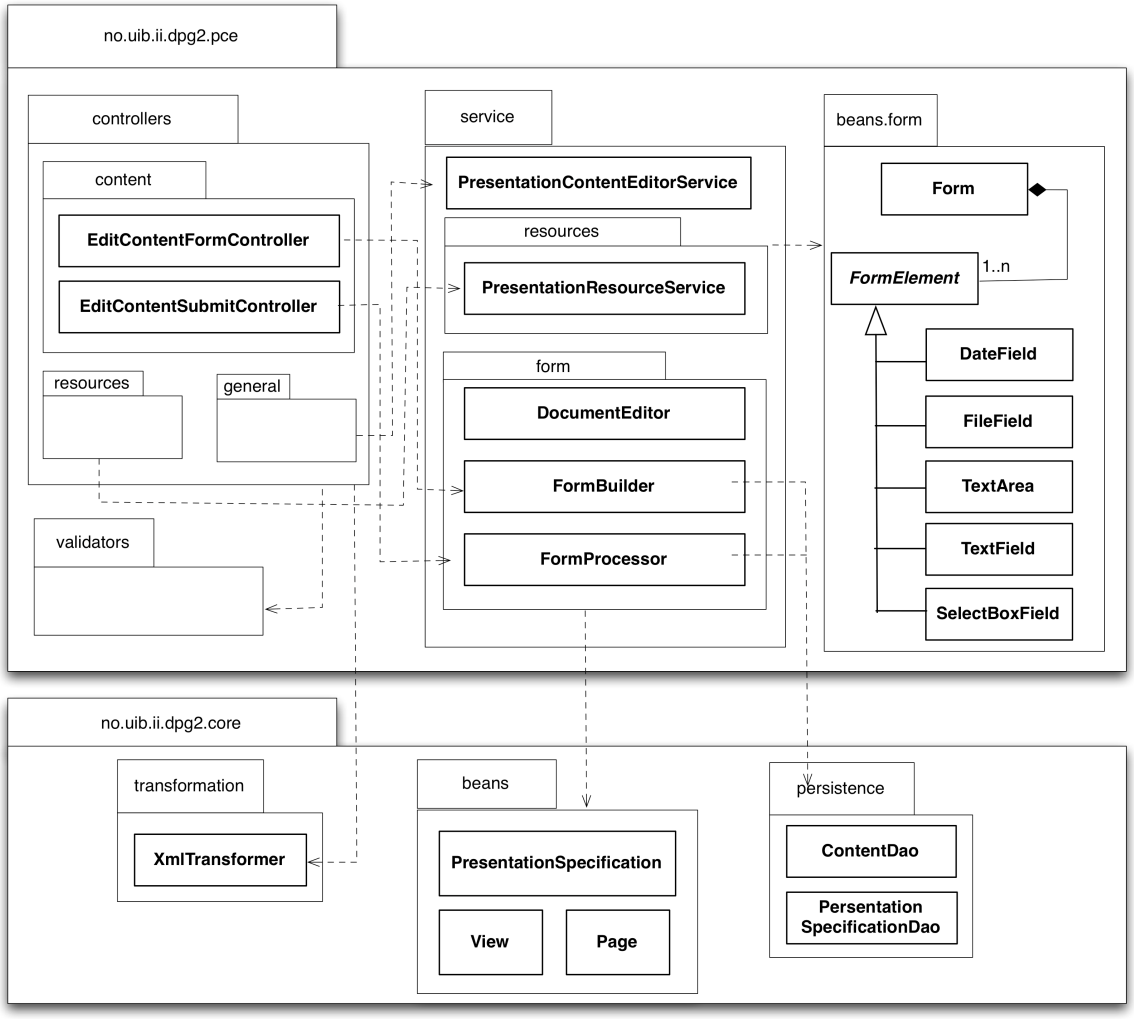
8.1 Oversikt over viktige pakker og klasser

Før vi ser i detalj på hvordan dette delsystemet er bygget opp, skal vi se en oversikt over hvilke pakker og klasser som er involvert (se figur 8-1). Hovedpakken (`no.uib.ii.dpg2.pce`) har omtrent samme struktur som Presentation Viewer, og består dermed blant av underpakkene `controllers`, `service`, `beans`. Siden Presentation Content Editor er et noe mer omfattende delsystem enn Presentation Viewer var, er hver av disse pakken igjen delt i mindre pakker.

I `controllers`-pakken finner vi underpakkene `content`, `resources` og `general`. Den første av dem inneholder to klasser som tar seg av HTTP-forespørsler relatert til skjemabehandling (vi vil se nærmere på disse to klassene i detalj senere i kapittelet). De to øvrige pakkene inneholder klasser for håndtering av ressurser (opplasting av filer, flytting av filer, osv.) og mer generell presentasjonshåndtering. I tillegg har `controllers`-pakken en tilhørende `validators`-pakke, som inneholder klasser som brukes for å validere brukerinput i de forskjellige skjemaene.

`service`-pakken består bl.a. av `PresentationContentEditorService`, en klasse som brukes av kontrollerne i general-pakken. Den tilbyr funksjonalitet for blant annet å deaktivere `pages` eller `views` som ikke trengs i presentasjonen (skjule dem fra bruker) og endre navn på lenker. `PresentationResourceService` brukes av kontrollerne i `resource`-pakken, og som navnet tilsier, er formålet dens å håndtere ressurser for presentasjonene. Det innebærer blant annet lagring av opplastede filer, endre navn på eksisterende ressurser, samt flytte og slette ressurser.

I `beans`-pakken finner man en rekke forskjellige beans som brukes av stort sett alle de øvrige klassene i `pce`-pakken. For å hindre at diagrammet ble for uoversiktlig, vises kun de mest interessante klassene i denne pakken, nærmere bestemt klassene relatert til skjema. Klassen `Form` representerer et skjema for en bestemt entitet i en presentasjon, og inneholder en liste over skjemaelementer, alle basert på den abstrakte klassen `FormElement`. Denne abstrakte klassen har en rekke subklasser, som hver svarer til en felttype i presentasjonsmønsterspesifikasjonen. For eksempel vil klassen `TextField` brukes for alle felt av typen `string`, mens `TextArea` brukes for alle felt av typen `xhtml`. Mer om dette senere i kapittelet.



Figur 8-1: Oversikt over noen av de viktigste pakkene og klassene i dette verktøyet

Mange av klassene man finner i hovedpakken til Presentation Content Editor benytter seg på en eller annen måte av felleslaget (`no.uib.ii.dpg2.core`), på samme måte som Presentation Viewer gjør. Blant annet benytter klassen `EditContentFormController` seg av `XmlTransformer` for å rendre skjema før det sendes til bruker. Både `FormBuilder` og `FormProcessor` må ha tilgang til innholdsdokumentet til entitetsinstansen som redigeres, og sentrale `beans` relatert til presentasjonsspesifikasjonen benyttes av flere av underpakkene i verktøyet.

8.2 Redigering av presentasjonsinnhold

I dette avsnittet vil vi gå gjennom prosessen som skjer når man velger å redigere innholdet i en presentasjon. Vi vil se først på hvordan verktøyet oppleves fra brukerens ståsted, og deretter på hva som foregår bak kulissene i denne prosessen.

8.2.1 Hva brukeren ser

For lettere å illustrere prosessen som brukeren går gjennom for å redigere presentasjonsinnhold, har vi her tatt utgangspunkt i en enkel presentasjon basert på presentasjonsmønster for nettkurs (`coursePattern`), i dette tilfelle et kurs for videregående programmering i Java. Det første som møter brukerne når de åpner denne presentasjonen, er en velkomstside bestående av en kort velkomstmelding, informasjon om hvordan man kommer til kursets brukerforum, samt en liste over viktige meldinger. Videre består presentasjonen av en nettside med fremdriftsplan (med oversikt over pensum og oppgaver), samt en side som inneholder ytterligere informasjon om kurset (hvem som er kursansvarlige, pensumlitteratur, osv.).

Figur 8-2: Her ser vi hovedsiden i presentasjonen for nettkurset INF101

Redigering av innhold

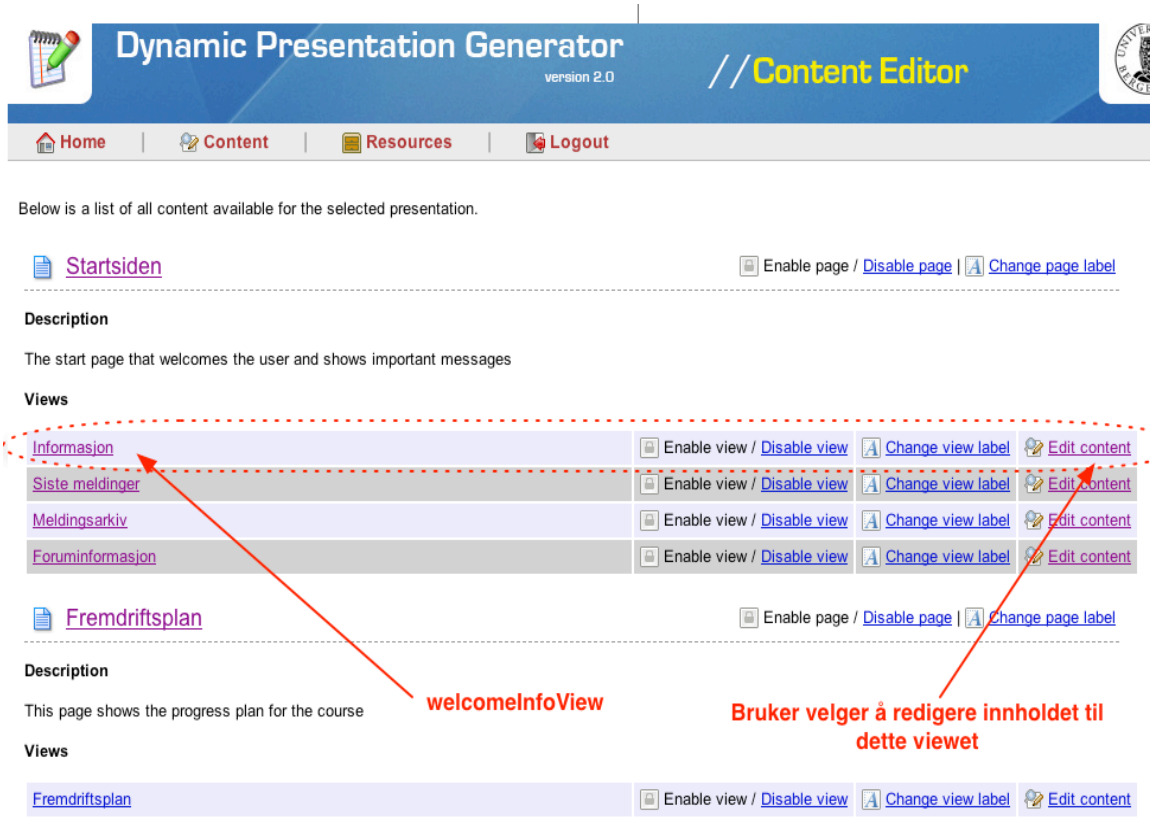
La oss så anta at en bruker med rollen `publisher` ønsker å redigere innholdet i `view`-et *welcomeInfoView* på nettsiden vist i figur 8-2. Brukeren trykker da på lenken til redigeringsverktøyet øverst på nettsiden, noe som fører til at bruker forlater Presentation Viewer og går inn i Presentation Content Editor (se figur 8-3).

Når man først åpner verktøyet får man en oversikt over alle nettsidene (`pages`) og de forskjellige `views` som disse inneholder. For hver nettside har man flere valg. Man kan

velge å slå av (`disable`) en hel nettside, eller kun et konkret view, og dermed gjøre den utilgjengelige for brukere med rollen `readers`. Dette kan være nyttig dersom man finner ut at en `page` eller `view` som tilbys av presentasjonsmønsteret ikke er nødvendig eller passende for den aktuelle presentasjonen, eller at informasjonen som nettsiden inneholder ikke skal offentliggjøres før etter en bestemt dato (for et nettkurs kan det være snakk om en nettside som inneholder eksamensresultater, som ikke man ønsker skal være synlig før etter eksamen). Dersom man slår av alle `views` for en bestemt nettside, vil hele nettsiden automatisk også bli slått av.

Man kan også endre på navnet (kalt `page label` for nettsider, og `view label` for `views`) på nettsiden. Som nevnt i kapittelet om det nye presentasjonsmønsteret, har man valgt å ikke la presentasjonsmønsterspesifikasjonen diktere disse navnene, noe som gjør det mulig for hver presentasjon å gi sine egne navn og dermed kan et presentasjonsmønster brukes av flere presentasjoner på tvers av språk. I vårt nettkurseksempel har de forskjellig elementene i presentasjonsmønsteret engelskspråklige navn, mens selve presentasjonen er norsk.

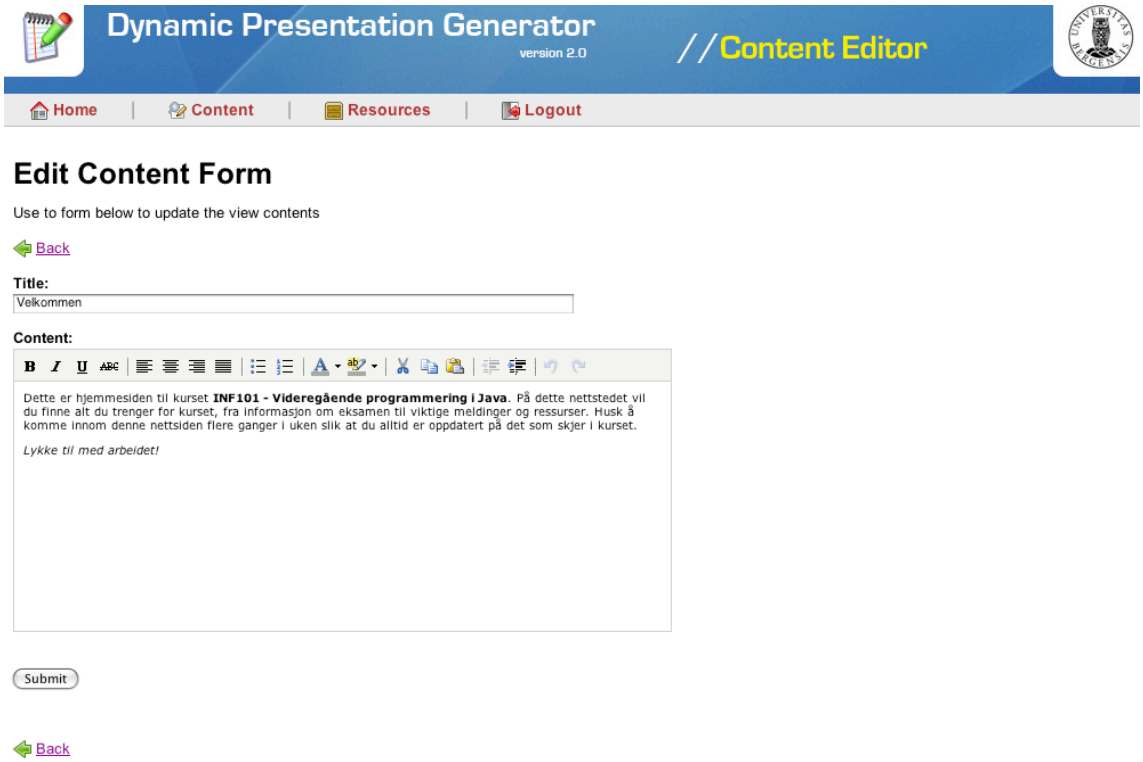
Ved å bla nedover finner brukeren den aktuelle nettsiden (i dette tilfelle *Startsiden*), og velger å redigere det `view`-et som inneholder informasjonen som skal endres (`welcomeInfoView`, under navnet *Informasjon*).



Figur 8-3: Her ser vi hvordan innholdsoversikten i verktøyet kan se ut.

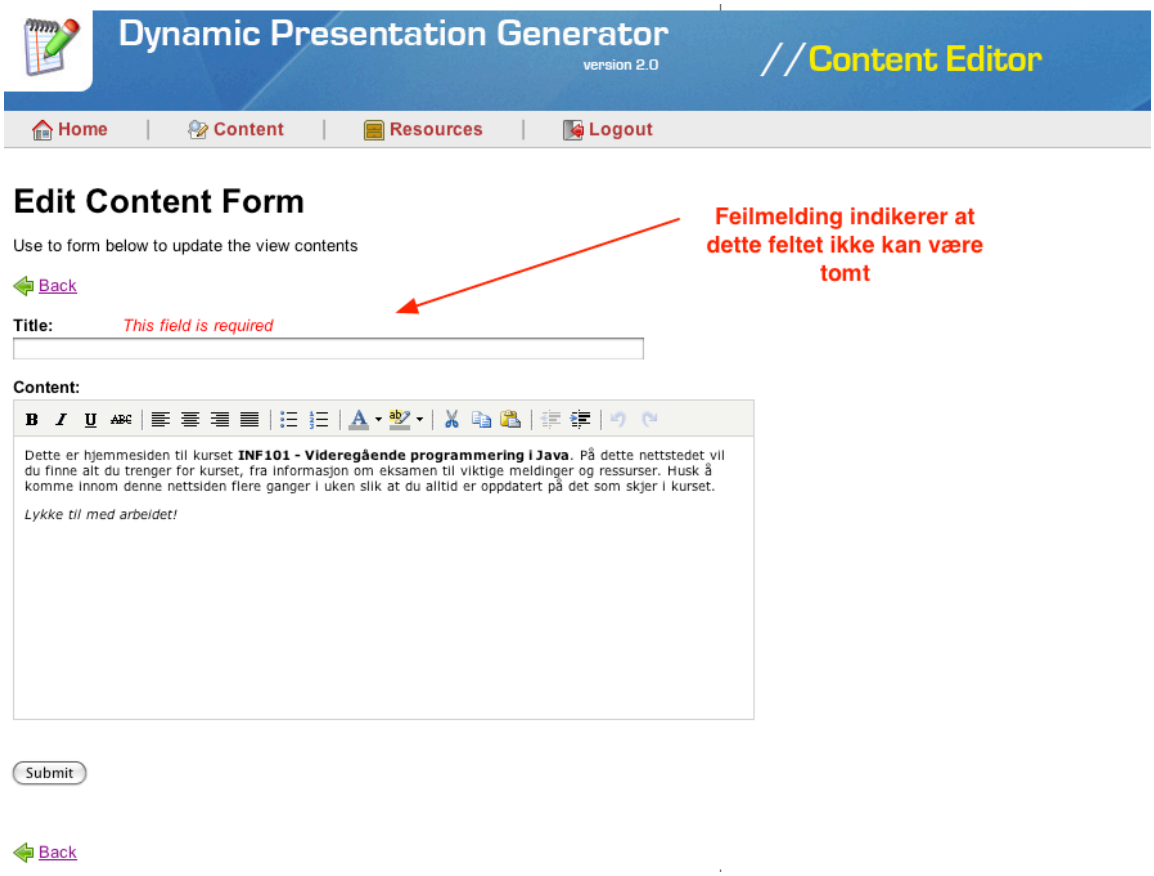
Figur 8-4: Oversikt over informasjonen tilknyttet viewet `welcomeInfoView`

I figur 8-4 har bruker valgt å redigere view-et som inneholder velkomstinformasjonen (`welcomeInfoView`). Først får bruker oppe en enkel oversikt over alt innhold som hører til dette viewet, som i dette tilfelle er en tittel (vanlig enkel tekst) og et innholdsfelt (bestående av såkalt rik tekst med støtte for formatering). Når bruker så velger en av `Edit`-lenkene til høyre, vil man få opp et skjema der man kan redigere informasjonen.



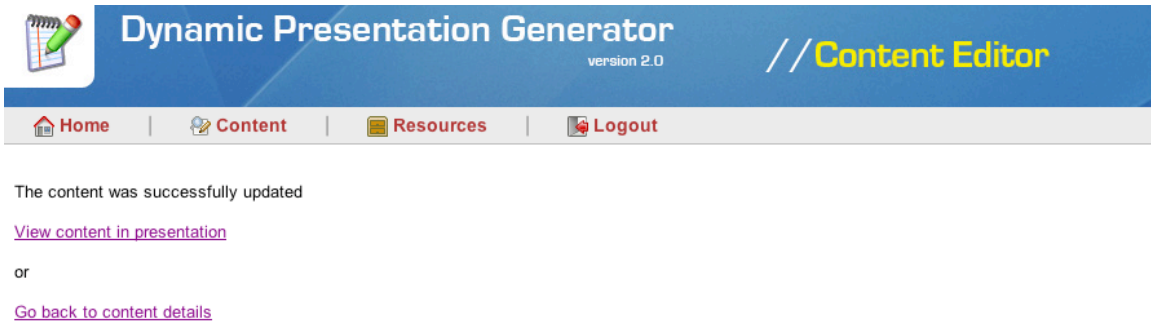
Figur 8-5: Skjema som verktøyet har generert for viewet `welcomeInfoView`

Når man velger å redigere informasjonen til et view, vil verktøyet dynamisk opprette et HTML-skjema som er tilpasset de tilhørende felttypene (se figur 8-5). Vi skal se på de forskjellige felttypene som er støttet av verktøyet litt senere i dette kapittelet. I vårt eksempel består skjema av et tekstfelt (`textfield`) og et tekstområde (`textarea`). Tittelen består kun av ren tekst, og representeres derfor av et ordinær tekstfelt. Selve innholdet derimot, består av rik tekst med støtte for ulik formatering (blant annet uthevet, skråstilt og understreket skrift, ulike justeringsmuligheter, farger, og så videre). Brukeren kan dermed redigere innholdet på tilsvarende måte som i en vanlig teksteditor (for eksempel MS Word, MS Wordpad eller OpenOffice Writer).



Figur 8-6: Skjema vises på nytt med en feilmelding som angir hvilket felt som mangler

La oss nå si at brukeren velger å endre på tittelen til dette viewet, og trykker *submit*. Skjema blir da sendt til verktøyet for validering. Som beskrevet i kapittelet om det nye presentasjonsmønsteret, er det mulig å angi hvilke felt som skal være påkrevd (ved hjelp av attributtet *required*), og dette vil vertkøyet ta hensyn til når skjemaet sendes inn. Dersom noen av de påkrevde feltene står tomme, vil ikke endringene bli lagret og skjema blir vist på nytt med feilmelding. Som figur 8-6 viser, har bruker i dette tilfelle forsøkt å fjerne tittelen helt, og får da beskjed om at dette feltet er påkrevd.



Figur 8-7: Bruker får opp en bekreftelse på at endringen er utført

Etter at skjema er sendt inn og endringen er utført, får brukeren opp en bekreftelse på at endringen ble utført, som figur 8-7 viser. Samtidig har bruker valget mellom enten å gå tilbake til innholdsoversikten i verktøyet, eller å gå direkte til presentasjonen og åpne nettsiden som inneholder informasjonen som nettopp ble endret.

Figur 8-8: Velkomstinformasjonen i presentasjonen er nå oppdatert med den nye tittelen

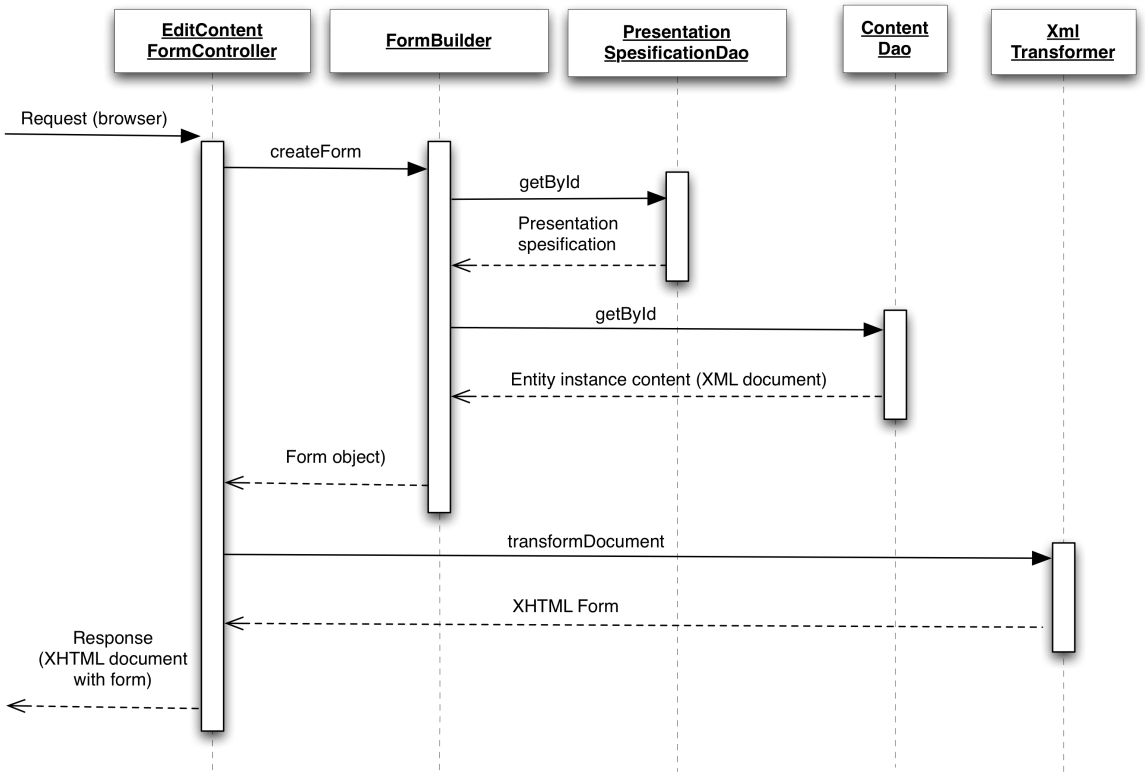
I figur 8-8 ser vi samme startside som før, men der tittelen på velkomstmeldingen har blitt endret til *"Velkommen til kursets hjemmeside!"* ved hjelp av verktøyet. Den øvrige informasjonen på både denne og de andre nettsidene endres på akkurat samme måte. Sammenligner man denne prosessen med hvordan redigering av innhold foregikk i DPG 1.0, ser man helt klart en kraftig forbedring med tanke på brukervennlighet.

8.2.2 Hva som foregår i bakgrunnen

Vi skal nå se på hva som skjer i systemet når en bruker velger å redigere innhold, altså hvilke klasser og pakker som er involvert og måten de samarbeider på. Hele prosessen kan deles opp i to hovedprosesser, den første består i å generere et HTML-skjema med informasjonen til et view, den andre av validering av et innsendt skjema og lagring av endringer. Vi vil se på disse to hovedprosessene ved hjelp av sekvensdiagrammer.

Generering av HTML-skjemaet

Prosessen for generering av HTML-skjema er illustrert i figur 8-9. Det hele begynner med at klassen `EditContentFormController` får inn en forespørsel (i form av en HTTP-forespørsel) som inneholder en rekke forskjellige parametre, blant annet hvilken presentasjon det gjelder, hvilket view som inneholdt informasjonen som skal endres, og eventuelt hvilken subentitet (dersom det er snakk om en nøstet entitet i en liste i et view) som skal endres på. Disse attributtene vil bli hentet ut fra HTTP-objektet og deretter sendt videre til `FormBuilder`.



Figur 8-9: Sekvensdiagram som viser hva som skjer når et skjema genereres av verktøyet

`FormBuilder` har som hovedoppgave å bygge et `Form`-objekt som er en objektrepresentasjon av alle de forskjellige feltene som skjemaet består av. Hvilke felter som skal være med dikteres av presentasjonsmønsterspesifikasjonen som tilhører den aktuelle presentasjonen. Derfor må denne klassen først ha tak i selve presentasjonsspesifikasjonen, noe den får ved å spørre klassen `PresentationSpesificationDao`.

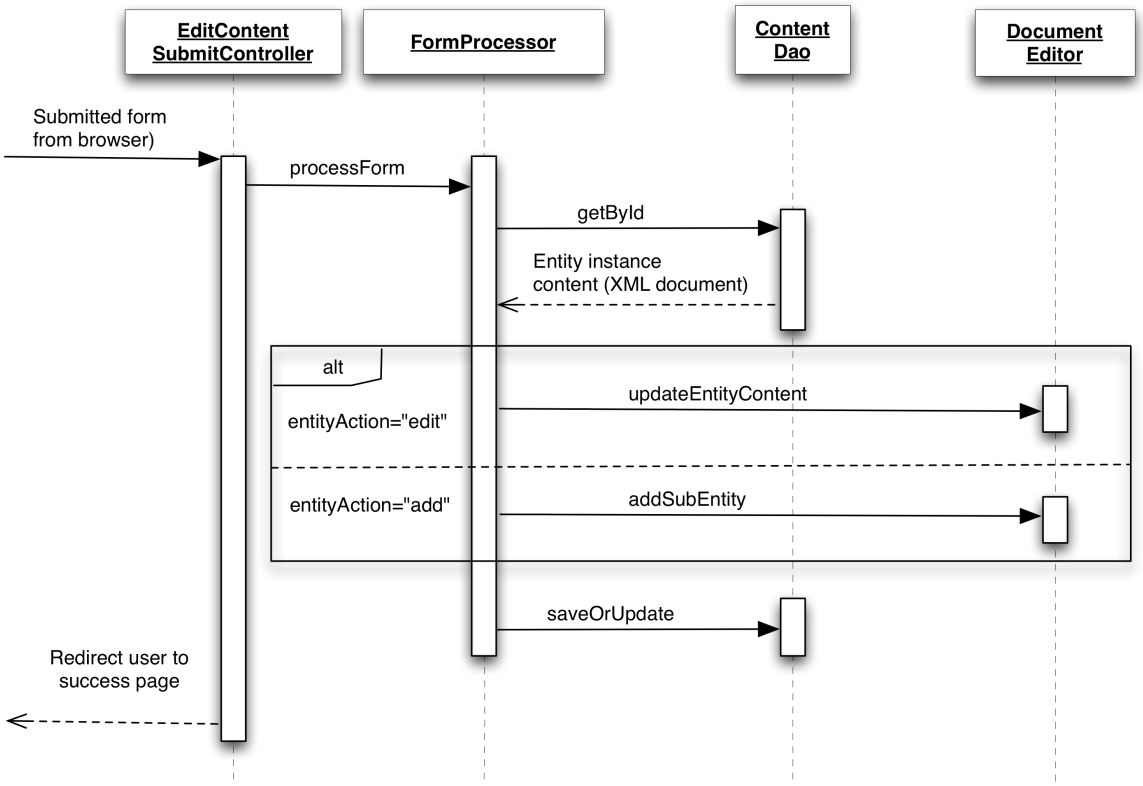
Når en bruker velger å redigere eksisterende informasjon i et view, må skjema vises utfylt med informasjonen som allerede finnes i dette viewet. Derfor må `FormBuilder` også ha tak i innholdsdokumentet til den aktuelle `entity instance` som tilhører viewet, noe den får via `ContentDao`. Når dette er på plass, vil `FormBuilder` gå gjennom strukturen for den aktuelle entiteten og konstruere et passende skjemaelement for hver av de forskjellige felttypene (oversikt over hvilke skjemaelementer gjennomgås senere i kapittelet). Resultatet er et `Form`-objekt bestående av en liste med skjemaelementer, samt detaljer om hvilken presentasjon, view og eventuell subentitet det hører til.

Dette `Form`-objektet returneres så til `EditContentFormController` og gjøres klar til rendring. `Form`-objektet blir også lagret (cachet) i den inneværende sesjonen. Et `Form`-objekt har innebygget støtte for å konvertere seg selv til XML-format, som igjen er lett å transformere ved hjelp av XSLT. Alt `EditContentFormController` trenger å gjøre er å be `XmlTransformer` om å konvertere dette til XHTML, ved hjelp av et XSLT-dokument som er laget spesielt for Presentation Content Editor. XHTML-skjema blir så satt inn i en ordinær JSP-side og sendt tilbake til klienten.

Prosessering av HTML-skjemaet

Etter at bruker har fylt ut skjema og valgt submit, sendes det inn og tas i mot av `EditContentSubmitController` (se figur 8-10). Denne vil først hente ut `Form`-objektet som ligger lagret i sesjonen, og sørge for at det objektet oppdateres med informasjonen som bruker har skrevet inn i skjema. Et `Form`-objekt har også muligheten til å undersøke om skjema er gyldig, altså at hver av skjemaelementene har en lovlig verdi i henhold til de reglene som er definert i presentasjonsmønsterspesifikasjonen (for eksempel attributtet *required*). Dersom skjema inneholder lovlige data, blir det sendt til `FormProcessor` for videre behandling.

`FormProcessor` vil hente ut innholdsdokumentet (som XML), ved hjelp av `ContentDao`, som inneholder den nåværende informasjonen. Dette dokumentet, sammen med `Form`-objektet, blir så sendt videre til `DocumentEditor`. Hvilken metode i denne klassen som skal benyttes er avhengig om det er en eksisterende entitet som skal redigeres, eller om det er snakk om en ny subentitet som skal legges til innholdsdokumentet. I begge tilfellene vil `DocumentEditor` oppdatere innholdsfilen med informasjonen som ligger i `Form`-objektet.



Figur 8-10: Oversikt over hva som skjer når et skjema blir sendt inn av en bruker

Når `DocumentEditor` har oppdatert innholdsfilen, blir den lagret ved hjelp av `ContentDao`. Dersom alt gikk bra, blir brukeren sendt videre til en statusside som forteller at endringen ble utført.

8.3 Gjennomgang av forskjellige skjemaelementer

I dette avsnittet vil vi gå gjennom de forskjellige skjemaelementene som er støttet i redigeringsverktøyet. Hvilket element som skal vises i skjema er avhengig av hvilken felttype som er deklarert i presentasjonsmønsterspesifikasjonen til presentasjonen som redigeres. En fullstendig oversikt over hvilke felttyper som er tilgjengelig finner man i kapittel 6.

Den modulære oppbyggingen av redigeringsverktøyet sørger for at nye skjemaelementer kan legges til dersom det skulle være ønskelig å utvide den nye presentasjonsmønsterimplementasjonen med flere felttyper. Alle skjemaelementene arver fra en felles abstrakt klasse, `FormElement`. Ønsker man å legge til nye elementer, kan man enkelt gjøre dette ved å ved å utvide denne klassen, samt legge til kode for behandling av det nye skjemaelementet i få av klassene til Presentation Content Editor.

8.3.1 Enkel tekst

For enkel tekst, det vil si for felttypen `string`, vises det som et enkelt tekstfelt i skjema. Her har man ingen formateringsmuligheter. Dersom en bruker, enten ved et uhell eller

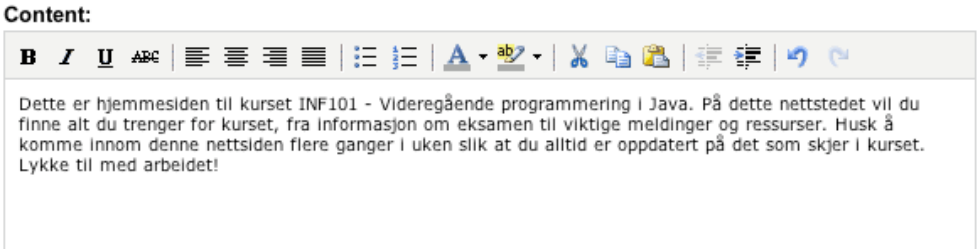
bevisst, skriver inn XHTML-tagger, vil disse bli enkodet (HTML Encoding, Wikipedia u.d.) slik at nettleseren ikke vil tolke disse som en tag. Dette hindrer at man får formatering i de tilfeller der det ikke er ønskelig. Figur 8-11 viser hvordan et slikt tekstfelt kan se ut i verktøyet.

Figur 8-11: Her ser vi hvordan enkel tekst ser ut i skjema

8.3.2 Rik tekst

For felttypen `xhtml`, får man opp et støtte tekstområde med ekstra formateringsfunksjonalitet. I motsetning til andre innholdssystemer, da særlig forum, wikier og diverse bloggingverktøy, benytter man her ikke spesielle tagger eller symboler direkte synlig i teksten. Se figur 8-12 for hvordan dette feltet kan se ut.

I stedet benytter man javascript for å skape en såkalt WYSIWYG (What You See Is What You Get)-effekt. Bruker trenger da ikke å lære seg forskjellige formateringstagger, og all formatering som legges på vises slik som det vil bli seende ut etter innholdet er publisert. Denne funksjonaliteten realiseres ved hjelp av TinyMCE, en Javascript WYSIWYG-editor laget av Moxiecode Systems og baserer seg på åpen kildekode (TinyMCE - JavaScript WYSIWYG Editor). Denne er lett å integrere inn i eksisterende systemer og er utvidbar gjennom en kraftig pluginarkitektur, skulle det være behov for flere funksjoner etter hvert som Presentation Content Editor videreutvikles.

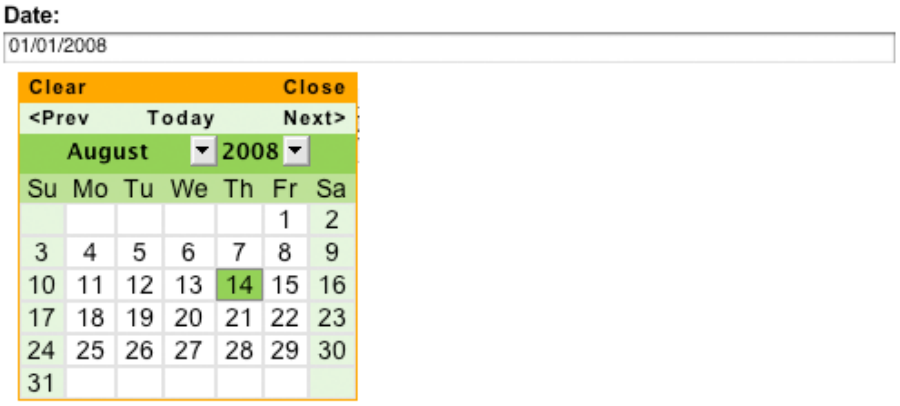


Figur 8-12: Her ser vi hvordan rik tekst kan redigeres i skjema

Alle som er komfortabel med bruk av et enkelt tekstredigeringsverktøy som MS Word, OpenOffice Writer eller lignende vil ikke ha noe problemer med å ta i bruk formateringsmulighetene dette skjemaelementet tilbyr.

8.3.3 Datovelger

I de tilfellene man bruker felttypen `date` vil man få opp et tekstfelt slik som for enkel tekst. Men når dette tekstfeltet er aktivt, vil det sprette opp en datovelger som hjelper bruker å velge en gyldig dato. Da unngår man at bruker skriver inn en gal dato, eller skriver inn dato med feil format. Se figur 8-13 for et eksempel på denne datovelgeren.



Figur 8-13: Her ser vi hvordan datovelgeren ser ut i et skjema som inneholder et datofelt

Dette er også laget i javascript, og baserer seg på et plugin for jQuery (jQuery JavaScript Library), et populært javascript-rammeverk som er integrert i Presentation Content Editor. Det konkrete pluginet som benyttes er `DatePicker` (jQuery JavaScript Library - DatePicker Plugin) og det er mulig å konfigurere både kalenderens utseende og virkemåte skulle det være behov for det under videreutviklingen av redigeringsverkøyet.

8.3.4 Filopplasting

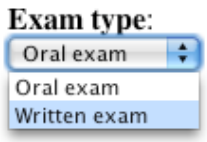
For felttypen `file` dukker det opp en helt ordinært XHTML-felt for filopplasting. Bruker trygger på `velg..`-knappen og får opp en vanlig filvelgervindu. Den lokale stien til filen vises i feltet, og blir lastet opp når skjema sendes inn. Alle filer som blir lastet opp havner i ressursmappen til den aktuelle presentasjonen. Dersom man redigerer innhold som allerede har en fil, vil navnet på den nåværende filen (`current file`) vises før filvelgerelementet. Om bruker velger å laste opp en ny fil, blir den nåværende filen fortsatt liggende i ressursmappen, og kan senere bli slettet (ved hjelp av ressurshåndteringen forklart senere i kapittelet) om ønskelig. Se figur 8-14 for et eksempel på dette.

Figur 8-14: Her ser vi skjemaelementet for filopplasting

8.3.5 Begrenset valg

Dersom man i mønsterspesifikasjonen har felt av typen `string` og som har attributtet `allowedValues`, vil det i stedet for et vanlig tekstfelt vises en såkalt dropdown-boks i skjema. Verdiene man kan velge mellom er de samme verdiene angitt av attributten `allowedValues`. Dette hindrer at noen skriver inn en ulovlig verdi for det aktuelle feltet. Figur 8-15 viser hvordan dette feltet kan se ut. I dette tilfelle er det snakk om

eksamenstype, der det kun er to mulige verdier å velge mellom, altså muntlig og skriftlig eksamen.



Figur 8-15: Her ser vi hvordan et felt med et sett lovlige verdier kan se ut i skjema

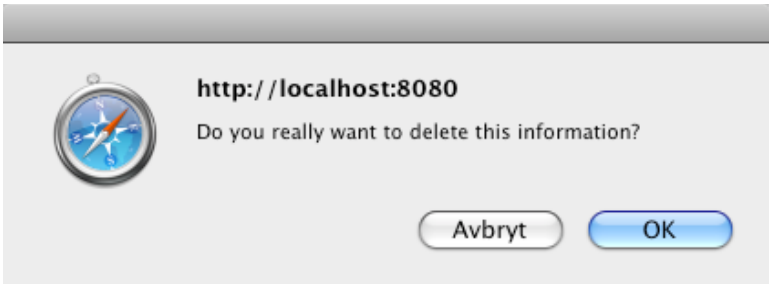
Presentation Content Editor er smart nok til å sjekke det som sendes inn i skjema opp mot de lovlige verdiene for dette feltet, slik at det ikke hjelper å tukle med html-skjemaet i et forsøk på å sende inn ulovlige verdier. Mer om potensielle problemer som kan oppstå og hvordan slike angrep utføres, se (Huseby 2004).

8.3.6 Håndtering av subentiterer

Den nye presentasjonsmønsterimplementasjonen inneholder mulig å nøste entiteter, og dermed lage lister av entiteter, og til og med lister av lister, om ønskelig. Når man velger å redigere innholdet i en `entity-instance` som består av en liste med nøstede, vil verktøyet automatisk legge til lenker for å slette eller redigere et bestemt entitet i listen, eller legge til en ny entitet. Se figur 8-16 for et eksempel på dette. Øverst til høyre finner man lenken for å legge til en ny melding i feltet `messages` (som er av typen entity-list i presentasjonsmønsterspesifikasjonen). For hver melding kan man velge å redigere innholdet, eller fjerne meldingen fra listen.

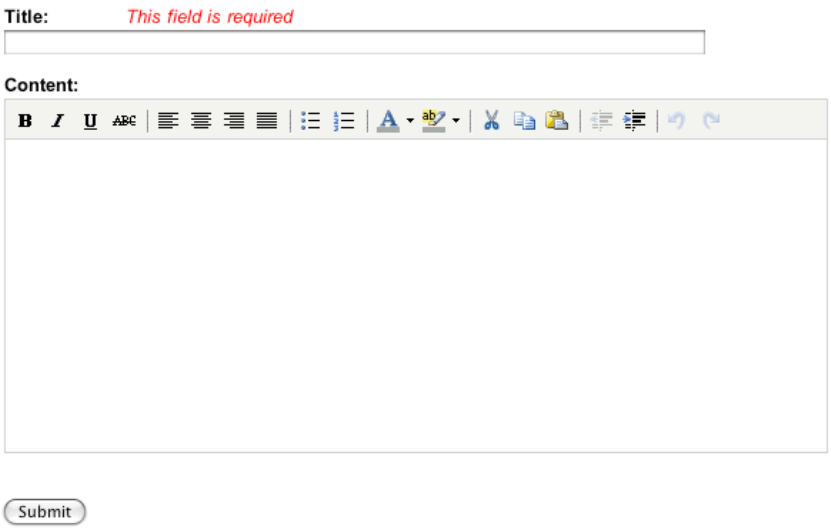
Figur 8-16: Hvordan en liste med meldinger kan se ut i verktøyet

Dersom bruker velger å slette en entitet i en liste, vil verktøyet først be om en bekreftelse på dette (se figur 8-17). Dette er for å unngå at man ved et uhell sletter noe man ikke ønsker å slette. Denne dialogen er implementert i JavaScript, noe de aller fleste nettlesere har aktivert. Merk at utseende på denne dialogen er fullstendig avhengig av hvilken nettleser som benyttes, og den kan derfor avvike noe fra den figuren som er vist her.



Figur 8-17: En dialog vil sprette opp når bruker forsøker å slette et element fra listen

Redigering av en eksisterende entitet i listen skjer på samme måte som øvrige entiteter, og trenger dermed ingen nærmere forklaring. Når man derimot velger å legge til et element i en liste, vil et skjema bli generert på omtrent på vanlig måte, bare det vil være helt tomt (se figur 8-18). Et skjult flagg blir også lagt inn i skjema som forteller verktøyet at det innsendte skjema er et nytt listeelement, og ikke et eksisterende listeelement.



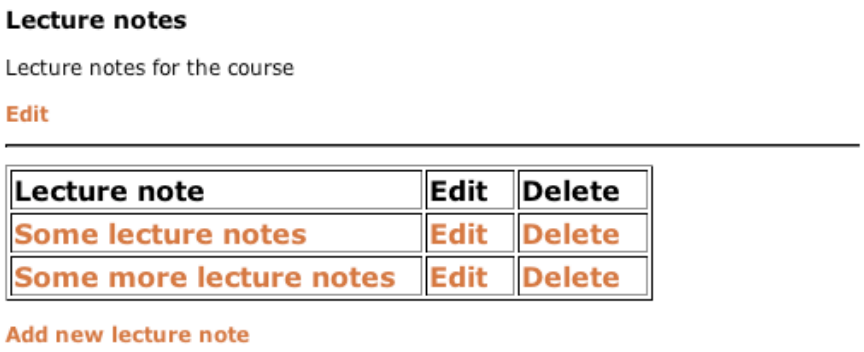
Figur 8-18: Et tomt skjema for å legge til en entitet i listen over meldinger

8.4 Tettere integrasjon mellom Presentation Viewer og Presentation Content Editor

Dersom en `publisher` ønsker å endre på informasjonen i en presentasjon, kan man enkelt bytte til Presentation Content Editor ved å klikke på lenken i presentasjonens `navigation bar`, finne frem til hvilken page og hvilket view som skal endres på, og velge rediger. For de fleste presentasjoner vil dette være en grei løsning, spesielt der informasjonen leses oftere enn den endres på.

Men i enkelte tilfeller kan det være ønskelig med en mye tettere integrasjon mellom `Presentation Viewer` og `Presentation Content Editor`. Det er derfor mulig å vise lenker for redigering av informasjonen i et view direkte i `Presentation Viewer`, uten at man må først gå inn i redigeringsverktøyet og finne frem til viewet på nytt. Med disse lenkene kommer man direkte til det aktuelle stedet i redigeringsverktøyet, uten mellomsteg. Fordelen er at det tar noe mindre tid å redigere innholdet på siden, og det er lettere å finne frem til informasjonen man ønsker å endre på.

I figur 8-19 ser man hvordan disse lenkene kan se ut. Her ser vi et view, med en liste med forelesningsnotater, der man har tatt i bruk muligheten for redigeringslenker direkte i Presentation Viewer. Plasseringen av lenkene er opp til utvikler, slik at lenkene kan sys inn i resten av designet på nettsidene. Merk at disse lenkene er kun synlig for brukere som er `publisher` i den aktuelle presentasjonen



Figur 8-19: Et view der man har plassert redigeringslenker direkte i Presentation Viewer

For å ta i bruk denne funksjonaliteten kreves det noe implementeringsarbeid på presentasjonsmønsternivå. DPG-en kan nemlig ikke selv automatisk plassere disse lenkene, da plasseringen av disse vil være avhengig av layout og utseende, kosmetiske ting som et automatisk system ikke kan ta stilling til. Det er mulig å automatisere dette helt, men det ville gått utover utseende og dermed brukeropplevelsen.

Hver gang innholdet et view skal rendres til XHTML ved hjelp av XSLT, undersøker Presentation Viewer hvilken rolle brukeren har i systemet. Dersom denne brukeren har rollen publisher for den aktuelle presentasjonen, vil et flagg ved navn `isPublisher` bli satt til true og sendt inn til XSLT-transformatoren.

```xml
<?xml version='1.0' encoding='UTF-8'?>
<lectureNotesListEntity xmlns:xhtml="http://www.w3.org/1999/xhtml"
    entityInstanceId="34343" type="rootEntity"
    entityPath="lectureNotesListEntityInstance_lectureNotesListEntity"
    createdBy="khalid" createdOn="01/01/2008 14:00" updatedBy="khalid" updatedOn="01/01/2008 14:00">

    <title type="string">Lecture notes</title>
    <description type="xhtml"><![CDATA[Lecture notes for the course]]></description>

    <lecturenotes
        type="entity-list"
        entityPath="lectureNotesListEntityInstance_lectureNotesListEntity_lecturenotes_lectureNotesEntity">
        <lectureNotesEntity
            type="subEntity" entityInstanceId="1234"
            entityPath="lectureNotesListEntityInstance_lectureNotesListEntity_lecturenotes_lectureNotesEntity_1234"
            createdBy="khalid" createdOn="01/01/2008 14:00" updatedBy="khalid" updatedOn="01/01/2008 14:00">
                <title type="string">Some lecture notes</title>
                <url type="resource-url">springbook.jpg</url>
        </lectureNotesEntity>
        <lectureNotesEntity
            type="subEntity" entityInstanceId="1238"
            entityPath="lectureNotesListEntityInstance_lectureNotesListEntity_lecturenotes_lectureNotesEntity_1238"
            createdBy="khalid" createdOn="01/01/2008 14:00" updatedBy="khalid" updatedOn="01/01/2008 14:00">
                <title type="string">Some more lecture notes</title>
                <url type="resource-url">javabook.jpg</url>
        </lectureNotesEntity>
    </lecturenotes>
</lectureNotesListEntity>
```

Eksempel 8-1: Innholdsfil før den er prosessert av `EntityUrlActionGenerator`

I tillegg vil XML-dokumentet med innholdet (se eksempel 8-1) bli modifisert slik at hver entitet og subentitet får de nødvendige lenkene (add, delete, eller edit) som trengs for å endre på informasjonen. Disse URL-ene blir automatisk bygget opp av klassen `EntityUrlActionGenerator`, og lagt til entitetene som XML-attributter, noe som er lett å hente ut i XSLT-dokumentet som presentasjonsmønsterutvikleren har implementert. Hvordan den oppdaterte innholdsfilen ser ut etter at dette er utført, kan man se et eksempel på i eksempel 8-2.

```
<?xml version="1.0" encoding="UTF-8"?>
<lectureNotesListEntity xmlns:xhtml="http://www.w3.org/1999/xhtml"
    entityInstanceId="34343" type="rootEntity"
    entityPath="lectureNotesListEntityInstance_lectureNotesListEntity"
    createdBy="khalid" createdOn="01/01/2008 14:00" updatedBy="khalid" updatedOn="01/01/2008 14:00"
    editUrl="../pce/content/editContentForm.htm?presentationId=inf100h08&amp;pageId=resourcePage
             &amp;viewId=lectureNotesView&amp;entityPath=lectureNotesListEntityInstance_
             lectureNotesListEntity&amp;entityAction=edit">

    <title type="string">Lecture notes</title>
    <description type="xhtml"><![CDATA[Lecture notes for the course]]></description>

    <lecturenotes type="entity-list"
        entityPath="lectureNotesListEntityInstance_lectureNotesListEntity_lecturenotes_lectureNotesEntity"
        addUrl="../pce/content/editContentForm.htm?presentationId=inf100h08&amp;pageId=resourcePage
            &amp;viewId=lectureNotesView&amp;entityPath=lectureNotesListEntityInstance_lectureNotes
            ListEntity_lecturenotes_lectureNotesEntity&amp;entityAction=add">
        <lectureNotesEntity
            type="subEntity" entityInstanceId="1234"
            entityPath="lectureNotesListEntityInstance_lectureNotesListEntity_lecturenotes_lectureNotesEntity_1234"
            createdBy="khalid" createdOn="01/01/2008 14:00" updatedBy="khalid" updatedOn="01/01/2008 14:00"
            editUrl="../pce/content/editContentForm.htm?presentationId=inf100h08&amp;pageId=resourcePage&amp;
                     viewId=lectureNotesView&amp;entityPath=lectureNotesListEntityInstance_lectureNotesListEntity_
                     lecturenotes_lectureNotesEntity_1234&amp;entityAction=edit"
            deleteUrl="../pce/content/deleteContent.htm?presentationId=inf100h08&amp;pageId=resourcePage&amp;
                     viewId=lectureNotesView&amp;entityPath=lectureNotesListEntityInstance_lectureNotesListEntity_
                     lecturenotes_lectureNotesEntity_1234&amp;entityAction=delete">
                <title type="string">Some lecture notes</title>
                <url type="resource-url">springbook.jpg</url>
        </lectureNotesEntity>
        <lectureNotesEntity
            type="subEntity" entityInstanceId="1238"
            entityPath="lectureNotesListEntityInstance_lectureNotesListEntity_lecturenotes_lectureNotesEntity_1238"
            createdBy="khalid" createdOn="01/01/2008 14:00" updatedBy="khalid" updatedOn="01/01/2008 14:00"
            editUrl="../pce/content/editContentForm.htm?presentationId=inf100h08&amp;pageId=
                     resourcePage&amp;viewId=lectureNotesView&amp;entityPath=lectureNote
                     sListEntityInstance_lectureNotesListEntity_lecturenotes_lectureNotes
                     Entity_1238&amp;entityAction=edit"
            deleteUrl="../pce/content/deleteContent.htm?presentationId=inf100h08&amp;pageId=
                     resourcePage&amp;viewId=lectureNotesView&amp;entityPath=lectureNotesList
                     EntityInstance_lectureNotesListEntity_lecturenotes_lectureNotesEntity_
                     1238&amp;entityAction=delete">
                <title type="string">Some more lecture notes</title>
                <url type="resource-url">javabook.jpg</url>
        </lectureNotesEntity>
    </lecturenotes>
</lectureNotesListEntity>
```

Eksempel 8-2: Innholdsfil etter at den er prosessert av `EntityUrlActionGenerator`

Når disse lenkene så er bygget opp og lagt til innholdsdokumentet, kan man plassere dem på nettsiden ved hjelp av transformasjonsdokumentet (XSLT) til hvert view. Hvordan dette kan gjøres ser man i eksempel 8-3. Her ser man at presentasjonsmønsterutvikler har full kontroll over plasseringen og utseende til disse lenkene, slik at det ikke funksjonaliteten går på bekostning av utseende og brukervennlighet. Merk at man alltid må teste om flagget isPublisher er satt til å være sant, ellers risikerer man at vanlige brukere får se lenkene (disse vil likevel ikke kunne utføre noen endringer, da sikkerhetslaget i Presentation Content Editor alltid undersøker rettighetene til de enkelte bruker som ønsker tilgang).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" version="1.0" encoding="UTF-8" indent="yes"/>
    <xsl:param name="resourcesUrl" select="'resourcesUrl'" />

    <!-- Få tilgang til flagget som sier om en bruker er publisher eller ikke -->
    <xsl:param name="isPublisher" select="'isPublisher'" />

    <!-- Regel for selve listen -->
    <xsl:template match="lectureNotesListEntity">
        <div id="selectedView">
            <div class="messageBoxHeader"><xsl:value-of select="title" /></div>
            <p><xsl:value-of select="description" disable-output-escaping="yes" /></p>

            <!-- Dersom bruker er publisher, vis edit-lenke for tittel og beskrivelse -->
            <xsl:if test="$isPublisher='true'">
                <xsl:element name="a">
                    <xsl:attribute name="href"><xsl:value-of select="@editUrl" /></xsl:attribute>
                    Edit
                </xsl:element>
            </xsl:if>
            <hr />

            <!-- Sett opp tabell for forelesningsnotater, ta hensyn til om edit/delete skal vises -->
            <table border="1">
                <thead>
                    <th>Lecture note</th>
                    <xsl:if test="$isPublisher='true'">
                        <th>Edit</th>
                        <th>Delete</th>
                    </xsl:if>
                </thead>
                <tbody>
                    <!-- Behandle alle listelemetene her -->
                    <xsl:apply-templates />
                </tbody>
            </table>

            <!-- Legg til add-lenke for forelesningsnotat -->
            <xsl:if test="$isPublisher='true'">
                <p>
                    <xsl:element name="a">
                        <xsl:attribute name="href">
                            <xsl:value-of select="lecturenotes/@addUrl" />
                        </xsl:attribute>
                        Add new lecture note
                    </xsl:element>
                </p>
            </xsl:if>
        </div>
    </xsl:template>
```

(eksempel 8-3 fortsetter på neste side)

```
    <!-- Regel for et element i listen -->
    <xsl:template match="lectureNotesEntity">
        <tr>
            <!-- Vis tittel og lenke til selve innholdsfilen -->
            <td>
                <xsl:element name="a">
                    <xsl:attribute name="href">
                        <xsl:value-of select="$resourcesUrl" /><xsl:value-of select="url" />
                    </xsl:attribute>
                    <xsl:value-of select="title" />
                </xsl:element>
            </td>

            <!-- Vis lenker for edit og delete, hvis bruker er publisher -->
            <xsl:if test="$isPublisher='true'">
                <td>
                    <xsl:element name="a">
                        <xsl:attribute name="href"><xsl:value-of select="@editUrl" /></xsl:attribute>
                        Edit
                    </xsl:element>
                </td>
                <td>
                    <!-- Merk bruk av onclick javascript confirm dialog for sletting -->
                    <xsl:element name="a">
                        <xsl:attribute name="href">
                            <xsl:value-of select="@deleteUrl" />
                        </xsl:attribute>
                        <xsl:attribute name="onclick">
                            return confirm('Do you really want to delete this message?');
                        </xsl:attribute>
                        Delete
                    </xsl:element>
                </td>
            </xsl:if>
        </tr>
    </xsl:template>

    <!-- Dette elementet skal vises som HTML og skal ikke escapes -->
    <xsl:template match="content">
        <xsl:value-of select="." disable-output-escaping="yes" />
    </xsl:template>

    <!-- Disse templatene kalles i eksplisitt i hovedregelen og skal derfor stå tomme -->
    <xsl:template match="title"/>
    <xsl:template match="description"/>

</xsl:stylesheet>
```

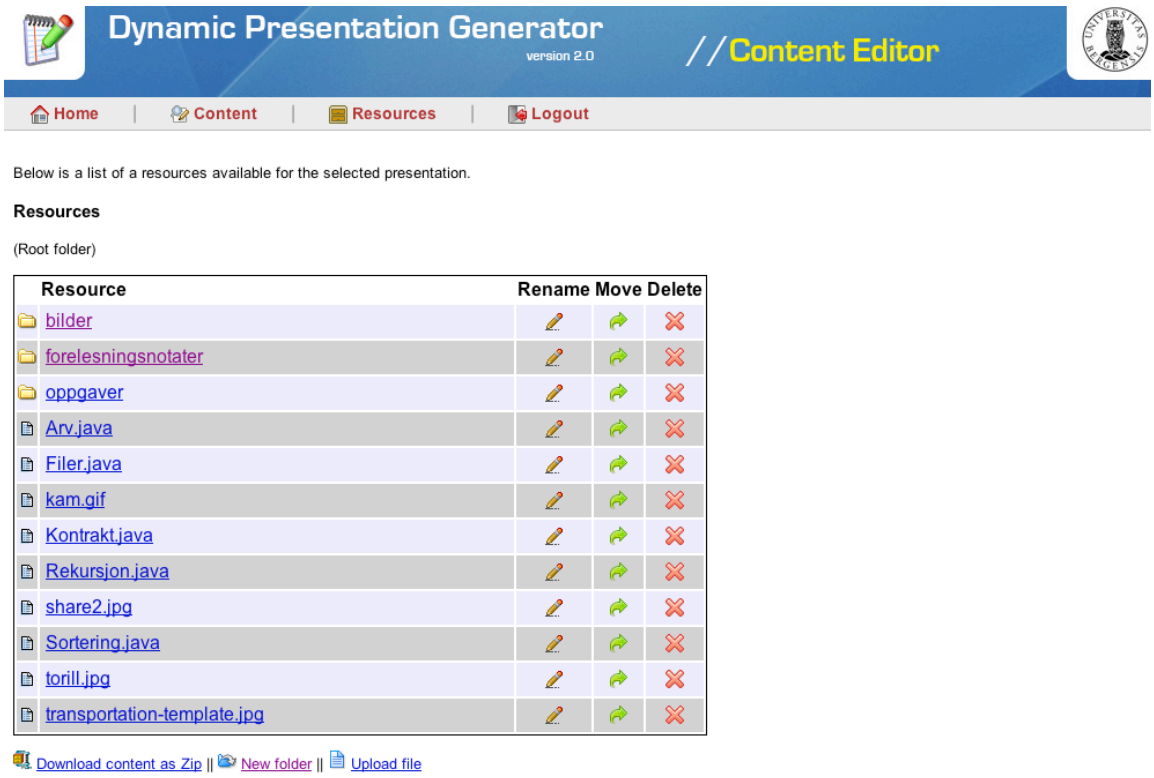
Eksempel 8-3: Transformasjon til et view med redigeringslenker utplassert

Kort oppsummert er dette en funksjonalitet som kan være ideelt å bruke i enkelte situasjoner. Ved å bruke dette kan man kunne utføre endringer noe raskere og enklere. Ulempen kan være når man har komplekse views med flere nøstede elementer, da kan nemlig mengden med add, edit og delete-lenker gjøre et view uoversiktlig og vanskelig å lese, om man ikke er svært nøye med layout og utseende.

8.5 Håndtering av ressurser

Alle filer som lastes opp til en presentasjon gjennom et skjema havner i ressursområdet til presentasjonen. Dette ressursområdet kan ses på som en hovedmappe, som kan ha flere filer og undermapper, på samme måte som en harddisk på en vanlig datamaskin. Når man

fra verktøyet velger `Resources` fra menyen, får kan en oversikt over hva som ligger på ressursområdet til presentasjonen (se figur 8-20).



Figur 8-20: Oversikt over tilgjengelige ressurser til en presentasjon

Presentation Content Editor tilbyr tilsvarende funksjonalitet som man forventer av et enkelt filhåndteringssystem. Det innebærer at man kan legge til filer (laste opp) eller mapper, flytte disse fra et sted til et annet, endre navn, eller fjerne et element. Flytter man eller endrer navn på en fil som er knyttet opp til et eller flere view, vil verktøyet automatisk sørge for at stier i innholdsdokumentene til viewet oppdateres slik at fillenker ikke blir ødelagt. Ødelagte lenker var et stort problem i DPG 1.0, og dette er en god måte å unngå dette på i DPG 2.0.

8.6 Hva som er oppnådd

Som vi har sett, gjør det nye redigeringsverktøyet det veldig enkelt for `publishers` å endre på innholdet i presentasjoner. Her følger en kort oversikt over de viktigste fordelene med dette verktøyet i forhold til DPG 1.0:

8.6.1 Enklere innholdsredigering

Det er ikke lenger nødvendig for de som er ansvarlig for innholdet i en presentasjon å manuelt redigere avanserte XML-dokumenter, med alle de allerede godt dokumenterte ulemper det fører med seg. Nå skjer all innholdshåndtering gjennom lettfattelige HTML-skjema, og krever derfor ikke mer kunnskaper utover vanlig bruk av datamaskin. Støtte

for rik tekst med formateringsmuligheter krever heller ikke spesielle tagger eller kode, da dette gjøres ved hjelp av WYSIWYG-editor. Samtidig er det bedre ressurshåndtering, og muligheter for å organisere relaterte filer i egne undermapper, gjør det lettere å holde oversikt over hvilke ressurser som er knyttet til en presentasjon.

8.6.2 Bedre integrasjon med Presentation Viewer

Selve grensesnittet er også forbedret betydelig i forhold til Repository Administration Tool (RAT) i DPG 1.0, med enklere navigasjon og mer oversiktlige nettsider. RAT-en hadde også egen separat innloggingsvindu, og fremsto dermed som et selvstendig system, og det var ikke lett å navigere fra det ene systemet til det andre. I det nye systemet er det mye bedre integrasjon mellom Presentation Viewer og Presentation Content Editor, det er lett å gå fra den ene til den andre, og man kan til å med integrere redigeringslenker direkte i Presentation Viewer, om ønskelig.

8.6.3 Bedre feilhåndtering og datavalidering

I det gamle systemet var det alt for lett å gjøre kritiske feil, siden man redigerte XML-dokumenter direkte. Slike feil kan være alt fra mindre alvorlige ting, som for eksempel at man skriver en verdi på galt format, til mer kritiske ting som for eksempel at man glemmer å lukke et element, som igjen kan føre til at systemet blir ustabilt og data går tapt. Det nye redigeringsverktøyet hindrer alt dette, siden man ikke ser noe til XML-dokumenter. De nye skjemaelementene, som WYSIWYG-editor, dropdown-boks for begrenset valg og datovelger, sørger for at informasjonen blir lagret på lovlig format. Innsendte skjema med feil blir ikke bli godkjent og i stedet visst på nytt, med en forståelig feilmelding (fremfor kryptiske XML-feilmeldinger fra parseren, noe som var vanlig i det gamle systemet).

8.6.4 Tilrettelagt for videreutvikling og fremtidige utvidelser

Som en følge av at det nye systemet er mer modulært og har bedre laginndeling enn RAT-en, vil det være mye lettere å vedlikeholde og utvide i tiden som kommer. Igjen er det brukt interfaces mellom de forskjellige lagene, slik at det er lettere å skifte ut en implementasjon med en annen uten at det skaper problematiske ringvirkninger i systemet.

9 Presentation Manager (PM)

I dette kapittelet skal vi se nærmere på det tredje og siste delsystemet i DPG 2.0, kalt Presentation Manager. Dette verktøyet brukes av administratorer for blant annet å konfigurere eksisterende presentasjoner, opprette nye presentasjoner, og slette presentasjoner det ikke lenger er behov for. Her styres med andre ord hele livssyklusen til en presentasjon. Vi skal se nærmere på de viktigste funksjonene verktøyet tilbyr, hvordan det grafiske grensesnittet er lagt opp, og hva som skjer bak kulissene i systemet.

9.1 Oversikt over viktige pakker og klasser

Aller først skal vi få en oversikt over de viktigste pakkene og plassene, hvordan de er organisert og hvilke relasjoner det er mellom dem (se figur 9-1). Presentation Manager har omtrent samme pakkestruktur som Presentation Viewer og Presentation Content Editor. Underpakkene `controllers`, `beans`, `service` og `validators` burde derfor være kjent fra tidligere kapitler. Kontrakter (`interfaces`) er brukt der det er hensiktsmessig for å sørge for løsere kobling mellom lagene og mer fleksibel oppbygging av systemet.

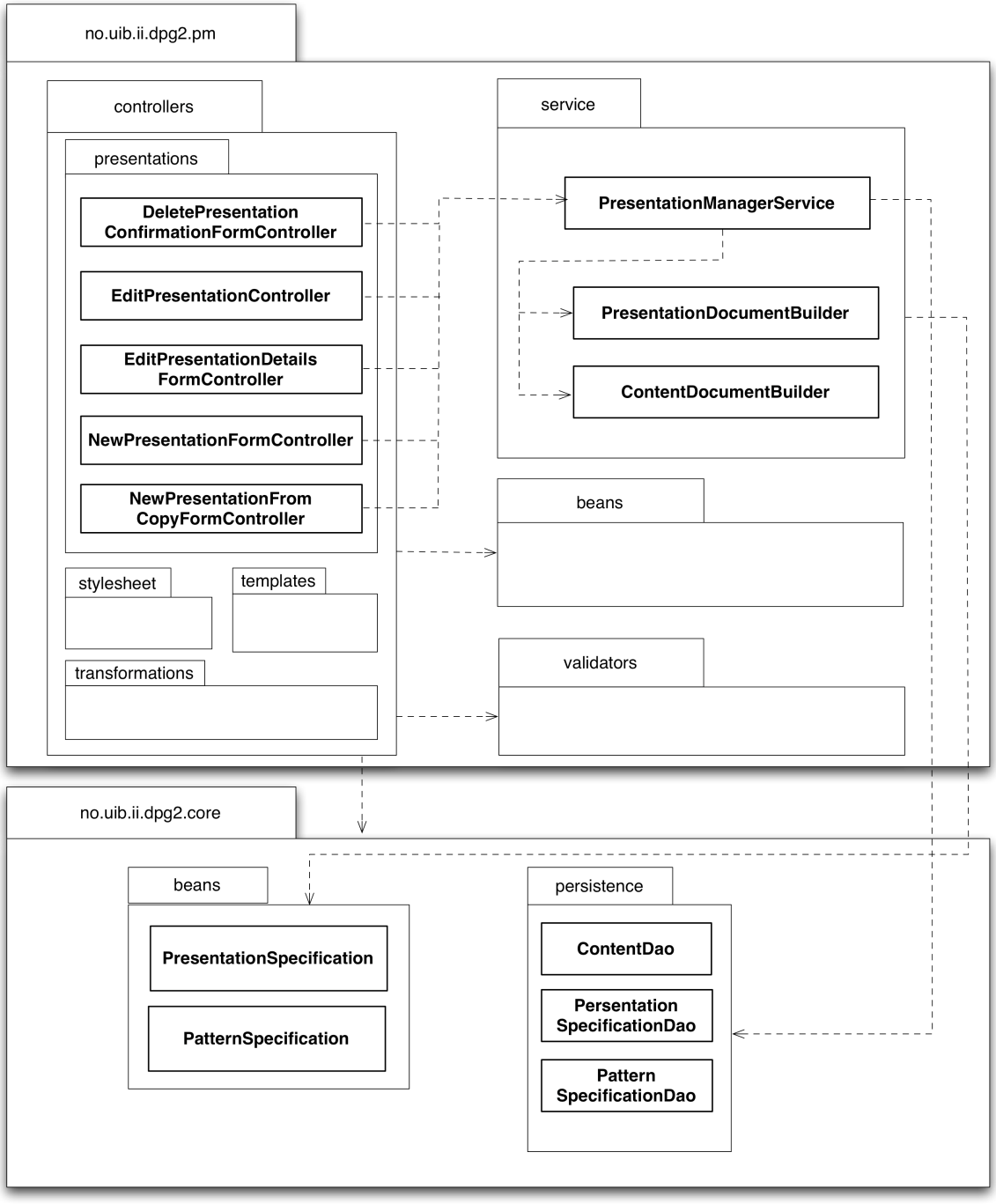
Den første pakken vi kan ta for oss, er `controllers`-pakken. Siden verktøyet inneholder en rekke funksjoner, er antall kontrollere relativt høyt, og av den grunn inndelt i forskjellige underpakker. For å unngå at diagrammet blir for uoversiktlig og vanskelig å forstå, vises kun de mest interessante klassene, nemlig dem i underpakken `presentations`. Denne inneholder kontrollere rettet mot hovedlivssyklusen til presentasjonene, nemlig opprettelse, konfigurasjon og sletting.

Det er to måter å opprette nye presentasjoner på. I det første tilfelle oppretter man en ny tom presentasjon basert på et eksisterende presentasjonsmønster. Denne presentasjonen vil da ikke ha noe innhold, noe som er ønskelig i de fleste tilfeller. I `controllers`-pakken iverksettes denne prosessen av klassen `NewPresentationFormController`, som får inn et skjema med informasjon fra bruker om hvilken presentasjon som skal opprettes.

Den andre måten å opprette nye presentasjoner på, er ved å kopiere en eksisterende presentasjon. Den nye presentasjonen vil da ikke bare benytte samme presentasjonsmønster som originalen, men også ha en kopi av det samme innholdet. Dette er svært nyttig i situasjoner der man ønsker å opprette en ny presentasjon som skal ha svært likt innhold som en eksisterende presentasjon, kun enkelte småting må endres på. For å unngå at man da manuelt må skrive av den samme informasjonen i den nye presentasjonen, man Presentation Manager gjøre dette automatisk for en administrator. I `controllers`-pakken håndteres dette av klassen `NewPresentationFromCopyFormController`, som får inn et skjema med informasjon fra bruker om hvilken presentasjon som skal kopieres.

Redigering av eksisterende presentasjoner håndteres av kontrollerne `EditPresentationController` og

`EditPresentationDetailsFormController`. Førstnevnte brukes kun for å vise en liste over de valgmuligheter som en administrator har i verktøyet for en gitt presentasjon (for eksempel konfigurere eller slette presentasjonen). Sistnevnte brukes i forbindelse med konfigurasjonsskjema, der administrator kan velge å endre på ulike innstillinger som presentasjonens tittel, beskrivelse og brukergruppe.



Figur 9-1: Oversikt over de viktigste klassene og pakkene i Presentation Manager

Den siste kontrollerklassen vi har tatt med i diagrammet er klassen `DeletePresentationConfirmationFormController`, som viser et bekreftelsesskjema for sletting av presentasjoner. For å unngå at en administrator sletter en presentasjon ved et uhell, er det her gjort noen spesielle tilpassninger utover hva som er vanlig i et HTML-skjema, noe som er nærmere forklart senere i dette kapittelet

I `service`-pakken finner vi tre klasser. Den første av dem er `PresentationManagerService`, en klasse som benyttes av de fleste klassene i kontrollerpakken. Kontrakten til denne inneholder metoder som styrer livssyklusen til presentasjonene, deriblant opprettelse, konfigurasjon og sletting av presentasjoner. Denne klassen benytter seg av de to andre service-klassene, `PresentationDocumentBuilder` og `ContentDocumentBuilder`. `PresentationDocumentBuilder` inneholder metoder for å bygge opp XML-dokumentet som utgjør presentasjonsspesifikasjonen, som beskrevet i kapittelet om det nye presentasjonsmønsteret. `ContentDocumentBuilder` bygger også opp XML-dokumenter, men i dette tilfelle for innholdsdokumentet for en gitt `entity instance`.

Mange av klassene i hovedpakken til Presentation Manager benytter seg av pakker og klasser i felleslaget (`core`), og de mest interessante av dem er tatt med i diagrammet. Spesifikasjonsklassene i beans-pakken for presentasjoner og mønstre er særdeles viktig, spesielt under opprettelse av nye presentasjoner. For å lagre nye presentasjoner og utføre endringer på dem benytter man også de viktigste datalagringsklassene som `ContentDao`, `PresentationSpesificationDao` og `PatternSpesificationDao`.

9.2 Opprette nye presentasjoner

Vi skal nå se på hvordan opprettelsen av nye presentasjoner basert på eksisterende presentasjonsmønstre skjer ved hjelp av dette administratorverktøyet. Først skal vi se på det grafiske grensesnittet for dette, deretter hva som skjer bak kulissene under denne prosessen.

En av de største problemene med den gamle DPG-en var at det ikke var noen enkel måte å opprette nye presentasjoner basert på eksisterende presentasjonsmønstre. Man måtte som regel inn i filsystemet og manuelt sette opp og konfigurere hver ny presentasjon, en prosess som krevde stor teknisk innsikt fra personen som utførte dette. I tillegg var det veldig lett å gjøre feil under konfigurasjonen, noe som resulterte i at systemet ikke starter igjen etterpå. Mye tid ble kastet bort på feilsøking og testing for å få systemet oppe å gå igjen (systemet måtte nemlig stenges ned før man kunne legge til en ny presentasjon), og dette går selvsagt ut over brukerne av de presentasjonene som allerede ligger i systemet og er i daglig bruk.

I det nye systemet derimot, skjer all instansiering av presentasjoner gjennom et lettfattelig grensesnitt, som vi vil se i dette avsnittet. Det er ikke lenger noe behov for manuell

redigering av konfigurasjonsfiler, og man trenger heller ikke sette systemet ut av drift mens opprettelsen foregår. Dermed er et av de største problemene med den gamle DPG-en løst.

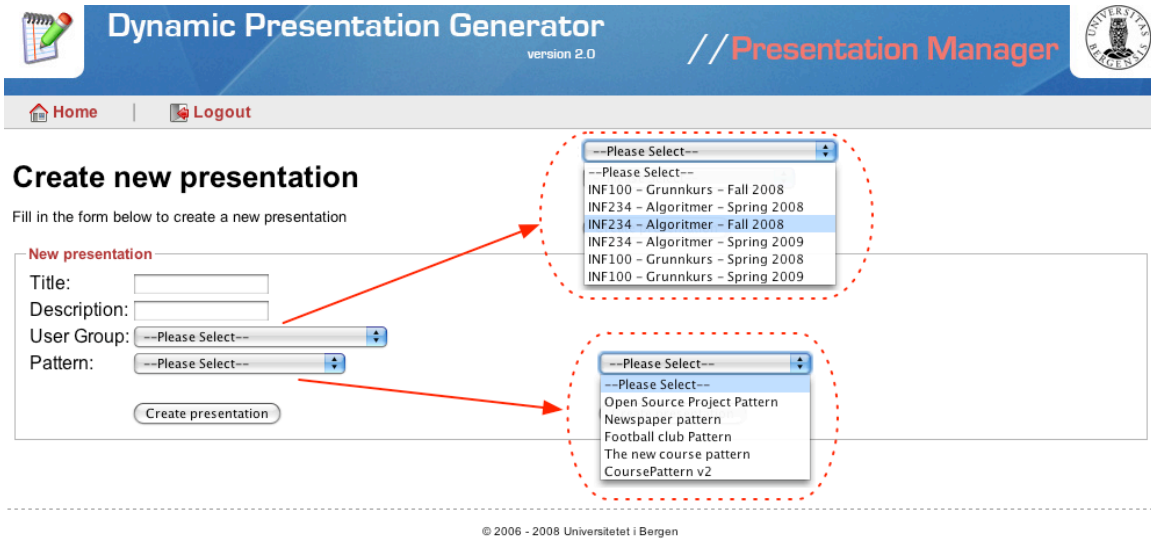
9.2.1 Hva brukeren ser

La oss nå si at en administrator ønsker å opprette en ny presentasjon basert på et eksisterende mønster allerede installert i systemet. Etter pålogging befinner man seg i lobbyen med oversikt over alle presentasjoner i systemet. I menyen øverst vil det da være egne lenker for opprettelsen av nye presentasjoner (se figur 9-2). Den første oppretter en ny presentasjon basert på et presentasjonsmønster, og det er dette som blir forklart i dette avsnittet. Det andre valget, kopiering av presentasjoner, blir forklart senere i kapittelet. Merk at disse lenkene vil kun være tilgjengelige dersom brukeren har administratorrettigheter.

Figur 9-2: Lenker i lobby for opprettelsen av nye presentasjoner

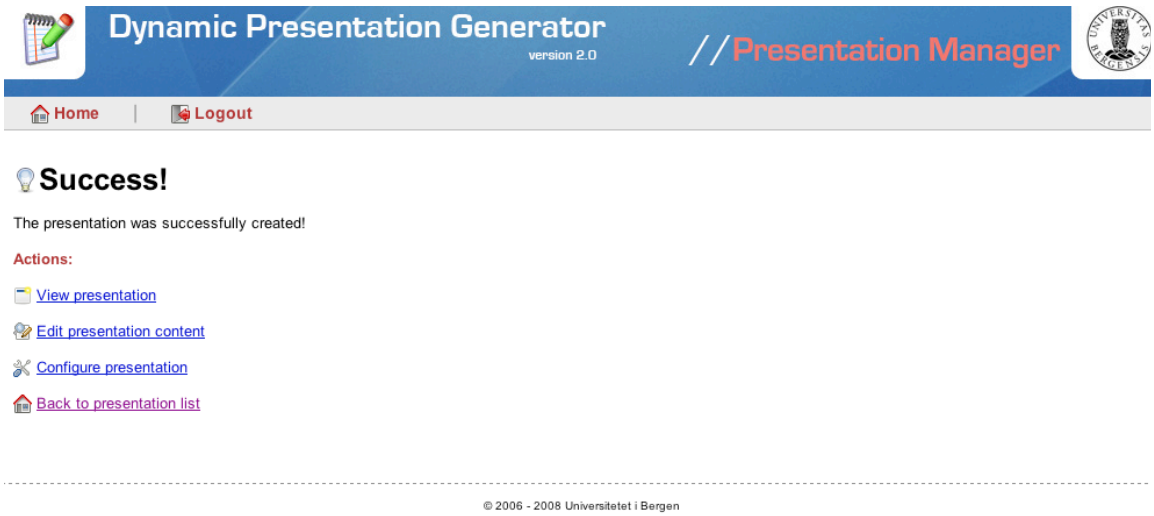
Når man som administrator så velger å opprette en ny tom presentasjon, vil man få opp skjemaet vist i figur 9-3. I dette skjema må administrator først oppgi en tittel og beskrivelse på den nye presentasjonen. Deretter velger man hvilken brukergruppe som skal ha tilgang til presentasjonen. Hvilke brukergrupper som er tilgjengelig bestemmes av det eksterne brukerhåndteringssystemet Webucator 3.0, som beskrevet i nærmere detaljer i (Løvik 2008).

Det siste og kanskje viktigste valget i skjema er hvilket av de installerte presentasjonsmønstrene som skal benyttes. Listen inneholder alle presentasjonsmønstrene som er installert. Siden dette har store konsekvenser for hvordan presentasjonen skal bygges opp, kan man ikke senere bytte til et annet mønster. Kun tittel, beskrivelse og brukergruppe kan endres i etterkant når presentasjonen først er opprettet.



Figur 9-3: Skjema for opprettelsen av en ny, tom presentasjon

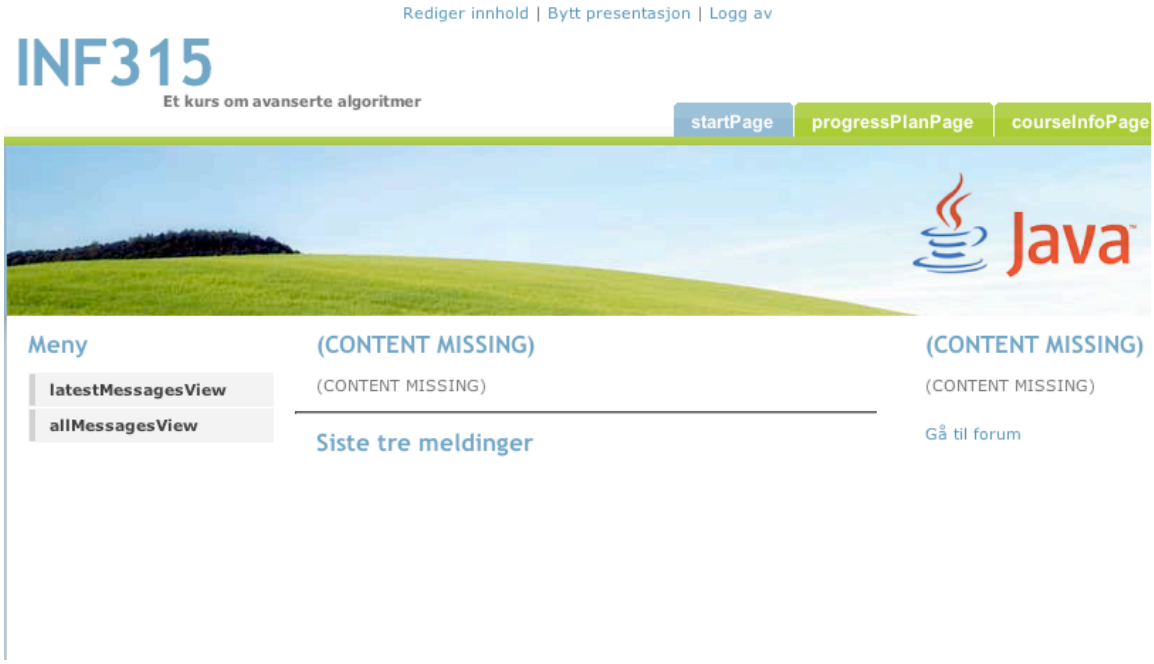
Når skjema er utfylt, tar det som regel kun få sekunder å generere og installere den nye presentasjonen. Når det er gjort, vil bruker få opp en nettside (se figur 9-4) som forteller at den nye presentasjonen ble opprettet, sammen med en rekke valg som administrator kan ta videre. Brukeren kan velge å enten se hvordan den nye presentasjonen ser ut i Presentation Viewer, redigere innholdet ved hjelp av Presentation Content Editor eller konfigurere presentasjonen ytterligere gjennom Presentation Manager. Hvis administrator ønsker å opprette flere presentasjoner, kan det enkelt gjøres ved å gå tilbake til presentasjonsoversikten og starte prosessen på nytt.



Figur 9-4: Bekreftelse på at presentasjonen har blitt opprettet

I figur 9-5 ser man hvordan en nylig opprettet presentasjon kan se ut. Som vi ser, har den ikke noe innhold foruten såkalte *placeholders* (”content missing”) satt inn av verktøyet under opprettelsen av presentasjonen. Denne midlertidige teksten indikerer de forskjellige steder der innhold forventes å komme etter hvert. Man kan nå gå inn i

redigeringsverktøyet og begynne å legge inn informasjon på de forskjellige nettsidene. Man vil trolig også endre på `page` og `view`-titler slik at lenkene får mer brukervennlige navn, da disse blir automatisk satt til å være det samme som identifikatorene i presentasjonsmønsterspesifikasjonen under opprettelsen.



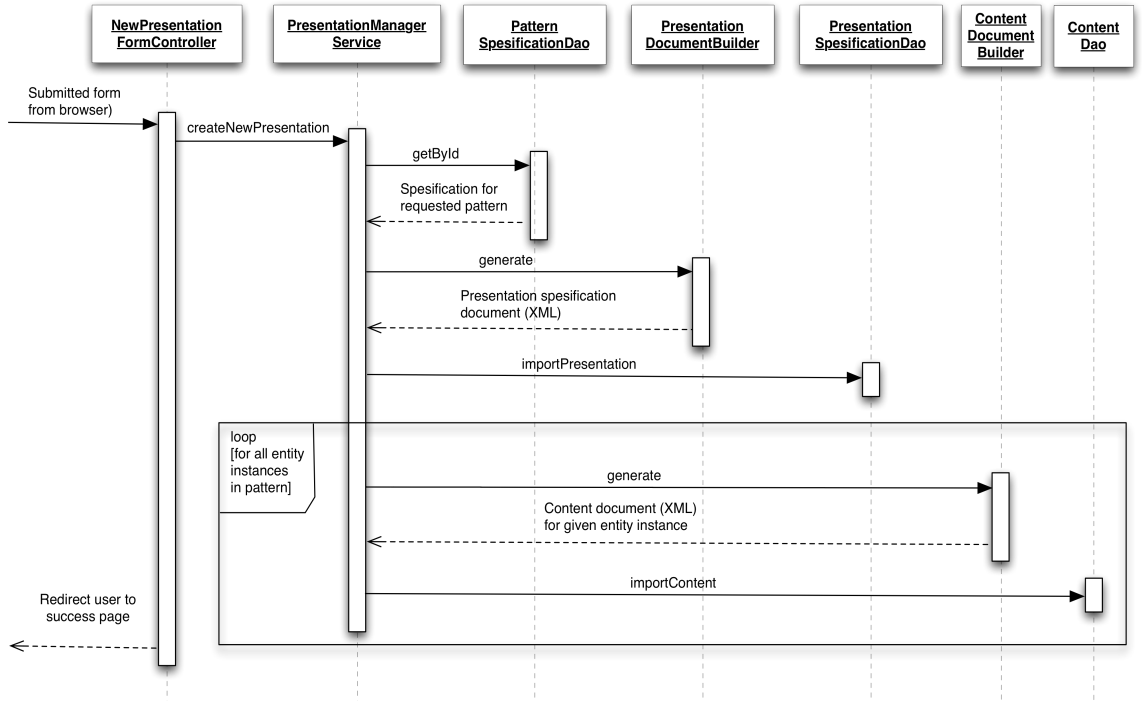
Figur 9-6: Den nye presentasjonen er nå klar til bruk

Dette er alt en administrator trenger å gjøre for å opprette en ny instans av en presentasjon basert på et eksisterende presentasjonsmønster. Sammenligner man dette med hvordan man opprettet presentasjoner i DPG 1.0, innser man fort hvor betydelig forbedret denne viktige prosessen er sett med administrators øyne.

9.2.2 Hva som skjer i systemet

Vi skal nå se nærmere på hva som skjer bak kulissene i denne prosessen. Dette gjør vi ved å se på de forskjellige klassene som er involvert og hvordan de samarbeider med hverandre for å utføre oppgaven.

I sekvensdiagrammet (Figur 9-7) ser man en oversikt de viktigste klassene som er involvert i denne operasjonen, og hvordan disse samarbeider. Det hele starter med at kontrollerklassen `NewPresentationFormController` får inn et ferdig utfylt skjema som inneholder tittel, beskrivelse, brukergruppe og identifikatoren til presentasjonsmønsteret for den nye presentasjonen. Kontrollerklassen vil da først sørge for at alle disse verdiene er angitt og lovlige, og sender forespørselen videre til `PresentationManagerService`.



Figur 9-7: Sekvensdiagram for klasser som deltar under opprettelsen av en ny, tom presentasjon

Det er klassen `PresentationManagerService` som styrer hoveddelen av denne prosessen. Det første den gjør, er å få tak i spesifikasjonen til det aktuelle presentasjonsmønsteret basert på identifikatoren gitt i skjemaet. Deretter ber den `PresentationDocumentBuilder` konstruere XML-dokumentet som inneholder presentasjonsspesifikasjonen til den nye presentasjonen. Denne spesifikasjonen sendes så til persistenslaget gjennom `PresentationSpesificationDao`.

Så blir alle innholdsdokumentene (XML-dokumenter) opprettet en for en i en løkke, en for hver `entity instance` som fins i presentasjonsmønsterspesifikasjonen. Klassen `ContentDocumentBuilder` hjelper `PresentationManagerService` med å konstruere disse XML-dokumentene, som deretter sendes til persistenslaget.

Når dette er gjort, er den nye presentasjonen ferdig opprettet og klar til bruk. `NewPresentationFormController` vil da sende tilbake en nettside som forteller administrator at det hele gikk bra, sammen med lenker til den nye presentasjonen.

9.3 Kopiere eksisterende presentasjoner

Her skal vi se på hvordan man kan kopiere eksisterende presentasjoner ved hjelp av dette administratorverktøyet. Først skal vi se på det grafiske grensesnittet for kopieringen, og til slutt hva som skjer bak kulissene under denne prosessen.

Kopiering av eksisterende presentasjoner kan være svært nyttig i mange sammenhenger, spesielt i fjernundervisningssammenheng. Her har man som en rekke kurs som går over flere semestre, og der kursene ikke varierer så veldig mye fra semester til semester i

informasjonen som ligger på nettsidene. Når en foreleser skal sette opp et nettsted et kurs som har gått et semester tidligere, og som kommer til å ha nesten identisk innhold (som regel vil ukenumre og datoer i forelesningsplanen variere), vil det være veldig tidkrevende å sette opp igjen alt på nytt. Dette var dessverre vanlig i DPG 1.0, fordi eneste alternativ var å gå inn i filsystemet, kopiere filer og konfigurere omfattende konfigurasjonsfiler for hånd, en prosess som var veldig avansert og risikofylt selv for en dreven administrator. Ikke bare risikerte man at den nye presentasjonen ikke fungerte, men man kunne også være uheldig og ødelegge for presentasjonen man kopierte fra, skulle man være så uheldig å endre på eller slette noe som man ikke burde.

I det nye systemet, derimot, er denne prosessen betydelig forenklet. Alt foregår nå i et lettfattelig grafisk grensesnitt, som vi vil straks se nærmere på.

9.3.1 Hva brukeren ser

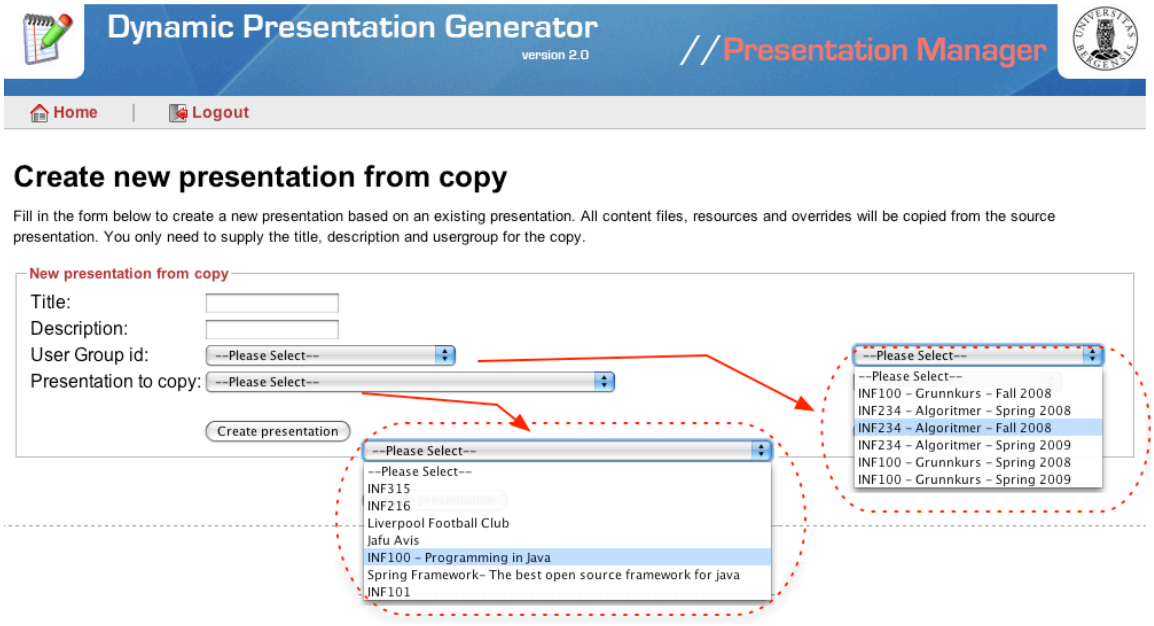
La oss igjen ta utgangspunkt i et nettkurs. I figur 9-8 har vi allerede en presentasjon for kurset INF100, våren 2008. Anta at en kursansvarlig med administratorrettigheter ønsker å holde samme kurset igjen, bare nå for høsten 2008. Det meste av presentasjonens innhold, fra kursinformasjon, informasjon på forsiden, samt fremdriftsplanen vil inneholde omtrent den samme informasjonen. Kun få oppdateringer, spesielt knyttet til datoer, vil være nødvendig.

Figur 9-8: Nettkurset som vi ønsker å basere den nye presentasjonen på

Eksempelet vi vil ta for oss her, fokuserer på kursets fremdriftsplan. Planen for det nye kurset vil ha dem samme strukturen, men selvsagt med andre ukenumre siden kurset går om høsten fremfor om våren.

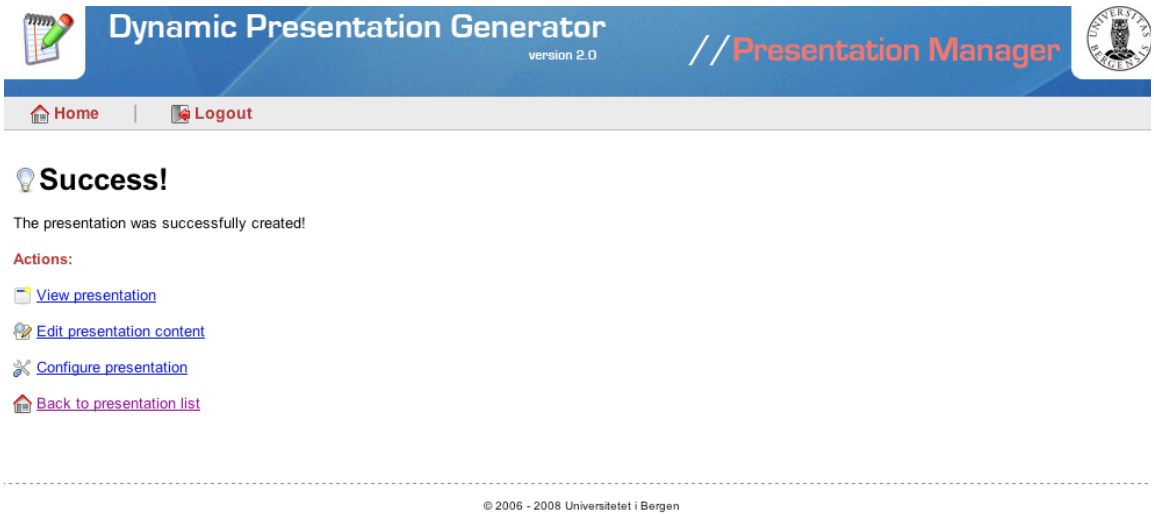
Figur 9-9: Lenke i lobby for kopiering av en eksisterende presentasjon

På samme måte som for opprettelsen av en ny, tom presentasjon, starter prosessen for kopiering av en eksisterende presentasjon i lobbyen (se figur 9-9). Administrator for da opp et skjema (se figur 9-10) der man må angi tittel, beskrivelse, brukergruppe og presentasjonen som skal kopieres. Tilgjengelige brukergrupper bestemmes som tidligere av det eksterne brukerhåndteringssystemet Webucator, mens listen over presentasjoner inneholder alle presentasjoner som fins i systemet.



Figur 9-10: Skjema for opprettelsen av ny presentasjon basert på en eksisterende presentasjon

Systemet vil da opprette den nye presentasjonen og administrator får så frem en nettside som forteller at alt gikk bra (se figur 9-11). Denne nettsiden inneholder blant annet en lenke til den nye presentasjonen, samt lenker for redigering og ytterligere konfigurasjon.



Figur 9-11: Nettside som forteller administrator av den nye presentasjonen er opprettet

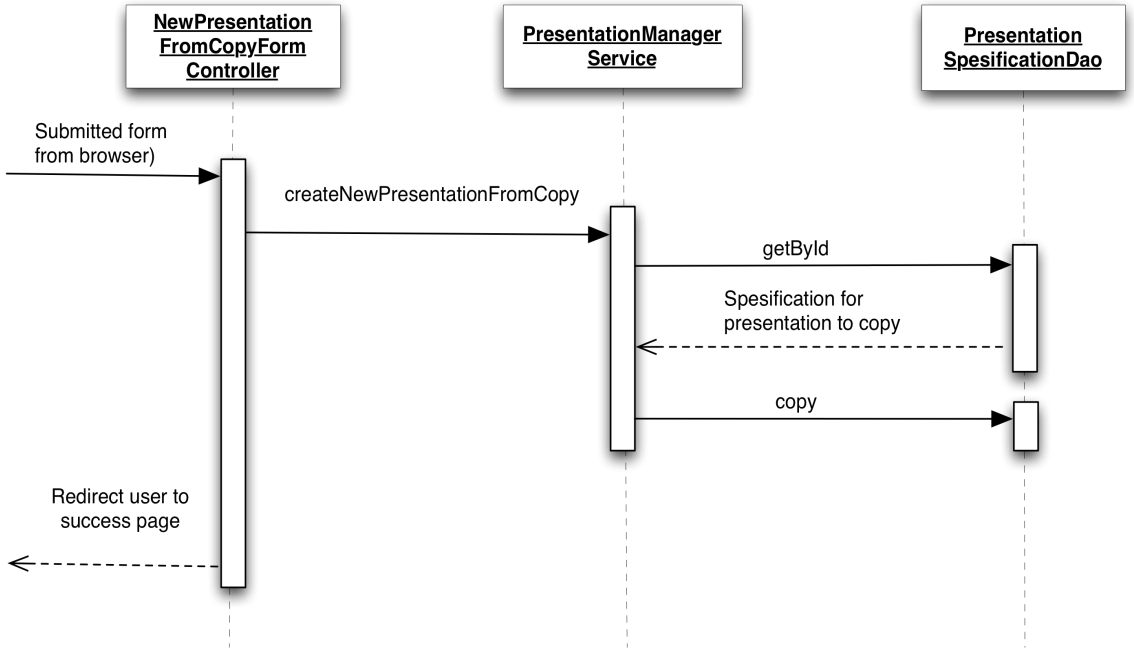
Figur 9-12: Den nye presentasjonen for høstsemesteret til kurset

I figur 9-12 ser vi hvordan den nye presentasjonen ser ut. Alt innhold, deriblant fremdriftsplanen, vil være identisk til den opprinnelige presentasjonen, foruten presentasjonens tittel og beskrivelse, som nå reflekterer at dette er høst-versjonen av kurset. Det eneste som kursansvarlig nå trenger å gjøre for å gjøre fremdriftsplanen korrekt, er å endre ukenumrene slik at de stemmer overens med høstsemesteret, og dermed er mye tid spart.

9.3.2 Hva som skjer i systemet

Vi skal nå se på hva som skjer i systemet når man skal kopiere en presentasjon. Denne prosessen er illustrert i sekvensdiagrammet i figur 9-13. Sammenlignet med opprettelsen av en ny, tom presentasjon, er denne prosessen betydelig enklere. Dette skyldes at det meste av kopieringen kan utføres på et lavere nivå, direkte i persistenslaget. Hvordan persistenslaget er implementert kan man lese mer om i (Berg, 2008).

Det hele begynner med at klassen `NewPresentationFromCopyFormController` får inn informasjonen fra skjemaet administrator har fylt ut. Den sørger så for at de påkrevde verdiene for tittel, beskrivelse, brukergruppe for den nye presentasjonen, samt presentasjonen som skal kopieres, er til stede og gyldige.

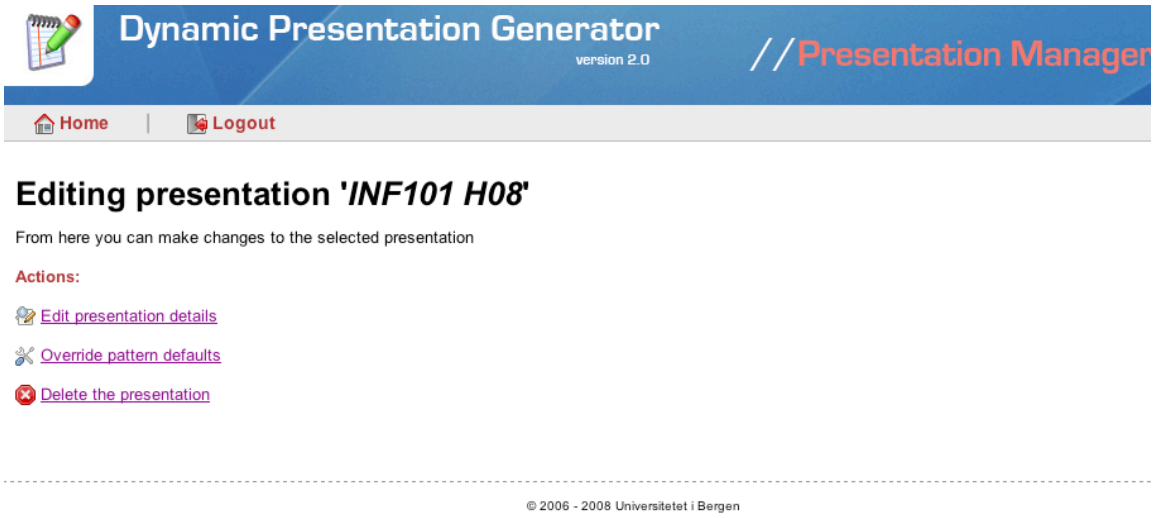


Figur 9-13: Sekvensdiagram med oversikt over de viktigste aktørene i kopieringsprosessen

Deretter sendes så disse verdiene til `PresentationManagerService`, som styrer resten av prosessen. Den finner frem til spesifikasjonen for presentasjonen som skal kopieres, og sender så all informasjonen videre til `PresentationSpesificationDao`, som utfører selve kopieringen. Til slutt blir administrator oversendt en nettside som forteller at opprettelsen av den nye presentasjonen ble fullført.

9.4 Administrasjon av eksisterende presentasjoner

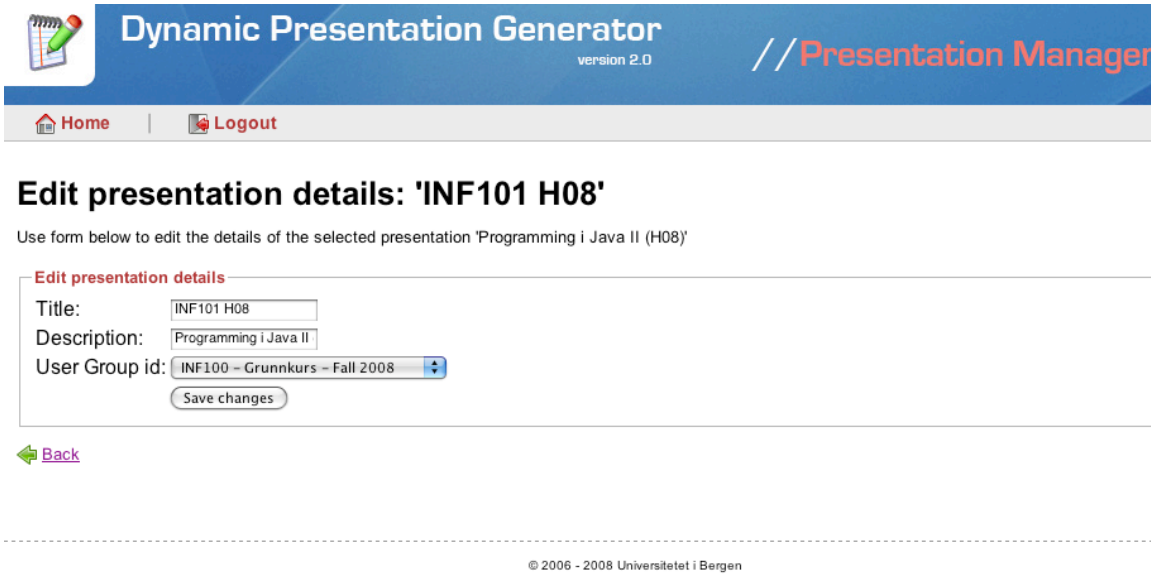
Verktøyet tilbyr også funksjonalitet for å administrere eksisterende presentasjoner. Som vist i figur 9-14 har man hovedsakelig tre forskjellige funksjoner; redigering av presentasjonsdetaljer, overkjøring av innstillinger i mønsteret, og sletting av en presentasjon. Vi vil nå se nærmere på hva disse funksjonene kan tilby.



Figur 9-14: Hovedmenyen for konfigurasjon av en eksisterende presentasjon

9.4.1 Endre konfigurasjonsinnstillinger til en presentasjon

Etter at en presentasjon først er opprettet, kan man ikke lenger endre på hvilket presentasjonsmønster som den skal benytte. Dette er fordi presentasjonsmønsteret dikterer hele datastrukturen, og informasjonen vil derfor trolig ikke være kompatibel med et annet mønster.



Figur 9-15: Endringsskjema for generell informasjon om en eksisterende presentasjon

Det man derimot kan ender på, og som figur 9-15 viser, er presentasjonens tittel, beskrivelse, og brukergruppe. Evt. endringer vil umiddelbart være synlig i presentasjonen og listen over tilgjengelige presentasjoner.

9.4.2 Overkjøre innstillinger i mønsteret

Trolig vil man ofte oppleve situasjoner der en administrator som ønsker å opprette en presentasjon, har funnet et presentasjonsmønster som passer ganske bra, *men ikke helt*, til det aktuelle formålet. For å unngå at dette fører til en eksplosjon av nesten identiske presentasjonsmønstre, har man valgt å ta i bruk konseptet overkjøring. Dette betyr at en administrator kan gå inn og overkjøre (eller overstyre) innstillinger definert av det valgte presentasjonsmønsteret for en presentasjon. Dette fører til at man har veldig stor fleksibilitet og muligheter for å skreddersy en presentasjon til de enkelte formål. Eneste ulempen er at dette krever at administrator har en god forståelse av hvordan et presentasjonsmønstrer er bygd opp, og har kjennskap til de involverte teknologiene (XHTML, CSS, Velocity Templates).

Utgangspunkt for den nye presentasjonen

For å demonstrere litt av de mulighetene som overkjøring tilbyr, skal vi nå ta for oss et konkret scenario. Igjen tar vi utgangspunkt i nettkurssammenheng, og presentasjonsmønsteret `coursePattern` som det er vist skjermbilder av i en rekke kapitler fram til nå. Anta nå at en administrator ønsker å opprette et nettkurs basert på `coursePattern` som skal brukes for et (fiktivt) biologikurs ved Universitet i Bergen. Leder for dette kurset har sett på presentasjonen som ble brukt i programmeringskursene, og ønsker nå å benytte samme mønsteret. Han får da en administrator til å opprette en ny tom presentasjon, på måten beskrevet tidligere i kapittelet, med tittelen `BIO120` og beskrivelse `Introduction to aquamedicine`. I figur 9-16 ser vi hvordan presentasjonen ser ut til å begynne med.

Figur 9-16: Viser hvordan presentasjonen ser ut rett etter at den har blitt opprettet

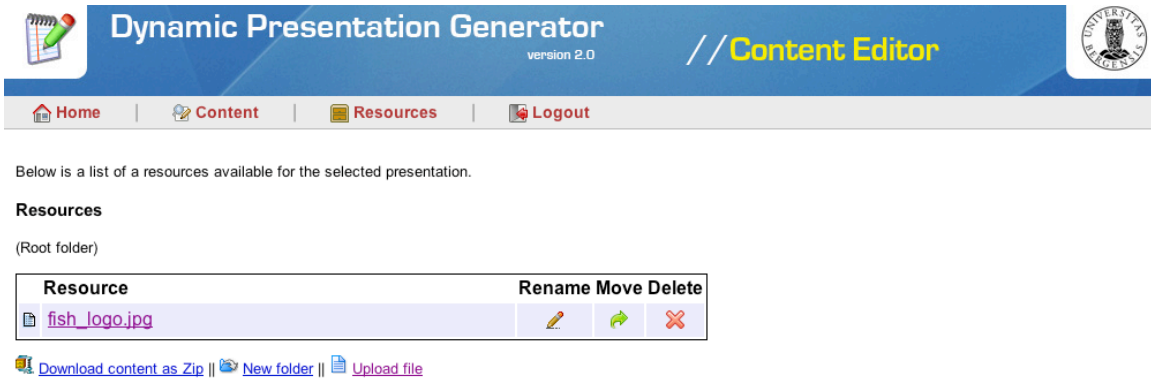
Problemer og utfordringer
Biologiprofessoren har nå en fungerende presentasjon basert på kursmønsteret. Men det er momenter i kursmønsteret som ikke er passende eller ønskelig for et biologikurs. For det første, er det tydelig at den som utviklet coursePattern hadde programmeringskurs i tankene, da den store banneren har en Java-logo i seg. Her ville det passet bedre med en banner som reflekterte det kurset handler om, og et bilde av havet og fisker ville passet bedre. Det blir derfor nødvendig å bytte ut banneren med noe annet.

Samtidig kan det være greit å endre på standardfargene slik at kurssidene for biologi skiller seg litt ut i fra programmeringskursene. For å løse dette, må man endre på stilarket (CSS) for presentasjonen.

For øvrig vil dette kurset bli tilbytt internasjonale studenter, slik at all informasjon som fins på nettsiden skal være på engelsk. Eventuelle norske titler på de forskjellige sidene må derfor endres på. I motsetning til programmeringskursene, vil det ikke i dette tilfelle bli benyttet et forum, og viewet for lenke til dette må da skjules.

Alle disse utfordringene er det fullt mulig å løse, delvis ved hjelp av overkjøring, og delvis ved hjelp av funksjonaliteten i Presentation Content Editor. Vi vil nå gå raskt gjennom denne prosessen og vise litt at det som er mulig å gjøre.

Endre på banner
Først vil vi bytte ut banneren med en annen. Administrator har laget en ny, mer fiskeorientert banner (`fish_banner.jpg`) i et grafikkprogram, med samme dimensjoner som den gamle banneren for at det skal lettere passe inn i den eksisterende layouten. Ved hjelp av ressursopplastingsfunksjonaliteten i Presentation Content Editor laster han denne filen opp til presentasjonen (se figur 9-17).



Figur 9-17: Den nye banneren er nå lastet opp som en ressurs i presentasjonen

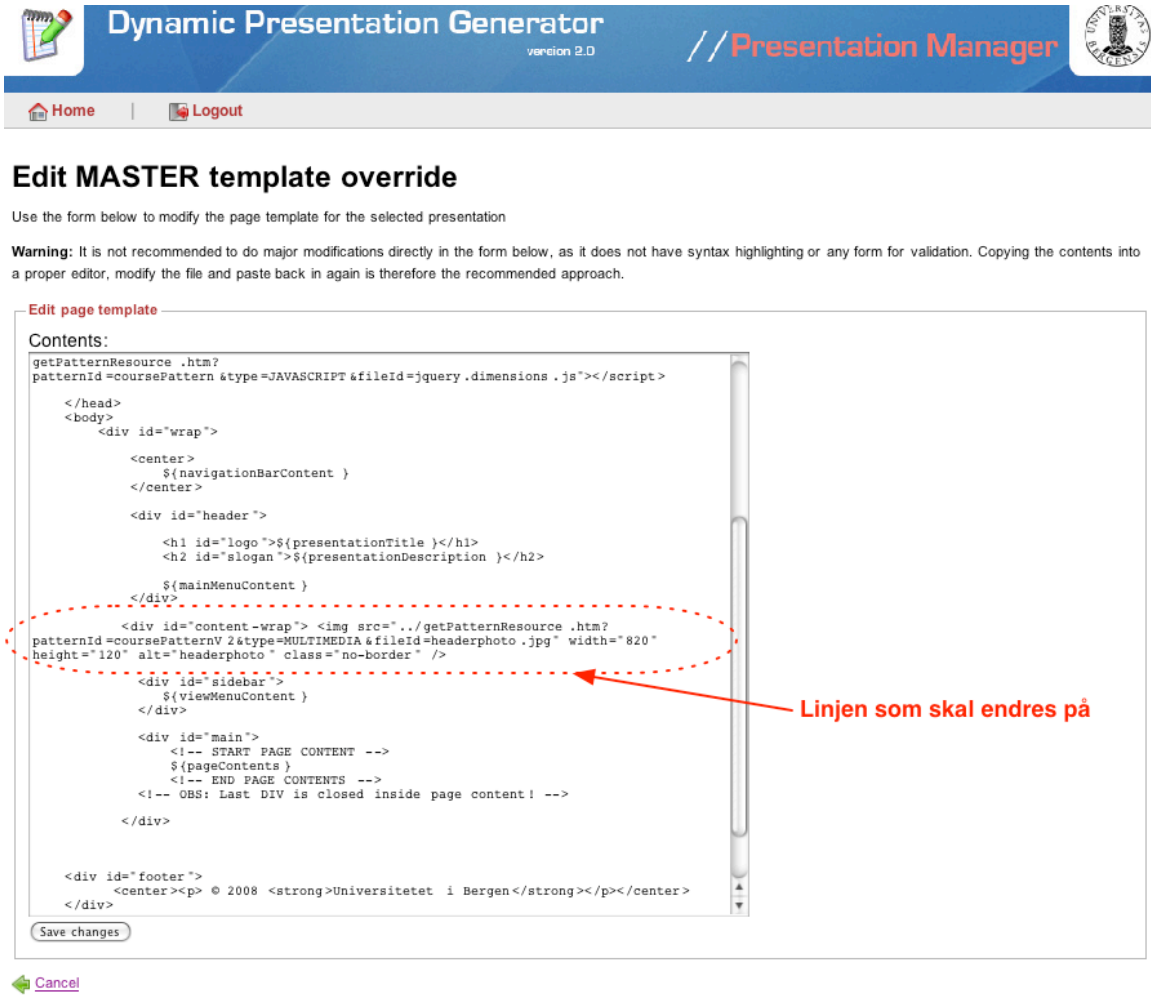
Administrator går nå tilbake til Presentation Manager og velger funksjonen for overkjøring, og får opp skjermbilde vist i figur 9-18. Det er fire forskjellige typer elementer som kan overkjøres; `page templates` (inkludert `master page template`), `master transformations`, `view transformations` og `stylesheet`. For hver type vises det en del informasjon om elementet, samt om et

element for øyeblikket er overkjørt (`overridden`), samt lenke til skjema for overkjøring.

Figur 9-18: Oversikt over de forskjellige elementene fra mønsteret som kan overkjøres

For å bytte ut banneren, må man i `coursePattern` bare endre på èn linje i `master page layout`. Administrator velger derfor å modifisere `master page template` og får opp et skjema som vist i figur 9-19. Filen som redigeres blir vist i et enkelt tekstområde, uten noen form for syntaksfargelegging eller avanserte formateringsfunksjoner. Det er derfor viktig å presisere at dersom man skal gjøre større endringer, er det bedre å klippe og lime inn innholdet i en egnet editor for den aktuelle filtypen, utføre endringen, og lime det inn igjen. I dette tilfelle er det kun èn linje som må endres på, slik at dette ikke er nødvendig.

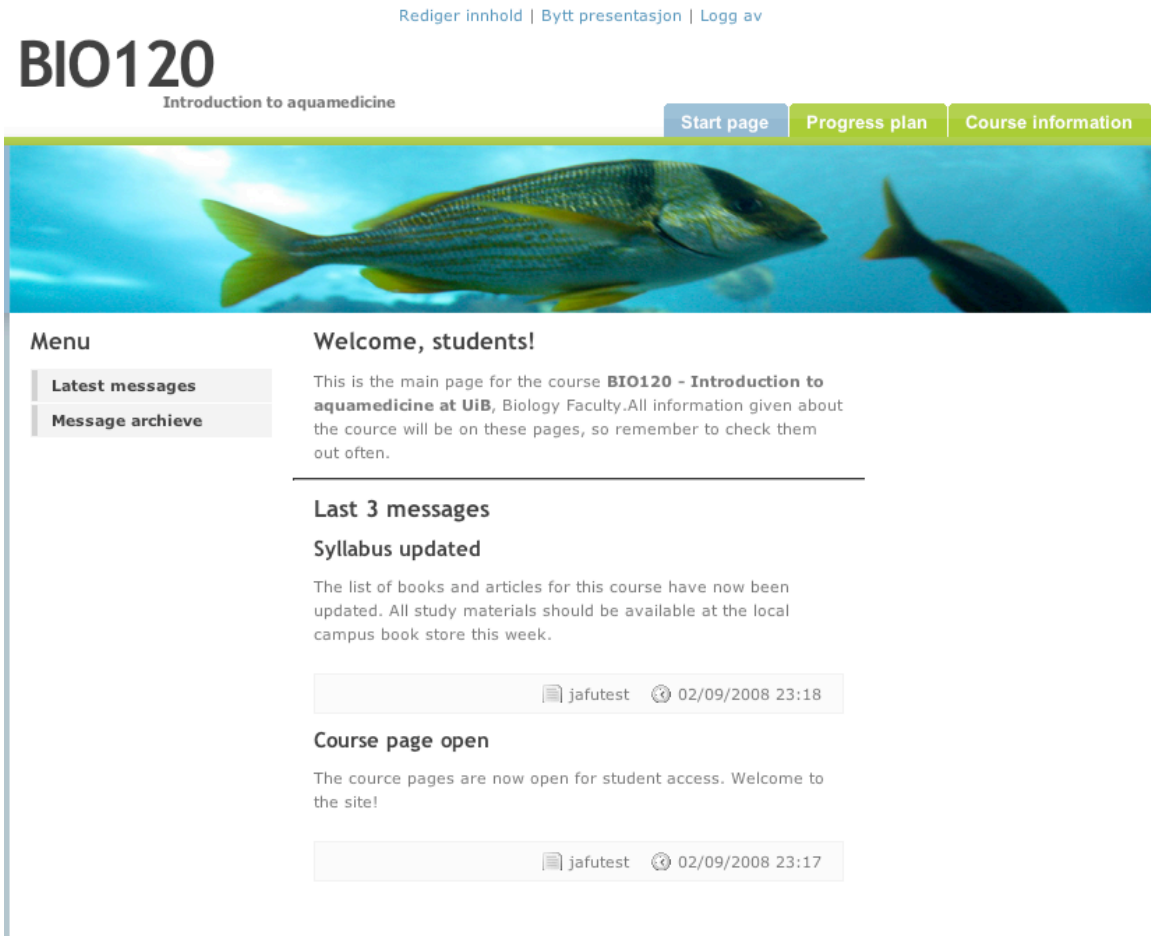
Vi ser at `<img>`-taggen bruker en url som henter ut en ressurs fra mønsteret (`getPatternResource....headerphoto.jpg`). Denne må endres slik at den henter ut *presentasjonsressursen* som vi har lastet opp. Den nye URL-en blir i dette tilfelle `../getPresentationResource.htm?pid=4683603&fileId=fish_logo.jpg`. Nærmere forklaring på hvordan disse URL-ene fungerer ser man i (K. Berg, Persisetensproblematikk i Dynamic Presentation Generator 2008).



Figur 9-19: Skjema for overkjøring av `master page template`

Øvrige endringer og den endelige presentasjonen

Når endringen så er utført og lagret, vil man i overkjøringsoversikten se at master template nå er overkjørt, og den redigerte versjonen vi bli brukt fremfor den som ligger i presentasjonsmønsteret. Nå vil den nye banneren vises i Presentation Viewer. Endringer i stilarket (CSS) kan utføres på samme måte.

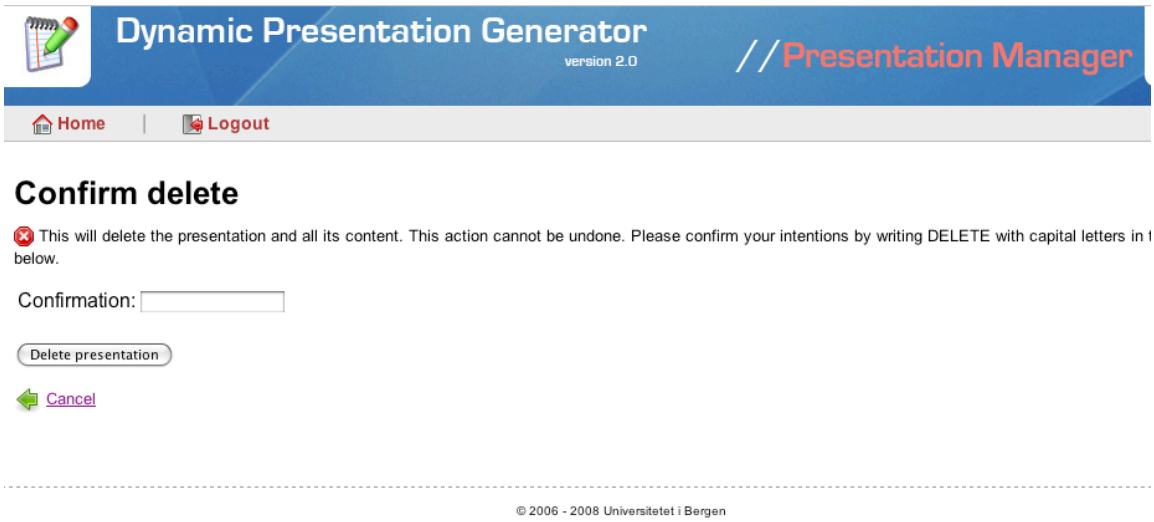


Figur 9-20: Den endelige presentasjonen etter at de ønskede endringene er utført

Figur 9-20 viser hvordan presentasjonen ser ut etter at en rekke øvrige overkjøringer og justeringer er utført; banner er byttet ut, en del farger er endret, view for forum er skjult og alle titler på nettsidene er endret til engelsk. Litt informasjon er også lagt inn for å kunne se bedre hvordan resultatet vil bli for studentene. Alle de endringene som vi ser her er gjort ved hjelp av en kombinasjon av Presentation Manager og Presentation Content Viewer.

9.4.3 Slette en presentasjon

Den tredje og siste hovedfunksjonen man kan utføre er sletting av en eksisterende presentasjon. Dette innebærer at hele presentasjonen, inkludert all informasjon som ligger lagret i den, blir fjernet for godt. Denne funksjonen er viktig for å sikre at systemet ikke blir tregere etter lengde tids bruk på grunn av at det har hopet seg opp med eldre, utgåtte presentasjoner.



Figur 9-21: Skjema for sletting av eksisterende presentasjoner

Figur 9-21 viser hvordan skjemaet for sletting av eksisterende presentasjoner ser ut. Siden dette er en såpass destruktiv operasjon, er det lagt inn et ekstra sikkerhetstiltak for å hindre at noen ved et uhell sletter en presentasjon. Administrator må skrive, med store bokstaver, nøkkelordet `DELETE`, i tekstfeltet i skjema. Kun dette nøkkelordet blir akseptert, noe som dramatisk minsker sjansen for at noen ubevist utfører en sletteoperasjon.

9.5 Hva som er oppnådd

I dette kapittelet har vi gått gjennom hovedfunksjonaliteten til administratorverktøyet Presentation Manager. Sammenliknet med Repository Administration Tool (RAT) i DPG 1.0, er det en rekke forbedringer så vel som mange nyskapende egenskaper, som vi vil oppsummere i dette avsnittet.

9.5.1 Enklere opprettelse av presentasjoner

Sammenliknet med det gamle systemet er det nærmest trivielt å opprette nye presentasjoner, enten det er basert på installerte presentasjonsmønstre eller eksisterende presentasjoner. Før var dette en vanskelig og tidkrevende prosess som krevde av systemet måtte stenges ned mens dette foregikk. Nå kan man raskt og enkelt legge til nye presentasjoner uten å merkbart forstyrre driften av de øvrige presentasjonene i systemet.

9.5.2 Kopiering av eksisterende presentasjon

Det å kunne opprette en presentasjon basert på en eksisterende, kan som vi har sett i flere tilfeller spare både `administrators` og `publishers` for mye unødvendig manuelt arbeid. Dette var en mye etterspurt funksjonalitet for DPG-systemet i forbindelse med fjernundervisningen, og vil nok kunne være nyttig også i flere andre formål enn akkurat nettkurs.

9.5.3 Håndtering av livssyklus

Presentation Manager håndtere hele livssyklusen til en presentasjon, fra opprettelsen, til modifikasjoner og overkjøringer, til sletting når presentasjonen ikke lenger trengs. Det er

mulig å endre på enkelte detaljer selv etter at en presentasjon først er opprettet. Man trenger heller ikke være redd for å slette en presentasjon ved et uhell pga det noe utradisjonelle, men effektive sletteskjema. Alt dette, foruten overkjøring, foregår gjennom et lettfattelig grensesnitt, og de fleste operasjoner utføres på sekunder med noen få museklikk. Dette skiller seg betraktelig i forhold til Repository Administration Tool i DPG 1.0, og dette verktøyet er derfor et stort sprang som gjør hverdagen til en administrator betydelig lettere.

9.5.4 Overkjøring av mønsterelementer

Som vi også har sett, gir funksjonaliteten for overkjøring av elementer fra presentasjonsmønster rom for betydelige tilpasninger. Dette bidrar til å begrense tendensen til at det utvikles en mengde nesten identiske presentasjonsmønstre. I stedet kan hver enkelt presentasjon selv gjøre de nødvendige kosmetiske endringene og på den måten skreddersy presentasjonen til hver enkelt situasjon.

9.5.5 Fleksibel pakkestruktur

Presentation Manager har en pakkestruktur som er veldig lik de to øvrige delsystemene, Presentation Viewer og Presentation Content Manager. Dette er en ganske fleksibel struktur som det burde være lett å vedlikeholde og utvide i fremtiden, skulle dette være nødvendig. Grensesnitt (interfaces) er brukt mellom lagene der dette var fornuftig, slik at det skal være mulig å bytte ut implementasjoner uten at dette får store konsekvenser for de øvrige lagene.

10 Forslag til videre arbeid

I dette kapittelet vil se nærmere på en rekke forslag til videre arbeid på DPG 2.0. Dette er basert på tanker og ideer som kan være nyttig for fremtidige studenter å se nærmere på i årene som kommer. Det kan godt tenkes av noen av forslagene som presenteres her kan gis som prosjekt i et kurs, eller inngå i eller danne grunnlaget for fremtidige masteroppgaver.

10.1 Utvikling av nye presentasjonsmønstre

Under utviklingen av DPG 2.0 ble det samtidig utviklet en rekke forskjellige presentasjonsmønstre. De fleste av disse ble kun utviklet for å teste ut konkret funksjonalitet i systemet og gi en demonstrasjon av hva som er mulig å få til i DPG 2.0. De var aldri ment for å være komplette, ferdige presentasjonsmønstre ment for produksjon.

10.1.1 Kursmønster

Et naturlig steg videre vil derfor være å utvikle mer målrettede presentasjonsmønstre som kan brukes i produksjon. Siden DPG 1.0 hovedsakelig ble brukt i fjernundervisningssammenheng ved instituttet, vil man trolig ønske å videreutvikle utkastet til det nye `coursePattern` som fins i DPG 2.0 i dag. Det bør brukes bedre tid på å finne ut nøyaktig hvilken informasjon det er behov for, og komme opp med et bedre grafisk utseende slik at det gir et mer seriøst inntrykk. Trolig kan det være fornuftig å se på stilene som er brukt i en rekke andre nettsider tilknyttet universitet og la nettkurssidene få et tilsvarende utseende, slik at studentene får et mer helhetlig inntrykk av de forskjellige nettstedene som de må forholde seg til gjennom et semester.

Universitet tilbyr for øyeblikket kurset `INF219 – Prosjekt i programmering.`, et ganske fritt prosjektkurs som gir 10 studiepoeng. Dersom man ønsker å sette DPG 2.0 i drift allerede til vårsemesteret 2009, kan det derfor være lurt å tilby videreutviklingen av kursmønsteret som en prosjektoppgave til en eller to studenter allerede høsten/vinter 2008.

10.1.2 Realisere presentasjonsmønstre fra Ø. Larsens masteroppgave

I sin masteroppgave beskriver Øystein Larsen (Lervik Larsen 2006) en rekke mønstre, foruten nettkurs, som det kan være interessant å forsøke å realisere i DPG 2.0.

Slideshow-mønster

Det første av disse er et såkalt slideshow-mønster, som legger til rette for å publisere faglitteratur og artikler på nettet utenom de tradisjonelle, lukkede formatene som ellers er vanlig (MS Word/PowerPoint). `Publishers` vil da kunne laste opp eller legge inn informasjonen gjennom Presentation Content Editor, som så blir tilgjengelig for de forskjellige brukerne. Det legges også opp til at brukerne skal kunne utføre søk i informasjonen som ligger der, noe som da evt. vil kreve et plugin, da DPG 2.0 for øyeblikket ikke har støtte for fulltekstsøk. Et tilsvarende arbeid ble gjort i forbindelse med SEVU-prosjektet (K. Berg, Elektronisk Kompendium 2006) for DPG 1.0, og man vil

trolig kunne hente mye inspirasjon fra eller til og med gjenbruke noen av koden fra dette prosjektet for å realisere slideshow-mønsteret i DPG 2.0.

Nettavis-mønsteret
Mulighetene for å benytte DPG 2.0 som motor for en nettavis for en liten til mellomstor virksomhet er allerede til stede. Bjørn Ove Ingvaldsen har allerede demonstrert dette med sitt eksperimentelle nettavismønster i sin masteroppgave (Ingvaldsen 2008). Dette presentasjonsmønsteret viser at DPG 2.0, så vel som konseptet presentasjonsmønster, ikke bare fungerer i nettkurssammenheng, men kan også fungere bra i en helt ulik, kommersiell setting. Mye av det som Øystein L. Larsen beskriver i sin oppgave (Lervik Larsen 2006) knyttet til nettavis i sin oppgave er allerede oppfylt, og trolig vil det ikke være noe problem å benytte DPG 2.0 til en mindre nyhetsavis.

Eneste hindring vil være at DPG 2.0 per dags dato krever at man i det minste er en registrert reader, det vil si at anonym tilgang til presentasjonene ikke er tillatt, noe som trolig vil være en nødvendighet for en ordinær nettavis. Dette kan i midlertidig løses ved at man innfører en fjerde rolle, `anonymous reader`. En administrator kan da konfigurere en presentasjon til å støtte denne nye rollen, og dermed unngå påloggingsskjema for vanlig presentasjonstilgang. Da vil alle som vil kunne lese innholdet i den aktuelle presentasjonen, samtidig som det fortsatt kreves pålogging og de nødvendige rettighetene for å endre på innholdet.

Barnehagemønsteret
Det siste forslaget fremmet i (Lervik Larsen 2006) er å bruke DPG-systemet som motor for nettstedet til en barnehage eller tilsvarende institusjoner. Her påpekes også at denne type foretak gjerne har begrensede økonomiske midler, noe som gjør at disse er en ideell potensiell kunde for et lavkostsystem som DPG er ment for å være.

Med DPG 2.0 vil det trolig ikke være store utfordringer med å lage presentasjonsmønstre rettet mot denne type foretak. De nye verktøyene, spesielt Presentation Content Editor, gjør at selv personer med relativt grunnleggende datakunnskaper kan, med kort opplæring, kunne håndtere all informasjonen for slike nettsider. Da vil for eksempel lederen for barnehagen selv kunne holde informasjonen oppdatert, noe som er både tids og kostnadsbesparende.

Det er fullt mulig å tenke seg andre virksomheter, utover barnehage, som kan dra stor nytte av et lavkostsystem som DPG. Blant dem kan man nevne skolefritidsordninger (SFO), førskoler, musikkorps, idrettslag, turlag, og sikkert mange, mange flere.

10.1.3 Øvrige presentasjonsmønstre

Det blir også viktig å fortsette å utforske helt nye typer presentasjonsmønstre, gjerne mønstre som krever spesielle funksjoner eller tilpasninger som det ikke er åpenbart at DPG 2.0 har støtte for. Dette vil være viktig for å kartlegge muligheter og begrensninger som fins i det nye systemet, og danne grunnlag for modifikasjoner og videre utvikling.

10.2 Predefinerte `Themes` for presentasjoner

Etter hvert som funksjonaliteten for overkjøring av elementer fra presentasjonsmønsterspesifikasjonen ble utarbeidet, dukket det opp muligheten for en spennende fremtidig funksjonalitet som det kan være verdt å utforske videre. Det dreier seg om at den som utvikler presentasjonsmønsteret samtidig utvikler en rekke forskjellige `themes`, det vil si varianter av hvordan presentasjonsmønsteret kan se ut. En administrator som oppretter en presentasjon på bakgrunn av dette mønsteret kan deretter velge mellom de forskjellige `themes` som er tilgjengelig, og dermed velge det predefinerte utseende som passert best til presentasjonens formål.

Figur 10-1: Et utkast til hvordan nettsiden for valg av predefinerte utseender (`themes`) kan se ut

Et `theme` bygger videre på funksjonaliteten for overkjøring av elementer, og består rett og sett av en samling av overkjørte elementer (`page templates`, CSS-stilark,

transformasjoner, etc.) fra presentasjonsmønsterspesifikasjonen som kan installeres gjennom et enkelt brukergrensesnitt, uten manuell redigering av spesifikasjonselementer. I figur 10-1 ser man et forslag på hvordan nettsiden for valg av `themes` kan se ut i Presentation Manager. For hver av de predefinerte `themes` har man en radioknapp, en forhåndsvisning som gir et inntrykk av hvordan presentasjonen vil bli seende ut, samt en kort beskrivelse av dette.

Hvordan de forskjellige `themes` utvikles og installeres er ikke tilstrekkelig utarbeidet enda, da `themes` er noe som vi ikke fikk tid til under utviklingen av DPG 2.0. Men siden det egentlig bare er en utvidelse av funksjonaliteten rundt overkjøring, vil det trolig være relativt uproblematisk å implementere også denne funksjonaliteten i en fremtidig utvidelse av DPG 2.0.

10.3 Verktøy for utvikling av nye presentasjonsmønstre

Det har over lengre tid vært diskutert om det er mulig å utvikle et verktøy som kan bistå utviklere av nye presentasjonsmønstre. Det er nemlig ikke til å legge skjul på at selv om DPG 2.0 gjør det mye enklere å opprette og administrere presentasjoner, er selve utviklingen av en ny presentasjonsmønsterspesifikasjon og alle dens elementer en nokså krevende jobb.

10.3.1 Hvordan presentasjonsmønstre utvikles per dags dato

Utviklingsprosessen foregår i mer tradisjonelle verktøy egnet for de forskjellige teknologiene som er involvert. Man trenger blant annet et godt verktøy for redigering av XML og XSLT, og helt et som har mulighet for å prøvekjøre transformasjonsfilene slik at man kan se at XML transformeres til XHTML på korrekt måte. For å utvikle de presentasjonsmønstrene som DPG 2.0 inneholder i dag, er det brukt en rekke forskjellige verktøyer, deriblant Oxygen Editor (Oxygen XML Editor u.d.), Eclipse IDE (Eclipse Project u.d.) og IntelliJ IDEA (JetBrains IntelliJ IDE u.d.).

Dersom man forsøker å utføre transformasjoner som inneholder feil eller mangler direkte i DPG-systemet, kan dette føre til systemkræsj som kan påvirke stabiliteten til server og dermed forstyrre de øvrige presentasjonene som er i drift på serveren. Det er derfor viktig at man grundig prøvekjører nye presentasjonsmønstre på testserver som kjører DPG 2.0 for å unngå å utsette produksjonsserveren for uferdige og ustabile presentasjoner.

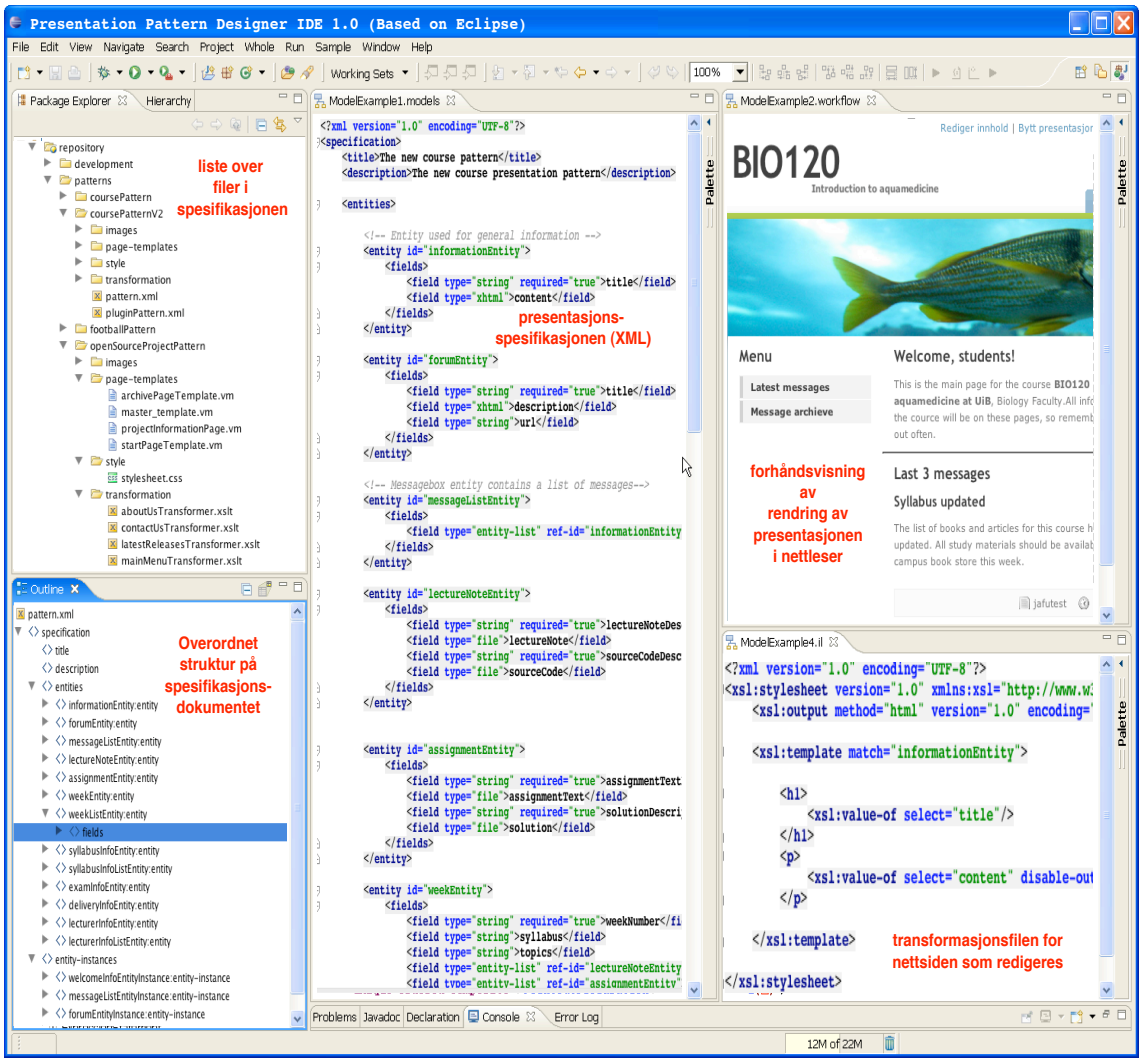
Hovedproblemet med bruk av forskjellige eksterne verktøy er at de har kun forståelse for de konkrete teknologiene som er involvert i utviklingen av presentasjonsmønstre, og mangler naturlig nok forståelsen av hvordan disse skal interagere i DPG 2.0-sammenheng. Dette gjør at man ikke får testet ut for eksempel hele rendringen av en nettside fra disse verktøyene.

I stedet må man hver gang deploye hele presentasjonsmønsterspesifikasjonen og alle dens elementer, og prøve dette ut i DPG 2.0-systemet manuelt. Har man for eksempel et sted referert til iden til et `view` som ikke eksisterer (for eksempel fordi man var uheldig og stavet navnet feil) vil man ikke oppdage dette før etter et systemkræsj og tidkrevende

feilsøking i systemloggen for å finne ut hvor det gikk galt. Dette er selvsagt en ganske tidkrevende og ofte frustrerende prosess for utviklerne av nye presentasjonsmønstre.

10.3.2 Mulighet for et IDE for Presentasjonsmønsterutvikling

En mulig løsning på dette problemet kan være å utvikle et skreddersydd IDE (Integrated Development Environment), kalt for eksempel **Presentation Pattern Designer IDE**, som inneholder all funksjonalitet som kreves for å utvikle, teste, og installere et nytt presentasjonsmønster. Flere av de mest omfattende Java-baserte IDE-ene, deriblant Eclipse og NetBeans, er begge plattformer som kan utvides og skreddersys til å utføre spesialiserte utviklingsoppgaver. Dette er mulig fordi de er alle utvidbare gjennom moduler og plugins, og disse mulighetene er godt dokumentert i en rekke bøker og informasjon fritt tilgjengelig på nettet.



Figur 10-2: Hvordan man kan tenke seg av Presentation Pattern Designer IDE kan se ut

I figur 10-2 ser vi en såkalt grafisk `mockup` (utkast) til hvordan man kan tenke seg at et verktøy for opprettelsen av nye presentasjonsmønstre kan se ut. Her har man

implementert forskjellige vinduer som har spesialiserte oppgaver. Øverst til venstre har man den tradisjonelle listen over filer som hører til det nåværende prosjektet. Her vil man kunne se alle de forskjellige komponentene, sortert etter type, som er involvert i en presentasjonsmønsterspesifikasjon. Man har selve spesifikasjonsdokumentet (XML), en rekke transformasjoner (XSLT), `page templates` (Velocity Templates), samt stilark og tilhørende grafikk.

Direkte under dette ser vi en struktur over det aktive dokumentet, det vil si det som vises i hovedvinduet i midten av verktøyet. Dette er svært vanlig i de fleste IDE-er og er en nyttig funksjon som lar utvikler lettere holde oversikt og navigere rundt i det aktive dokumentet.

I midten ser vi det aktive dokumentet, det som er åpnet for redigering. Eclipse har innebygget støtte for de fleste teknologier, deriblant XML, XSLT, og gjennom plugins, også Velocity og CSS. Dette gjør at man vil få korrekt syntaksfargelegging og intelligent hjelp (såkalt `code completion`) som gjør det raskere å skrive kode. Denne intelligente hjelpen kan og bør også utvides slik at den har en grunnleggende forståelse av hvordan de forskjellige teknologiene i DPG-en benyttes. Det vil si at dersom man i en `page template` refererer til et `view`, vil editoren sørge for at man skriver korrekt view id ved å sjekke den opp mot alle `views` i spesifikasjonsfilen. Dersom dette ikke er korrekt, vil linjen som inneholder feilen bli markert med en rød stiplet linje, på samme måte som feil angis i vanlig javakode. Verktøyet vil hele tiden sørge for at alle koblinger mellom de forskjellige elementene i spesifikasjonen er korrekte og fri for feilstavinger. Dermed vil man kunne eliminere en av de mest vanligste og frustrerende feil som begås under utvikling av presentasjonsmønstre med dagens verktøy.

Nederst til høyre har man åpent et sekundært editorvindu, som i dette tilfelle viser en XSLT-transformasjon som benyttes til å rendre et bestemt `view` på en `page`. De fleste editorer har innebygget støtte for å nesten ubegrenset menge åpnede vinduer samtidig, som gjør det mer effektiv dersom man har tilgang på større (eller flere) dataskjermer.

Det siste, men kanskje mest spennende vinduet, ser man øverst til høyre. Det er en innebygget nettleser (de fleste IDE-er har innebygget eller mulighet for å installere en forenklet nettleser direkte i IDE-et) som gir en forhåndsvisning av hvordan en presentasjon basert på presentasjonsmønsteret vil se ut for sluttbruker.

I et slikt verktøy bør det heller ikke lenger være nødvendig for utvikler å skrive XML-dokumentet for selve presentasjonsmønsterspesifikasjonen for hånd. Her kan man dra nytte av såkalte veivisere, som guider utvikler gjennom stegene for å opprette de forskjellige elementene spesifikasjonen består av (`entities`, `entity-instances`, `views`, `pages`). Veiviseren vil sørge for at alle koblinger mellom de forskjellige elementene er korrekte, og bruk av dropdown-menyer og avkrysningsbokser for felttyper og de øvrige attributtene (som `required` eller `default`) hindrer gal bruk og potensielle skrivefeil.

11 Evaluering og konklusjon

I dette kapittelet vil vi først og fremst foreta en evaluering av de ulike målsetningene som ble presentert i kapittel 1. Vi vil i tur og orden gjennomgå samtlige hoved- og delmål, og til slutt gi en kort konklusjon av oppgaven.

11.1 Evaluering av målsetninger

Her vil vi gå gjennom hver av de fem hovedmålene for oppgaven, og trekke inn eventuelle delmål der dette er relevant.

11.1.1 Design av den nye presentasjonsmønsterspesifikasjonen (Hovedmål 1)

En sentral del av oppgaven var å foreslå en helt ny presentasjonsmønsterspesifikasjon og samtidig gi en implementasjon denne for den nye versjonen av systemet. En viktig inspirasjonskilde var arbeidet utført av Alessandro Rossini og Graziano Liberati (Rossini & Liberati, 2005). Til tross for at den nye spesifikasjonen avviker betydelig fra deres forslag, bidro deres arbeid i stor grad til utformingen av denne.

Den nye presentasjonsmønsterspesifikasjonen har en rekke fordeler sammenlignet med den man finner i DPG 1.0. Ikke bare er selve spesifikasjonen lettere å forstå og jobbe med, men den er også mer fleksibel, spesielt med tanke på gjenbruk av entiteter. Samtidig gjør funksjonaliteten for overkjøring av elementer i presentasjonsmønsterspesifikasjonen det mulig å skreddersy hver presentasjon til det enkelte formål. Med bakgrunn i dette og det som kommer fram i kapittel 6 (Presentasjonsmønstre i DPG 2.0), er min oppfatning at dette hovedmålet har blitt oppnådd.

11.1.2 Design og implementering av dynamisk presentasjonsmotor (Hovedmål 2)

Med en ny implementasjon av presentasjonsmønsterspesifikasjonen på plass, var det også et behov for en ny motor for å rendre de forskjellige presentasjonene, en motor som tilsvarer Dynamic Presentation Engine (DPE) i DPG 1.0. Den nye motoren, Presentation Viewer (PV), ble bygget opp rundt den nye presentasjonsmønsterimplementasjonen og har en rekke forbedringer i forhold til DPE-en.

Den nye motoren er bygget opp på en mer modulær måte, som gjør det lettere å teste, vedlikeholde og videreutvikle. Dette står i sterk kontrast til DPE-en, og mye av dette er takket være bruken av Spring-rammeverket (The Spring Framework, SpringSource). Økt sikkerhet gjennom bruk av Acegi-rammeverket (Løvik, 2008), samt funksjonaliteten for lasting av overkjørte elementer fra presentasjonsmønsterspesifikasjonen, er bare noen av de mange momentene som bidrar til å gjøre Presentation Viewer til et viktig delsystem i DPG 2.0. Dette er nærmere beskrevet i kapittel 7 (Presentation Viewer), og dette hovedmålet er etter min mening også oppnådd.

11.1.3 Utvikling av redigeringsfunksjonalitet for presentasjonsinnhold (Hovedmål 3)

En annen viktig forbedring som var ønskelig i forhold til DPG 1.0 var bedre støtte for innholdsredigering. Mens man før måtte redigere XML-dokumenter for hånd, med alle de problemer dette medførte, går innholdsredigeringen som en lek i DPG 2.0. Nå foregår all redigering i lettfattelige HTML-skjema, som ikke krever noen spesiell opplæring (for eksempel kjennskap til XML og HTML) eller datakunnskaper utover det som regnes som vanlig bruk av datamaskiner. Dette gjør at et større spekter av personer kan være ansvarlig for innholdet på en presentasjon (altså ha rollen som `publisher`), noe som kan bidra til utstrakt bruk av DPG 2.0.

Sammenligner man det dedikerte delsystemet Presentation Content Editor (PCE) i DPG 2.0 med innholdsredigerings- og administrasjonsverktøyet Repository Administration Tool (RAT) i DPG 1.0, ser man helt klare forbedringer, først og fremst med tanke på brukervennlighet. Kapittel 8 (Presentation Content Editor) er dedikert til dette viktige delsystemet, og det som avdekkes der viser at også dette hovedmålet er oppnådd.

11.1.4 Utvikling av verktøy for å opprette og administrere presentasjoner (Hovedmål 4)

Det siste verktøyet som man ville ha på plass i DPG 2.0, var et system for å opprette og administrere presentasjoner. Dette verktøyet var ment for å brukes av personer med rollen som `administrator`, der hele livssyklusen til en presentasjon kan styres fra.

Et stort problem i det gamle systemet var at dersom man ønsker å opprette nye presentasjoner basert på eksisterende presentasjonsmønstre, måtte man som regel manuelt sette opp de ulike konfigurasjonsfilene og avhengighetene mellom dem. Dette var ikke bare tidkrevende og lite hyggelig affære, men det var også lett å gjøre små feil som førte til at systemet ble ustabilt eller gikk ned.

I det nye administrasjonsverktøyet i DPG 2.0, Presentation Manager (PM), er ting mye enklere. Opprettelsen av nye presentasjoner skjer gjennom enkle HTML-skjema, og hele prosessen er utført på sekunder sammenlignet med timer i den forrige DPG-en. Verktøyet gjør det også enkelt å modifisere presentasjonsinnstillinger, overkjøre elementer fra presentasjonsmønsterspesifikasjonen, samt slette gamle presentasjoner fra systemet. Alt dette er nærmere beskrevet i kapittel 9 (Presentation Manager), og dette hovedmålet er etter min oppfatning også oppnådd.

11.1.5 Vurdering av hva som er oppnådd og forslag til fremtidige utvidelser (Hovedmål 5)

Det har vært svært viktig å hele tiden vurdere og begrunne de valg som er tatt gjennom utviklingen av det nye systemet. Derfor vil man mot slutten av hvert kapittel i denne oppgaven finne en oppsummering av hva som er oppnådd. Dette er med på å tydeliggjøre hvilke forbedringer som er gjort i forhold til DPG 1.0, i tillegg til å avdekke ny funksjonalitet som nå er på plass.

Det var også viktig å se på hvilke nye muligheter som DPG 2.0 legger opp til. Kapittel 10 (Forslag til videre arbeid) gir en kort oppsummering av tanker og ideer rundt fremtidige utvidelser av systemet. Det er mitt håp at en eller flere av disse kan danne grunnlaget for fremtidige prosjektoppgaver eller masteroppgaver, som på den måten kan bidra til fortsatt arbeid rundt JAFU-prosjektet og videreutvikling av DPG 2.0.

På bakgrunn av de oppsummeringene som man finner mot slutten av hvert kapittel, samt forslag til videre arbeid som er presentert i kapittel 10, er også det femte og siste hovedmålet oppnådd.

11.1.6 Oppsummering av delmål

Et av delmålene var bedre støtte for gjenbruk av elementer i spesifikasjonen, eksempelvis informasjon som går igjen på tvers av nettsidene i en presentasjon. Dette delmålet er oppnådd blant annet som følge av bruken av `page templates` og `master templates` i presentasjonsmønsterspesifikasjonen, og er nærmere beskrevet i kapittel 6 (Presentasjonsmønstre i DPG 2.0).

Delmålet som går på høy standard på kildekoden som produseres, blant annet gjennom å benytte Spring-rammeverket, er også oppnådd. Spring-rammeverket er brukt i alle de tre delsystemene, og har vært en avgjørende faktor for å skape en systemarkitektur bestående av løst koblede komponenter som er lettere å teste og bytte ut.

Det siste delmålet var å se på muligheten for et verktøy for en tredje brukertype, *pattern designer* (mønsterdesigner). Dette verktøyet vil gjøre det lettere å utvikle nye presentasjonsmønstre, og forslaget til hvordan et slikt verktøy kan utvikles er beskrevet i kapittel 10 (Forslag til videre arbeid).

11.2 Vurdering av teknologier

Her følger en kort vurdering av de mest sentrale teknologiene som er benyttet i DPG 2.0.

11.2.1 Vurdering av Spring-rammeverket

The Spring Framework er et applikasjonsrammeverk for Java-plattformen basert på åpen kildekode, utviklet første gang av Rod Johnson og presentert i hans banebrytende bok (Johnson 2002). Siden den gang har interessen rundt dette rammeverket økt dramatisk, og ses av mange som et komplement eller til og med en fullverdig erstatning til Enterprise JavaBean (EJB)-arkitekturen. Til DPG 2.0 ble versjon 2.0.5 av rammeverket benyttet, da dette var den nyeste tilgjengelige versjonen da prosjektet startet opp.

De forskjellige delsystemene og deres komponenter er, mye takket være den positive innvirkningen Spring-rammeverket har på hele arkitekturen, veldig løst koblet med hverandre. Dette gjør det først og fremst lettere å bytte ut kode med annen implementasjon, noe som vil gjøre det lettere for kommende utviklere å vedlikeholde og videreutvikle systemet i fremtiden. Alle som har arbeidet med DPG 2.0, inkludert undertegnede, er svært fornøyd med dette rammeverket, og kommer nok til å benytte det i fremtidige prosjekter i mange år fremover.

11.2.2 Vurdering av utstrakt bruk av XSLT

En viktig forskjell fra DPG 1.0 og DPG 2.0 er at XSLT har fått en enda viktigere plass i rendringen av presentasjonene enn tidligere. Ikke bare benyttes XSLT til å transformere XML til XHTML, men man drar også nytte av de sorterings- og filtreringsmulighetene som teknologien tilbyr. Dette var noe som i det gamle systemet, samt i forslaget til (Rossini og Liberati 2005) ble definert i selve spesifikasjonsfilen for presentasjonsmønsteret. Ved heller å dra nytte av eksisterende funksjonalitet i XSLT, unngår man å måtte implementere funksjonalitet for sortering i selve presentasjonsmotoren, noe som ville vært tidkrevende og ytterligere komplisere rendringsprosessen. Bruk av XSLT til både sortering og filtrering, i tillegg til transformering av XML til HTML, har derfor visst seg å være svært fornuftig.

11.2.3 Vurdering av Apache Velocity Templates

Bruken av Apache Velocity Templates (Velocity Apache Project) er helt nytt sammenlignet med DPG 1.0. Denne teknologien har vist seg å ha stor nytteverdi i forbindelse med å definere felles layout på alle nettsidene (`master template`), samt den interne layout på den enkelte nettside (`page template`). Selve syntaksen er både lett å lese og lære seg, og er dermed ikke noe stort hinder for dem som skal vedlikeholde og videreutvikle presentasjonsmønstre. Resultatene av bruken av Velocity Templates har med andre ord vært svært gode.

11.3 Vurdering av utviklingsmetoder og verktøy

11.3.1 Vurdering av samarbeid mellom prosjektdeltakerne

DPG 2.0-prosjektet har hatt totalt fire deltakere involvert de siste to årene, inkludert undertegnede. Min oppfatning er at dette samarbeidet har gått meget bra. Evt. uenigheter omkring arkitekturen eller teknologier har blitt tatt opp i plenum og diskutert grundig før man har kommet frem til en enighet. Stort sett har man arbeidet på samme sted i et åpent kontorlandskap der det er lett å starte diskusjoner, og kalle inn til møter på kort varsel. Det var også en stor fordel at alle prosjektdeltakerne kom fra samme kull og studieretning, slik at vi kjente hverandre meget godt fra før av.

11.3.2 Vurdering av utviklingsmetodikker

Vi har så langt det var praktisk mulig forsøkt å holde oss til en mer agil utviklingsmetodikk, der man praktiserer en mer ustrakt brukt av test-drevet utvikling og har et mindre fokus på detaljert planlegging og produksjon av dokumenter i forkant av utviklingen. Årsaken til dette er de mange fordeler som postuleres ved bruk av disse agile metodene, og selv det var krevende å hele tiden holde seg til for eksempel test-drevet utvikling, fungerte dette stort sett bra. Dette har altså vært en positiv erfaring, og undertegnede ønsker selv å kunne arbeide i team som bruker agile utviklingsmetodikker i en fremtidig jobbsituasjon.

11.3.3 Vurdering av utviklingsverktøy

Til prosjektet ble det brukt en hel rekke forskjellige verktøy. Det var viktig for alle at de verktøyene som ble brukt ble regnet som industristandard, slik at de erfaringene som

gjøres kan lettere være til nytte i en jobbsituasjon senere. Vi vil her oppsummere erfaringene med de viktigste verktøyene som ble benyttet.

Subversion
Subversion (Subversion, Tigris) ble benyttet som versjonskontrollverktøy, og gjorde det enkelt for prosjektdeltakerne å holde orden på all kildekode og dokumentasjon som ble produsert under utviklingen av prosjektet. Man lærer seg fort de viktigste kommandoene, og alle IDE-er har også plugins som gjør det enkelt å sjekke inn og ut kode i prosjektet.

Ant
For å automatisere bygging av prosjektet ble byggeverkøyet Ant (Ant, Apache Project) benyttet. Dette verktøyet hadde samtlige prosjektmedlemmer brukt fra før av, noe som gjorde at det var naturlig å velge Ant for DPG 2.0. Dette fungere fint, uten store problemer.

JUnit
Under hele utviklingen var det et stort fokus på at de komponentene som ble utviklet skulle bli tilstrekkelig testet. Sentralt i dette stod testrammeverket JUnit (JUnit, Sourceforge), det mest brukte testrammeverket for Java-plattformen. Igjen er dette et verktøy som var godt kjent blant prosjektdeltakerne og støtte for å kjøre testene er innebygget i alle moderne Java-IDE-er, slik at bruken av dette i prosjektet var uproblematisk.

JMock
For å teste komponenter som har mange avhengigheter til andre komponenter, ble mock-rammeverket JMock (jMock - Lightweight Mock Object Library) brukt mye. Dette rammeverket gjorde det lett å lage realistiske kopier av komponenter som ikke nødvendigvis eksisterte på tidspunktet testen ble skrevet, og lot oss programmere dem til å oppføre seg på bestemte måter. Dette rammeverket viste seg å være uvurderlig i enkelte situasjoner, var lett å lære seg og bruke, og sparte oss for mye tid og krefter.

Cobertura
For å sørge for at systemet var tilstrekkelig testet, og at kritisk programkode ikke forble utestet, benyttet vi Cobertura (Cobertura Coverage Tool) for å måle testdekningen. Det gjorde at vi fikk et godt overblikk over hvilke deler av systemet som det var skrevet tester for, og hvor vi evt. da måtte gjøre en bedre innsats med tanke på skriving av flere tester. Dette verktøyet fungerte stort sett greit, selv om det etter hvert ble klart at målingene ikke var spesielt nøyaktige. I et større prosjekt skulle vi nok ønsket oss et mye mer presist måleverktøy for testdekning, gjerne med flere funksjoner.

Eclipse, NetBeans og IntelliJ
Når det gjaldt valget av IDE-er, satte vi ikke noe bestemt krav til hva den enkelte prosjektdeltaker skulle benytte. Man stod fritt til å velge det IDE-et man selv var mest komfortabel med, noe som førte til at en rekke IDE-er ble benyttet til prosjektet, og at enkelte byttet underveis. Undertegnede har selv benyttet både Eclipse (Eclipse Project), NetBeans (NetBeans u.d.) og IntelliJ (JetBrains IDEA IntelliJ) på et eller annet tidspunkt

under utviklingen av DPG 2.0. Inntrykket jeg fikk var at alle disse tre IDE-ene var mer enn kapable til å bli benyttet i prosjektet, og hadde mer til felles enn de hadde forskjeller. Alle hadde god integrasjon med de øvrige verktøyene (Subversion, Ant, JUnit) og, etter at man har lært seg de mest brukte hurtigtastene, var de lett å bli komfortable med. Det er også en fordel å ha vært borti flere forskjellige IDE-er før man skal ut i full jobb.

11.4 Konklusjon

11.4.1 Personlige erfaringer

Arbeidet som er gjort i forbindelse med denne oppgaven har gitt meg mange nye kunnskaper. Ikke bare har jeg blitt godt kjent med en rekke av dagens viktigste teknologier, som Spring Framework, XSLT, XML, Velocity og selvsagt XHTML, CSS og Javascript. Jeg har også lært mye om hvordan det er å arbeide tett sammen med øvrige prosjektmedlemmer. Jeg har blitt flinkere til å formidle tanker og ideer på en slik måte at andre kan forså hva jeg ønsker å oppnå, og på den måten sette i gang en konstruktiv dialog om hvordan dette evt. kan implementeres i systemet. Dette er alle erfaringer som garantert vil komme til nytte i min fremtidige karriere som systemutvikler.

11.4.2 Mitt bidrag til JAFU-prosjektet

Det er ingen tvil om at DPG 2.0 har et stort potensiale og er et omfattende bidrag til hele JAFU-prosjektet. Forhåpentligvis kan en eller flere av de ideer som er presentert i denne oppgaven danne grunnlag for flere spennende prosjekter knyttet til DPG 2.0 og konseptet rundt presentasjonsmønstre, og på den måte bidra til videreutvikling og økt interesse for hele JAFU prosjektet.

Bibliografi

Acegi Security. *http://www.acegisecurity.org/*.

AJAX, Wikipedia. *http://en.wikipedia.org/wiki/AJAX*.

Apache Ant. *http://ant.apache.org/*.

Apache Software Foundation. *Apache Tomcat.* 1999-2006.
http://tomcat.apache.org/tomcat-6.0-doc/realm-howto.html (funnet March 24, 2008).

Berg, Christian Magnus. *Statusrapport for presentasjonsmønstermotor (PMM).*
Department of Informatics, University of Bergen, 2004.

Berg, Karianne. *Elektronisk Kompendium,*
Institutt for Informatikk, Senter for Etter- og Videreutdanning (SEVU, UiB), 2006.

Berg, Karianne. *Persisetensproblematikk i Dynamic Presentation Generator*.
Institutt for Informatikk, Universitetet i Bergen, 2008.

Boiko, Bob. *Content Management Bible.* Wiley Publishing, 2005.

Cascading Style Sheet, W3C. *http://www.w3.org/Style/CSS/*.

Cobertura Coverage Tool. *http://cobertura.sourceforge.net/*

Cruickshanks, Kevin. *Verktøy for generering av XML-baserte presentasjonar: JPGen - Java presentasjons generator.* Department of informatics, University of Bergen, 2004.

DAO, Core J2EE Patterns, SUN.
http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html.

db4objects. *db4o :: Native Java and .NET Object Database :: Open Source.*
www.db4o.com (funnet 04 26, 2008).

DOJO Toolkit. *http://dojotoolkit.org/*.

EasyMock. *http://www.easymock.org*.

Eclipse Project. *http://www.eclipse.org*.

EMMA Java Code Coverage. *http://emma.sourceforge.net/*.

Enonic. *Progressive web content management for Java - Enonic.*
www.enonic.com (funnet 04 16, 2008).

Enterprise JavaBeans, SUN. *http://java.sun.com/products/ejb*.

Espelid, Yngve. *Dynamic Presentation Generator*. Department of Informatics, University of Bergen, 2004.

Fowler, Martin. Domain Spesific Language, *http://www.martinfowler.com/bliki/DomainSpecificLanguage.html*

Fowler, Martin. Dependency Injection. *http://martinfowler.com/articles/injection.html*.

Fowler, Martin. Mocks aren´t Stubs. *http://martinfowler.com/articles/mocksArentStubs.html*.

Fowler, Martin. *Patterns of Enterprise Application Architecture*. Pearson Education, Inc, 2003.

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

GNU Project. *The GNU General Public License GNU Project - Free Software Foundation (FSF)*. 12 2 2008. http://www.gnu.org/licenses/gpl.html.

Gould, I. H. *IFIP Guide to Concepts and Terms in Data Processing*. North-Holland publishing, 1971.

Graph Connectivity, Wikipedia. *http://en.wikipedia.org/wiki/Connectivity_(graph_theory)*.

HTML Encoding, Wikipedia. *http://en.wikipedia.org/wiki/Character_encodings_in_HTML*.

Huseby, Sverre H. *Innocent Code - A security wake-up call for web programmers*. Wiley, 2004.

Ingvaldsen, Bjørn Ove. «Multimedia i Dynamisk Presentasjons Generator 2.0.» Institutt for Informatikk, Universitetet i Bergen, 2008.

Java Server Pages, SUN. *http://java.sun.com/products/jsp*.

jMock - Lightweight Mock Object Library. *http://www.jmock.org*.

Johnson, Rod.
Expert One-on-One J2EE Design and Development.
Wrox Press, 2002.

jQuery JavaScript Library.
http://jquery.com/.

jQuery JavaScript Library - DatePicker Plugin.
http://docs.jquery.com/UI/Datepicker/datepicker.

JUnit, Sourceforge.
http://junit.sourceforge.net/.

Koskela, Lasse.
Test Driven - Practical TDD and Acceptance TDD for Java Developer.
Manning, 2008.

Løvik, Kristian.
"Webucator 3.0", Institutt for Informatikk, Universitetet i Bergen, 2008.

Løvik, Kristian, og Karianne Berg. *"Sikkerhetsproblemer i DPG 1.0."*
Institutt for Informatikk, Universitetet i Bergen, 2006.

Lervik Larsen, Øystein. *Utvikling av nye presentasjonsmønstre.* 2006.

Mughal, Khalid Azim. *Presentation Patterns: Composing Web-based presentations.*
Bergen: Department of Informatics, University of Bergen, 2003.

NetBeans IDE. *http://www.netbeans.org/*.

Nordbotten, Joan.
Multimedia Information Retrieval Systems. (not published yet), 2008.

Oxygen XML Editor.
http://www.oxygenxml.com/. SyncRO.

Rossini, Alessandro, og Graziano Liberati.
INF219 DPG Work Report,
Institutt for Informatikk, Universitetet i Bergen, 2005.

Schumacher, Markus, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Bushcmann, og Peter Sommerlad.
Security Patterns - Integrating Security and System Engineering.
Wiley, 2006.

Senselogic AB. *Senselogic.*

www.senselogic.se (funnet 04 16, 2008).

Spring MVC, SpringSource.
http://static.springframework.org/spring/docs/2.0.x/reference/mvc.html.

SpringSource, IoC Module.
http://static.springframework.org/spring/docs/2.0.x/reference/beans.html.

Subversion, Tigris.
http://subversion.tigris.org/.

Sun Microsystems. ***The Java Community Process (SM) program***
JSRs: Java Specification Requests - detail JSR-170.
http://www.jcp.org/en/jsr/detail?id=170 (funnet 04 16, 2008).

The Spring Framework. SpringSource
http://www.springframework.org/.

TinyMCE - JavaScript WYSIWYG Editor.
MoxieCode. http://tinymce.moxiecode.com/.

Velocity Apache Project.
Apache Project. http://velocity.apache.org/.

Web Service, Wikipedia. *http://en.wikipedia.org/wiki/Web_services*.

WYSIWYG Wikipedia. *http://en.wikipedia.org/wiki/WYSIWYG*.

XML Schema, W3C. *http://www.w3.org/XML/Schema*.

XPath, W3C. *http://www.w3.org/TR/xpath*.

Zope.org. *Zope*
www.zope.org, (funnet 04 16, 2008).