MULTIMEDIA I DYNAMISK PRESENTASJONS GENERATOR 2.0

Masteroppgave

Bjørn Ove Ingvaldsen
Institutt for informatikk
Universitetet i Bergen
9. September, 2008

Forord

Denne oppgaven er resultatet av mitt masterstudium i programutvikling. Oppgaven er gjennomført på Institutt for informatikk ved matematisk-naturvitenskapelig fakultet tilhørende Universitetet i Bergen.

Oppgaven omhandler en ny implementasjon av innholdshåndteringssystemet Dynamic Presetation Generator DPG, hvor hovedvekten av oppgaven tar for seg integrering av multimediafunksjonalitet i systemet.

Jeg vil gjerne takke veilederne mine, Khalid A. Mughal og Torill Hamre, som har bidratt med verdifulle innspill og ideer under utviklingen samt gode råd og kommentarer i forbindelse med oppgaveskrivingen. Jeg vil også takke mine medstudenter ved JAFU- prosjektet Bjørn Christian Sebak, Kristian Skønberg og Karianne Berg som har vært gode samarbeidspartnere gjennom hele masterstudiet.

Til slutt vil jeg takke min famile som har vært veldig tolmodig og forståelsesfull under hele prosessen med masteroppgaven.

Bergen, September 2008

Bjørn Ove Ingvaldsen

Innholdsfortegnelse

Liste over figurer

Liste over tabeller

Liste over eksempler

Liste over plugins

1 Introduksjon

1.1 Innledning

Denne masteroppgaven er en del av et prosjekt, med navn JAva FjernUndervisning (JAFU), på Institutt for Informatikk ved Universitetet i Bergen. Formålet med dette prosjektet har vært å lage systemer som gjør det mulig for universitetet å tilby e-læring til sine studenter. Dette har resultert i ulike typer programvare som for eksempel oppgaveinnleveringssystem, et interaktivt kontrollspørsmålsystem og ulike systemer for å håndtere og publisere kurssider på internett.

Java Presentasjons Generator (JPG) [1] var et online system som håndterte kurssider. Dette systemet brukte det vanlige filsystemet til datalagring, noe som raskt ble et problem og som dannet grunnlaget for videreutvikling.

Samtidig som man så på datalagringsproblematikken i JPG'en startet et nytt delprosjekt opp. Formålet med dette prosjektet var å utvikle et mer generisk system for web-baserte systemer. Resultatet ble en prototype med navn PresentasjonsMønster Motor (PMM) [2] som introduserte endel nye ideer i forbindelse med web-baserte presentasjoner. Men prototypen hadde også mange likhetstrekk med JPG'en. Det var derfor ingen grunn til å utvikle og vedlikeholde to slike system. Resultatet ble at man utviklet et nytt Content Management System (CMS) [3] som lagret strukturerte data og hadde et rammeverk for å publisere dataene.

Systemet fikk navnet Dynamisk Presentasjons Generator (DPG) [4]. Det ble introdusert for e-lærings studentene i 2005 og er fremdeles i drift i dag. DPG-en tar i bruk hovedideen til PMM'en der en benytter seg av presentasjonsmønstre [5]. Man skiller konsekvent mellom struktur og innhold, med det mål å kunne gjenbruke strukturen for annen type innhold.

DPG 1.0 har også endel mangler, først og fremst når det gjelder arkitektur og funksjonalitet. Dermed var det satt føringer for et nytt prosjekt, som har som mål å utvikle et nytt CMS basert på dagens teknologi og behov. Fokuset til denne oppgaven er å delta i utviklingen av en ny dynamisk presentasjonsmotor, hvor hovedfokuset vil være å utvikle bred støtte for multimedia.

1.2 Problembeskrivelse

Tenk på et senario der en webprogrammerer akkurat har gjort seg ferdig med en ny og moderne webapplikasjon for en fotballklubb. Den blir satt i drift, og etter en stund blir utvikleren kontaktet av en representant for en annen fotballklubb som uttrykker stor begeistring og gjerne vil ha samme systemet i sin klubb. I verste fall må utvikleren begynne helt på nytt. Problemet er ofte at formateringen er for tett koblet opp mot innholdet. Å skille de to igjen i ettertid, for å lage en lik applikasjon med nytt innhold, er ikke alltid like trivielt. Klippe og lime mellom eksisterende og nye sider kan fort by på problemer.

Hovedpoenget med å lage en presentasjon er å formidle informasjon på en måte som gjør det enkelt for mottakeren å fatte innholdet. Ved å strukturere innholdet på en måte som gjenspeiler den logiske og semantiske betydningen, oppnår man en god presentasjon. Et eksempel kan være en nyhetssak på en online nettavis. En slik artikkel har ofte overskrift, ingress, brødtekst, bilde og video. Dette kan sees på som et mønster for mange online aviser.

Hensikten til denne oppgaven er å videreutvikle DPG 1.0, som er et generisk Content Management System CMS basert på konseptet presentasjonsmønster. Hovedproblemet i denne oppgaven vil være å utvikle funksjonalitet eller støtte for multimedia. Dette var helt fraværende i DPG 1.0.

1.3 Mål for oppgaven

Hovedmålet til oppgaven er å utvikle støtte for multimedia i presentasjonsmotoren til den nye DPG2.0. For å nå dette målet, er det andre delmål som må oppfylles:

1. Det må utføres en grundig analyse av DPG 1.0 for å belyse løsninger som bør forbedres og løsninger som er gode og som bør inngå i det nye systemet.
2. Det må fastsettes hvilke krav man skal ha til multimediafunksjonalitet.
3. Det må utvikles en ny presentasjonsmønsterspesifikasjon som utfra analysen forbedrer den gamle spesifikasjonen og i tillegg gjør nødvendige utvidelser med hensyn til multimedia.
4. Det må utføres en analyse av hvordan multimediafunksjonalitet skal implementeres i det nye systemet.
5. Det må implementeres en ny motor som rendrer presentasjoner, hvor multimediafunksjonaliteten implementeres i henhold til analysen i punktet ovenfor.
6. Funksjonalitet som gjenspeiler kravene til multimedia må implementeres.
7. Det må utføres en analyse av nytteverdien til multimedia i DPG2.0.

Utviklingen av DPG 2.0 er et prosjektarbeid med fire deltagere. Det er Bjørn Christian Sebak, Karianne Berg, Kristian Skønberg Løvik og undertegnede. Bjørn Christian Sebak har ansvaret for den overordnede arkitekturen til DPG 2.0 [6], Karianne Berg har i sin oppgave ansvar for persistensproblematikken i DPG2.0 [7] og Kristian Skønberg Løvik har ansvaret for brukerhåndtering og aksesskontroll i DPG 2.0 [8].

Delmål 1 utføres ved et samarbeid mellom Bjørn Christian Sebak, Karianne Berg og undertegnede. Delmål 3 og 5 utføres ved et sammarbeid mellom Bjørn Christian Sebak og undertegnede. De resterende delmålene utføres ene og alene av undertegnede.

1.4 Organisering av dokumentet

Kapittel 2: Bakgrunn

I dette kapittelet presenteres først konseptet om innholdshåndteringssystem og hvordan de er bygget opp. Videre følger en gjennomgang av konseptet presentasjonsmønstre som er grunnsteinen i JAFU-prosjektet. Til slutt presenteres den gamle DPG 1.0 versjonen som skal erstattes med DPG 2.0.

Kapittel 3: Analyse av DPG 1.0

Utviklingen av Dynamic Presentation Generator versjon to, bygger på en grundig analyse av den første versjonen. Her følger en presentasjon av analysen. Spesielt blir det lagt vekt på den nåværende mønsterspesifikasjonen og presentasjonsmotoren for rendring av informasjon.

Kapittel 4: Det nye systemet

På bakgrunn av analysen av det gamle systemet presenteres her en modell av det nye systemet. Overordnede krav til multimedia blir belyst, samt en vurdering av mulige løsninger på multimediaproblematikken. Til slutt presenteres den overordnede arkitekturen til systemet.

Kapittel 5: Presentasjonsmønstre i DPG 2.0

Ønske om ny funksjonalitet, samt ønske om en litt annen oppbygging av systemet, har ført til en ny mønsterspesifikasjon. Kapittelet presenterer og analyserer først mønsterspesifikasjonen som er utarbeidet av Alessandro Rossini og Graziano Liberati [5]. Denne spesifikasjonen var hovedinspirasjonskilde til den nye spesifikasjonen. Kapittelet fortsetter med en redgjørelse av alle begrepene i den nye spesifikasjonen, samt sammenhengen mellom dem. En spesiell oppmerksomhet vil bli gitt til begrepene som er innført i sammenheng med multimedia.

Kapittel 6: Presentasjon av Presentation Viewer PV

Her følger en presentasjon av den nye presentasjonsmotoren, Presentation Viewer (PV). Først beskrives arkitekturen til delsystemet, deretter beskrives rendringsprosessen Det vil bli lagt vekt på forskjellene i rendring av innhold med og uten multimedia.

Kapittel 7: Arkitektur til pluginmodul

Dette kapittelet gjenomgår pluginmodulen som ble foretrukket i forbindelse med implementasjon av multimediastøtte i DPG 2.0. Presentasjon av mønster for å bygge en pluginsmodul, samt arkitekturen til modulen blir, gjennomgått i detalj.

Kapittel 8: Plugins i DPG 2.0

Kapittelet starter med å beskrive hvordan man kan koble seg til applikasjonen med et plugin, hva man kan oppnå med et plugin, og hvilke to kategorier man har av plugins. Til slutt presenteres en tabell som inneholder referanser til hvor en kan finne mer detaljert informasjon om plugins-ene som er utviklet til applikasjonen.

Kapittel 9: Bruk av multimedia i ulike mønster

Multimedia er i dag et mye brukt hjelpemiddel for å frembringe et budskap på en nettside. I dette kapittelet presenteres nytteverdien av å tilby multimedia støtte i den nye DPG-en. Det blir benyttet ulike mønster for å fremheve egenskapene til multimediaobjektene.

Kapittel 10: Evaluering og forslag til videre arbeid

I siste kapittel følger en konklusjon og en evaluering av eget arbeid. Det presenteres også ideer til videre utvikling av DPG'en.

2 Bakgrunn

Dette kapittelet er skrevet i fellesskap av følgende personer: Karianne Berg, Bjørn Christian Sebak og Bjørn Ove Ingvaldsen, og det inngår dermed i alle oppgavene.

2.1 Innholdshåndteringssystemer

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:

Joan Nordbotten [9] 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.

I H Gould [10] 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 bekriverTCP/IP-protokollen, vil gi forskjellig grad av informasjon til en nettverksprogrammerer og en bussjåfør. Dette innebærer at datamaskiner ikke – i hvertfall ikke i den formen de har i dag - er i stand til å lagre informasjon direkte. Dette synet støttes også av Bob Boiko [3].

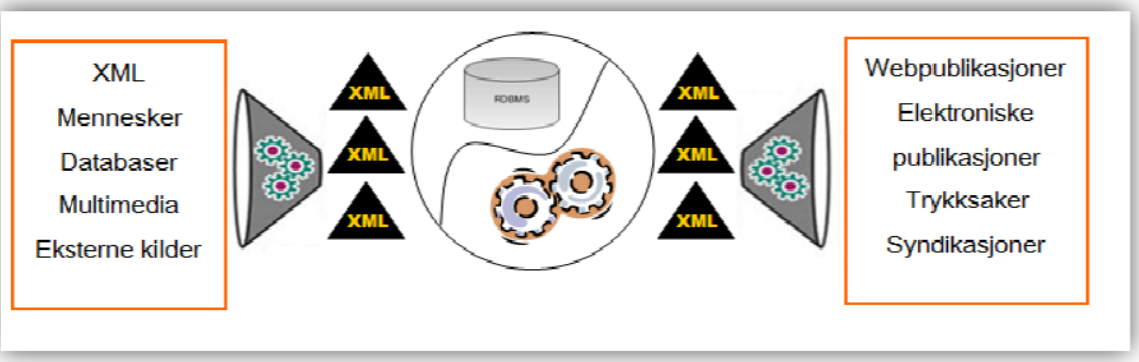
Innhold er en term som ikke er mye definert i litteraturen. Boiko i [3] har denne definisjonen av innhold: "Content […] is information that you tag with data so that a computer can organize and systematize its collection, management and publishing". Med dette mener han at siden datamaskiner ikke er i 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 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. (seFigur 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 ofte websider, men det bør også være mulig å produsere andre medier, for eksempel trykksaker.



Figur 2-1: Skjematisk oversikt over komponentene i et innholdshåndteringssystem. (Inspirert av [3])

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 [11]. Programvaren tilbyr et komplett system for innholdshåndtering, og er basert på åpne industristandarder. Løsningen benyttes av flere store norske selskaper, blant andre 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) [12] 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 [13]. Selskapet har Rikard Öberg i spissen, en kjent skikkelse i OpenSource-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 [14] for interne data, og overlater til utviklerne av portlets å velge persisensteknologi selv. Applikasjonen benytter seg av Velocity som språk for maler. Dette synes vi fungerer på en god måte, og er et langt bedre alternativ enn å ha malene i JSPX, slik som DPG 1.0 har.

Plone

Plone er et OpenSource innholdshåndteringssystem som er lisensiert under lisensen GNU General Public Licence (GNU) [15]. Det er basert på applikasjonsserveren Zope [16] og skrevet i programmeringsspråket Python.

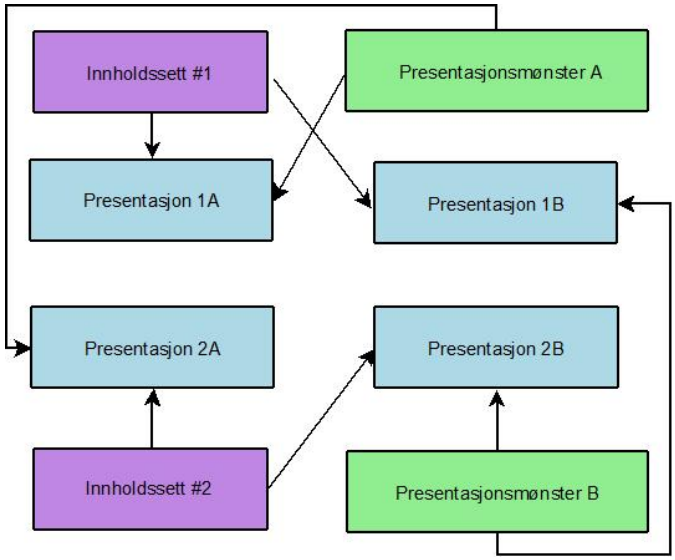
Plone bruker sin egen teknologi for å inkludere funksjonalitet, kalt add-ons. Plone kommer med Zopes interne, objektbaserte database (ZODB) som standard persistensteknologi, men kan også settes opp med de fleste relasjonelle databaser.

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, propritært 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

Konsept

DPG-en er basert på konseptet om *presentasjonsmønstre*, en ide som ble fremlagt av Khalid Mughal i 2003 [17]. 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 [18]. 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.

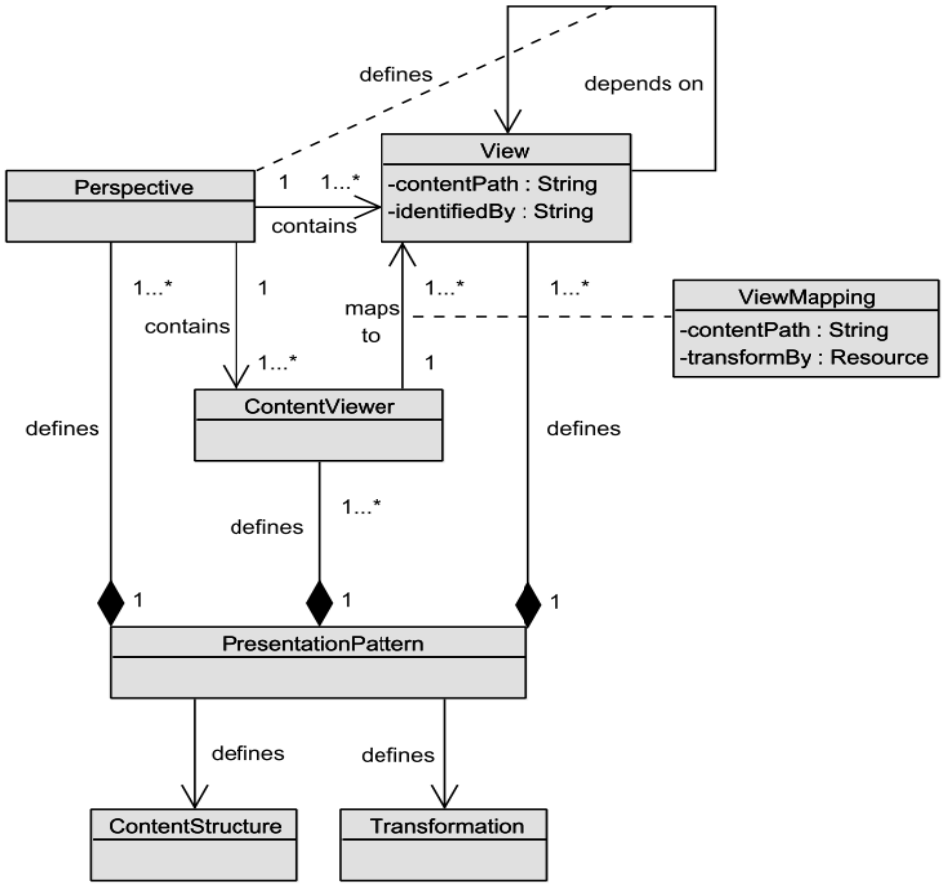
2.3 Presentasjonsmønstre i DPG 1.0

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.

Presentasjonsmønsterspesifikasjonen

Presentasjonsmønsterspesifikasjonen består av tre hovedkomponenter:

- View
- Content-viewer
- Perspective



Figur 2-3: Sammenhengen mellom de forskjellige komponentene som inngår i et presentasjonsmønster (fra 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 mønster-spesifikasjonen. 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: Et view som viser til en liste over studenter

Content-Viewer

Denne komponenten 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 content-path-attributten, 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: Hvordan en content-VIEWER kan defineres

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 ett 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 [4].

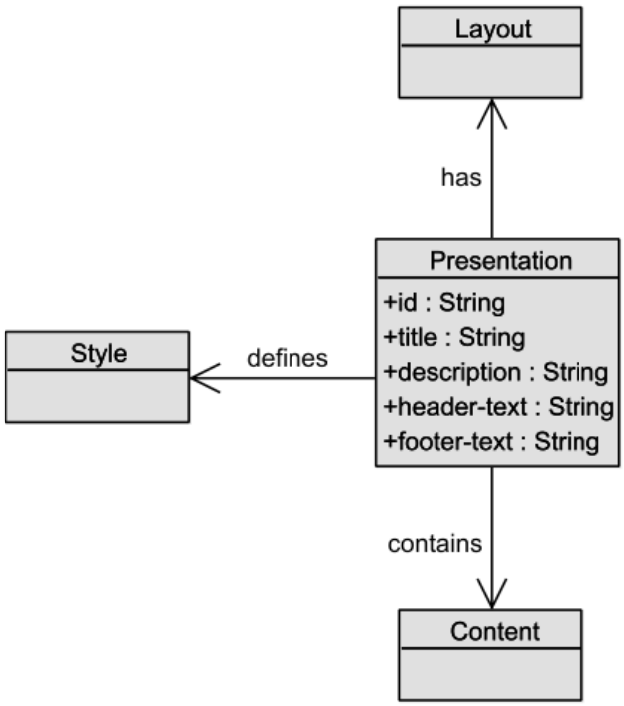
```
<perspective name="studentList">
<view-name>infoView</view-name>
<view-name>contactView</view-name>
<content-viewer-name>
            listOfAllStudentsViewer
<content-viewer-name>
</perspective>
```

Eksempel 2-3: Spesifikasjon av et perspective

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: Definisjon av hvilket innhold (content) som skal inngå i en presentasjon

Alle filer som inngår i presentasjonen må listes under `<content>`-taggen. 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 JSP-side. Eksempel 2-5 viser et eksempel på hvordan dette gjøres:

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

Eksempel 2-5: Spesifikasjon 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. Eksempel 2-6 viser et eksempel på hvordan man setter stilark:

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

Eksempel 2-6: Definisjon av stylesheet

2.4 DPG 1.0

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 verktøy til JAFU-prosjektet:

Navn på system	Beskrivelse	Sist gjort arbeid på
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)

DPG 1.0 ble skrevet av Yngve Espelid i 2004 som en del av masteroppgaven hans, ferdigstilt i 2004. Dette arbeidet er beskrevet i [4]. Espelid baserte noe av arbeidet sitt på resultatene til flere andre hovedfagsoppgaver og doktorgradsavhandlinger skrevet ved instituttet, spesielt [1] og [2], som omhandlet lignende prosjekter, kalt henholdsvis JPGen og PMM. Mer informasjon om disse prosjektene finnes i kapittel 2 av [4], 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.

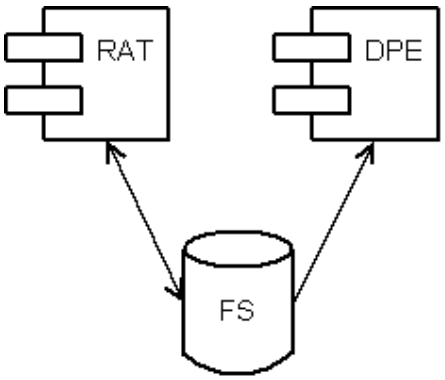
DPG 1.0 har gjennomgått endel vedlikeholdsarbeid etter Espelids oppgave. Sommeren 2005 foregikk det arbeid med utarbeidelse av systemdokumentasjon, retting av noen bugs, refaktorering av deler av koden, og det ble påbegynt reimplementasjon av presentasjonslaget i

MVC-rammeverket Spring MVC. 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 for 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 til presentasjonsmønstre og generiske systemer, og både sterke og svake sider med den nåværende DPG-implementasjonen ble avdekket.

Overordnet arkitektur

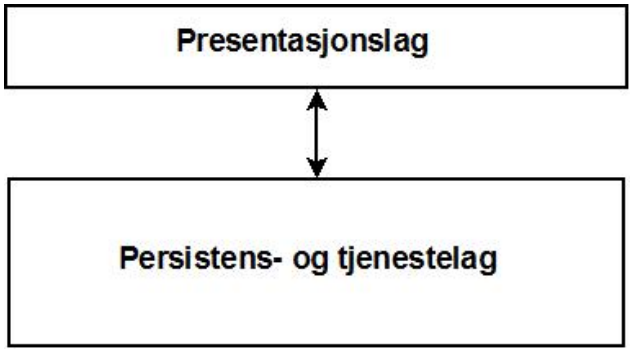
DPG 1.0 er delt inn i to hoveddeler(se Figur 2-5): Dynamic Presentation Engine (DPE) og Repository Administration Tool (RAT). Begge disse bruker et sentralt repository som datakilde. Dette repositoriet er en katalog på filsystemet med en forhåndsdefinert katalogstruktur. På grunn av problemer med kompilering av JSP-er [19] og Tomcat [20] og nedlasting av filer for brukere, må denne katalogen ligge inne i applikasjonens katalog i ”webapps”-katalogen til Tomcat.



Figur 2-5: Overordnet arkitektur i 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 parseXML-data for å bygge objekter og tilstandshåndtering.



Figur 2-6: Den overordnede laginndelingen i DPG 1.0

Presentasjonslaget til DPG-en bruker designmønsteret Front Controller [21], 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, som en del av en refaktorering, foretatt etter at Espelid var ferdig med oppgaven.

I persistens- og tjenestelaget til DPG 1.0 er designmønsteret Singleton [22] 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 skal skje kun en gang. Dette er i bunn og grunn en primitiv erstatning for caching 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 [23]. Ingen form for transaksjonshåndtering er implementert (for eksempel i form av låsing).

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. Administratorer 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 en Administrator har alle rettigheter som Publishers har.

Autentiseringen foregår via Tomcat, satt opp med et JDBC Realm. [24] Denne er backet av en PostgreSQL-database [25] 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

I kapittel 2 ble det foretatt en gjennomgang av det gamle systemet. Dette kapittelet vil ta for seg svakhetene ved det gamle systemet, hva som var bra med det gamle systemet som vi ønsker å ta med oss videre, hvilke muligheter man har for multimedia og til slutt en oppsummering av hvorfor vi ønsker å lage en helt ny applikasjon.

3.1 Svakheter i DPG1.0

Her følger en kort en presentasjon av svakhetene i DPG 1.0. For en mer inngående gjennomgang se kapittel 4 i [7].

Lite fleksibel arkitektur

Applikasjonen er kun delt opp i to lag: (1) Presentasjonlag og (2) Persistens- og tjenestelag. Kun to lag fører til en lite fleksibel og lite testbar applikasjon. Det blir mange avhengigheter mellom moduler med forskjellig ansvarsområde, noe som fører til at det blir vanskelig å reimplementere en modul. Et eksempel er at siden det er mange avhengigheter mellom klassene som utfører tjenesteoperasjoner og klassene som utfører persistensoperasjoner, vil det bli veldig vanskelig å bytte ut persistensstrategien.

En annen svakhet er at de to lagene som som applikasjonen inneholder, ikke er implementert på en god måte. Grensesnittene blir ikke brukt i instansieringen av objekter, noe som igjen fører til høy grad av kobling mellom lagene.

Det er også funnet referanser til objekter i persistens- og tjenestelaget som går utover ansvarsområdet til dette laget. Dette skaper en sterk kobling mellom lagene med mange avhengigheter, noe som påvirker fleksibiliteten til systemet i en negativ retning. Et eksempel er en referanse til et `HTTPServletRequest` objekt. Ved at klasser i persistens- og tjenestelaget har slike referanser, gjør det for eksempel veldig vanskelig å bytte ut teknologien som benyttes i presentasjonslaget, siden referanser til teknologispesifikke objekter finnes i lagene under.

Uoversiktlig brukergrensesnitt

Brukergrensesnittet til sidene som genereres av DPE er ikke fast, men bestemmes av mønsteret og den tilhørende presentasjonen. Et ankepunkt med dette grensesnittet er at brukere med rollen publisher ikke har mulighet å endre på innhold fra disse sidene.

Men brukergrensesnittet til RAT er det mer å utsette på. Hovedproblemet er at man må kunne XML [23] for å oppdatere innholdsfilene. Det er en stor ulempe siden en bruker med rollen publisher skal kunne endre på innhold. En publisher kan f.eks. være en foreleser med ingen kunnskap om XML og da vil det naturligvis være vanskelig å oppdatere innhold på en korrekt måte.

Repository må ligge i webapps mappen til Tomcat

Det er et krav at repository skal ligge i tomcat sin webapps mappe. Det er to grunner til dette:

- Det blir ikke brukt strømbasert filnedlasting noe som medfører at filer som skal lastes ned må befinne seg i en katalog som er tilgjengelig over HTTP [26].
- DPG1.0 benytter JSPX-filer for å spesifisere layout på sidene. JSPX filer må kompileres av Tomcat før bruk og må derfor ligge i web-inf mappen til webapplikasjonen.

Ulempen med dette er at hver gang man skal laste inn ny versjon av applikasjonen må man kopiere repository mappen manuelt fra serveren før en sletter applikasjonen, dette for å ungå å miste oppdatert informasjon i feks innholdsfilene. Så må man kopiere repository mappen tilbake igjen etter at man har redeployet applikasjonen.

Lite fokus på sikkerhet

Den mest alvorlige feilen under dette punktet er nok at alle filer i en presentasjon er tilgjengelig for alle hvis man har en korrekt Uniform Resource Locator (URL) [27] til filen. I DPG 1.0 er det gjort et forsøk på å implementere mønsteret Single Access Point [28] ved hjelp av tomcat-autentisering, men det er ikke tatt hensyn til at det er mulig å hente filer ved hjelp av fullstendige URL'er.

Et annet alvorlig tilfelle er at rollen publisher i en presentasjon har tilgang til alle JSPX filene til den bestemte presentasjonen. En har da tilgang til et krafig programmeringsspråk i og med at man kan skrive ren Java-kode i en scriplet tag. Dette kan utnyttes til å få applikasjonen til å gjøre uønskede ting, for eksempel stenge ned hele tomcat serveren.

Mangelfull feilhåndtering

I kildekoden til DPG 1.0 er det påfallende mange steder at man bruker try- catch [29] blokker, der catch blokken svelger unntaket, uten å sende videre en feilmelding som kan vises til brukeren og uten å logge feilen. Brukeren sitter da igjen uten å skjønne noe om hva som har gått galt. Det er også en del steder der unntak ikke fanges i det hele tatt. Eksempler på dette er stadige `NullPointerExceptions` [29] som dukker opp.

Lite effektiv minnebruk

I oppstarten av applikasjonen lastes alle presentasjonsmønstre og presentasjoner inn i minnet. Finnes det da mange presentasjonsmønstre med flere presentasjoner per mønster, sier det seg selv at systemet krever mye minne for å kjøre. Den laster inn alt av ressurser, uavhengig om noen ressurser nesten aldri blir brukt.

Det virker også som det er en del minnelekkasje som kommer av at det opprettholdes referanser til objekter som ikke trengs mer. Dette fører til at applikasjonen må startes på nytt med jevne mellomrom.

Programfeil (bugs)

Brukere med rollen readers vil med jevne mellomrom oppleve å få en `NullpointerException` når et view lastes. Denne feilen er pr dags dato ikke identifisert, og man aner derfor ikke hvorfor den forekommer.

En annen programfeil som er kritisk, omhandler oppdatering av spesielle typer XML trær. Her blir filene som representerer trærne oppdatert, mens objektene som representerer trærne i applikasjonen ikke blir oppdatert. Dette fører til at man må restarte applikasjoen og laste alt inn i minnet på nytt for at man skal kunne se endringene. Det er særlig i XML trærne som omhandler mønsterredigering at denne feilen oppstår.

Manglende tester

Noe av grunnen til at det er vanskelig å refaktorere kildekoden, er at det ikke finnes automatiserte tester. Enhetstester er også vanskelige å skrive, siden det er så høy grad av kobling mellom klasser og lag i systemet. Det er heller ingen akseptansetester eller integrasjonstester.

Mangelfull dokumentasjon

Av dokumentasjon finnes masteroppgaven til Yngve Espelid [4] og Javadoc [30] til kildekoden. Masteroppgaven forklarer konseptene godt, men sier ikke så mye om implementasjonen av systemet, og valg som er gjort i den forbindelse. Det har vist seg vanskelig å sette seg godt nok inn i systemet til å videreutvikle det og rette opp feil med den dokumentasjonen som har vært tilgjengelig.

3.2 Gode løsninger i DPG 1.0

Her følger en oversikt over gode løsninger som det er verdt å bygge videre på i den nye versjonen.

Presentasjonsmønster

Ideen om å bruke presentasjonsmønstre, hvor man skiller konsekvent mellom struktur og innhold, er noe som har vært en suksess i DPG 1.0. Systemet har vært benyttet som kursmønster for fag på institutt for informatikk, og man har gode erfaringer med å lage nye presentasjoner utfra et definert kursmønster. Ved å skille strukturen fra innhold, vil for eksempel en kursansvarlig enkelt kunne lage en ny presentasjon eller web-side for sitt kurs utfra en bestemt mønsterdefinisjon og legge til helt nytt innhold.

Stor fleksibilitet

Presentasjonsmønstre gir stor fleksibilitet til systemet med tanke på at man kan utvikle nye mønstre til ulike domeneområder. Et eksempel her er at en kursansvarlig trenger en bestemt struktur for å få frem informasjon til sine studenter, mens en barnehageansvarlig har andre krav til struktur for å få frem budskapet til foreldrene til barna i barnehagen,se [31].

Men det er også stor fleksibilitet til hvordan innholdet skal presenteres i ulike presentasjoner. Hver presentasjon har egne layout-filer, hvor en bestemmer blant annet plasseringen til de ulike elementene som skal vises. I tillegg har hver presentasjon egne stilark, der en for eksempel kan definere bakgrunnsfarger, skrifttype, logoer med mer

Rollene i systemet

Bruken av ulike roller for å begrense hva de ulike brukerne av systemet skal ha lov til å utføre, er også en god løsning. Et problem i DPG 1.0 var at publishere i en presentasjon hadde mulighet til å endre på layouten til sidene. Siden layout var definert i JSPX-filer, fikk publishere for store rettigheter i forhold til hva de egentlig var ment til å ha.

XML til å definere struktur og innhold

Fordelen med å bruke XML til å definere struktur og innhold, er at man på en enkel måte kan sjekke at innholdet i en presentasjon samsvarer med det bestemte mønsteret som presentasjonen skal gjenspeile. I tillegg er XML en veletablert og veldokumentert teknologi som forventes å ha lang levetid. XML er også godt egnet til å konvertere til andre formater, noe som kalles en tranformasjon.

XSLT transformering

Som nevnt i forrige avsnitt, er XML enkelt å konvertere til andre format. I dpg 1.0 brukes XSLT [32] stilark til å konvertere XML til HTML [18]. Dette er en god løsning for da legges innhold fra XML treet inn i HTML tagg-er, som igjen kan sendes til nettleseren.

JSP for å definere layout

JSP benyttes for å ha mulighet til å legge inn dynamisk innhold i responsobjektet til en web forespørsel. Jsp filene bestemmer layouten på siden, og hver presentasjon har egne layoutfiler. Dette medfører at en har stor fleksibilitet i og med at en kan bestemme hvor de ulike elementene skal plasseres for hver enkelt presentasjon. Ulempen med å bruke JSP som layout filer for presentasjoner, er at de er nødt til å kopileres av Tomcat serveren, noe som tvinger repository mappen til å ligge i webapps mappen til tomcat serveren.

3.3 Manglende støtte for multimedia

I DPG 1.0 var det ingen implememententert støtte for multimedia. Det var riktignok mulig å vise multimediaobjekter, men da måtte brukeren som opprettet et mønster, selv skrive all nødvendig XSLT-kode i transformasjonsarket. I Eksempel 3-1 ser man kode for å sette inn et bilde. Her bygger man opp et HTML `img` element ved bruk av språket XSLT. Elementet `Frontpage` i innholdsfilen inneholder URL-en til bildet.

Det var med andre ord ingen støtte for å rendre nødvendig HTML kode for å fremvise multimediaobjekter i DPE-en, og det var heller ingen støtte for oppdatere en innholdfil med et nytt multimediaobjekt. Det vil si at en måtte laste opp filen i RAT-en og så manuelt oppdatere innholdsfilen for å legge inn et nytt multimediaobjekt.

```
<!--Innholdsfil -->
<books label="Litteratur">
  <book>
    ...
    <frontpage>repository/presentations/inf100fv2005/resources/cover.jpg</frontpage>
  </book>
</books>

<!--XSLT stylesheet -->
<xsl:element name="img">
      <xsl:attribute name="src">
         <xsl:value-of select="frontpage"/>
      </xsl:attribute>
 </xsl:element>
```

Eksempel 3-1: Hvordan man kan sette inn et bilde i DPG1.0

Å vise frem et bilde er, som vist i eksempelet over, en ganske enkel operasjon i HTML. Alle nettlesere støtter oppbyggingen av elementet, og elementet vil, etter all sannsynlighet, ikke endre struktur i nærmeste fremtid. Da kan en leve med en slik løsning som DPG 1.0 har implementert. Det er selvsagt negativt at en må skrive elementet om igjen hver gang en skal sette inn et bilde, men siden sjansen for oppdatering er liten og det er relativt få linjer HTML kode, har dette vært en akseptert løsning.

Problemene starter hvis en ønsker andre mediatyper, som f.eks. video, på en nettside. Hovedproblemet er at ikke alle nettlesere følger samme standard. I tillegg må man oppdatere elementet hvis det kommer en ny versjon av spilleren som benyttes til avspilling av videoen. Man bruker HTML elementet `object` når man skal vise en video i nettleseren (Se Eksempel 3-2). Microsoft sin Explorer nettleser benytter seg av den ustandardiserte attributten `classid`. Denne attributten forteller nettleseren hvordan ressursen, `object` elementet inneholder, skal behandles. Verdien i `classid` attributten i eksempelet tilsier at nettleseren skal benytte Quicktime sin mediaspiller til å spille av ressursen. Problemet oppstår hvis Quicktime kommer med en ny versjon av mediaspilleren. Da må `classid` attributten endres for å samsvare med den nye spilleren. Alle andre nettlesere bruker mime-typen til filmen som skal vises i stedet for class-id attributten. Dette er grunnen til at det er nøstet et `object` element inne i det første `object` elementet. Attributten codebase har samme problem. Velger apple å endre adressen til hvor man kan laste ned Quicktime plugin-et, må også dette feltet endres.

```
<object classid="clsid:02bf25d5-8c17-4b23-bc80-d3488abddc6b"
   codebase="http://www.apple.com/qtactivex/qtplugin.cab"
   height="256"
   width="320">
      <param name="src" value="<<pathToFile>>/example.mov">
      <param name="scale" value="tofit">
      <param name="controller" value="true">
      <param name="autoplay" value="false">
       <!--[if !IE]>-->
         <object data="<<pathToFile>>/example.mov"
     height="256"
     type="video/quicktime"
     width="320">
            <param name="scale" value="tofit">
            <param name="controller" value="true">
            <param name="autoplay" value="false">
         </object>
       <!--<![endif]-->
</object>
```

Eksempel 3-2: Hvor komplekse HTML elementer for å sette inn video kan være.

Som Eksempel 3-2 viser, er `object` elementet til en Quicktime video ganske avansert, og det er stor sannsynlighet for at en må endre på elementet etterhvert som tiden går. En annen ulempe med DPG1.0 er at man blir ganske låst til å bruke Quicktime filmer hvis man definerer et slikt `object` element i transformasjonsarket til et bestemt mønster. Det finnes mange ulike videoformat som det er ønskelig å benytte seg av. Men siden object elementet endres utfra hvilket format som benyttes, blir dette veldig vanskelig.

Det finnes mange andre eksempler på avanserte HTML elementer som viser multimediaobjekter i nettleseren. På bakgrunn av det som er gjennomgått her, ser man at DPG 1.0 sin håndtering av multimedia er ganske tungvint. Det kreves at mønsterutvikleren kan skrive HTML-kode for alle mediatypene som utvikleren vil ha med i mønsteret, noe som ofte vil være et urimelig krav. Utvikleren må også skrive HTML-kode for hvert eneste multimediaelement som skal være med i mønsteret. Lager man et mønster der man vil ha inn endel videoer, er det lett å se at det blir mye arbeid i transformasjonsarkene. Det blir mye repeterende HTML-kode som det igjen er stor fare for må oppdateres på et seinere tidspunkt.

3.4 Grunnlag for reimplementasjon av DPG 1.0

Her følger en grov gjennomgang av årsakene til at det ble besluttet å utvikle en ny applikasjon. Ingen av punktene nedenfor var i seg selv nok til å utvikle applikasjonen på nytt, men en totalvurdering av alle punktene førte til denne beslutningen. For en mer utfyllende gjennomgang se kapittel 4 i [7] .

Nye krav

DPG 1.0 var ferdigstilt i desember 2004. Siden den gang har det vært stor utvikling på mange områder innen webprogrammering. Det har igjen ført til at det er kommet endel nye krav til

den nye applikasjonen. Et eksempel på dette er krav til multimediahåndtering, som denne oppgaven tar for seg.

Ny teknologi

Det har også kommet mye ny teknologi siden DPG 1.0 ble laget. Rammeverk er ofte gunstig å benytte seg av siden det som regel er utviklet av profesjonelle utviklere og testet ut i mange applikasjoner. Her kan det for eksempel nevnes Spring Application Framework [33], et rammeverk som er ønskelig å bruke i den nye applikasjonen.

Ny mønsterspesifikasjon

Etter at DPG 1.0 kom i drift i 2004 har det vært jobbet med å forbedre applikasjonen. Ett av områdene det har vært jobbet, med er mønsterspesifikasjonen. Alessandro Rossini og Graziano Liberati har skrevet en rapport hvor de kommer med et forslag til ny mønsterspesifikasjon, se [5]. Denne spesifikasjonen skal danne grunnlaget for enda en ny mønsterspesifikasjon i den nye applikasjonen.

Dårlig arkitektur

Som nevnt i seksjon 3.2, er ikke arkitekturen til applikasjonen god nok. For få lag gjør at ulike ansvarsområder blir samlet i samme lag og klasser. Dette gjør refaktorering av koden vanskelig siden det er høy kobling og mange avhengigheter mellom klassene og lagene. Mangel på enhetstester gjør saken enda verre. Implementasjon av multimediafunksjonalitet som vil være en ny funksjonalitet for systemet, blir veldig problematisk siden applikasjonen har tette koblinger mellom lagene med mange avhengigheter. Det blir da vanskelig å sette seg godt nok inn i systemet til å få utviklet en tilfredstillende modul for multimedia.

Mangelfull Dokumentasjon

Manglende dokumentasjon av arkitektur, kode og valg som er foretatt under implementeringen, er nok det argumentet som veide tyngst da beslutningen om at applikasjonen skulle skrives på nytt, ble foretatt.

4 Det nye systemet

4.1 Overordnet krav til multimedia

Som nevnt tidligere hadde ikke DPG 1.0 noen støtte for rendring av multimediaobjekter. Man kunne laste opp mediafiler manuelt til serveren, for så å bygge opp HTML elementet i transformasjonsarket. Dette måtte da gjøres for hver mediafil som skulle vises i mønsteret.

Det var kanskje til å leve med på den tiden da DPG1.0 ble implementert. Kravene for multimedia var stort sett å vise frem bilder. Men med dagens teknologi både innenfor multimedia, men også innenfor bredbånd til privatpersoner, er kravene til multimedia mye større. Dette fører også til et behov for at DPG-en har innebygget støtte for å hjelpe utviklerne av mønstre og presentasjoner med å håndtere multimedia.

Kravene til DPG 2.0 innenfor multimedia beskrives som at DPE-en skal kunne rendre HTML elementer for de mest brukte multimediaobjektene på nettet i dag (Se Tabell 4-1). I tillegg skal PCE-en tilby opplasting av multimediafiler og oppdatere innholds XML treet der objektet skal benyttes.

Media type	**Format**
Audio	MPEG-1 Audio Layer 3 MP3 (.mp3)
Bilde	Joint Photographic Experts Group JPEG (.jpg .jpeg) Portable Network Graphics PNG (.png) Graphics Interchange Format GIF (.gif)
Video	Quicktime (.mov) Audio Video Interleave AVI (.avi) MP4 (.mp4) 3GP (.3GP) Moving Picture Experts Group Phase 1 MPEG-1 (.mpg) Windows Media Video WMV (.wmv) Flash Video (.flv)
Flash animasjoner	Shockwave Flash (.swf)
PDF presentasjoner	Portable Document Format PDF (.pdf)
Kart	Google maps
Javakode til HTML	Java fil (.java)

Tabell 4-1: Oversikt over multimediakravene til DPG 2.0

I tillegg til disse kravene er det også andre typer multimediakrav til systemet. En av svakhetene ved DPG 1.0 var at man ikke kunne bytte ut en video med en video med et annet format. Grunnen til dette var at HTML for multimediaobjektene ble generert i transformasjonsarket. Transformasjonsarket hører til mønsteret, dermed kan man ikke endre

HTML-kode i hver presentasjon. Og som nevnt tidligere, endres `object` elementet når videoformatet endres. Dette er nå et krav i den nye DPG 2.0. Man skal kunne bytte ut en video med en video av et annet format.

Det fantes ingen mulighet for å gruppere mediaelementer, som for eksempel kunne danne en musikkspilleliste som en flashspiller kan håndtere, i DPG 1.0. Dette er også et krav som har kommet til i DPG 2.0. Man skal ha mulighet til å gruppere like mediaelementer sammen i en liste som kan brukes til f.eks. bildegalleri, musikkspilleliste eller videospilleliste.

En annet viktig funksjonalitet som er blitt et krav til DPG 2.0, er krav om at det skal være enkelt å oppdatere rendringsmotoren ved oppdatering av eksisterende format eller forekomst av nye format. Som nevnt delkapittell 3.3, må man ofte oppdatere HTML koden hvis for eksempel Quicktime kom med en ny videospiller. Det er da hensiktsmessig at en slik oppdatering vil være en enkel operasjon der en kun endrer kode et sted i applikasjonen. Det samme gjelder hvis det for eksempel kommer et nytt videoformat som blir populært å bruke på nettet. Det er da et krav om at systemet skal tilpasses slik at man enkelt kan implementere støtte for det nye formatet.

4.2 Vurdering av mulige løsninger på multimediaproblematikken

Flere løsninger ble vurdert i begynnelsen av prosjektet. En av løsningene som ble vurdert var å lage multimediafunksjonalitet som en del av kjernefunksjonaliteten til systemet. Det vil si at all multimedia håndtering foregikk i rendringsmotoren til systemet.

Det er en løsning som selvsagt er bedre enn det DPG 1.0 kunne tilby, men løsningen er likevel ikke helt tilfredstillende. De største problemene med denne løsningen er at elementer som benyttes i HTML for å vise multimediaobjekter, har egenskaper som kan endre seg over tid. I tillegg kan det komme nye behov innen multimedia som applikasjonen bør kunne takle på en enkel måte. Det er derfor ingen god løsning å implementere multimedia som kjernefunksjonalitet, siden små endringer og ny multimediafunksjonalitet krever at man må legge til kode i hovedapplikasjonen. Dette vil da føre til at hele systemet må kompileres på nytt, lastes opp til serveren og kjøre en omstart på serveren.

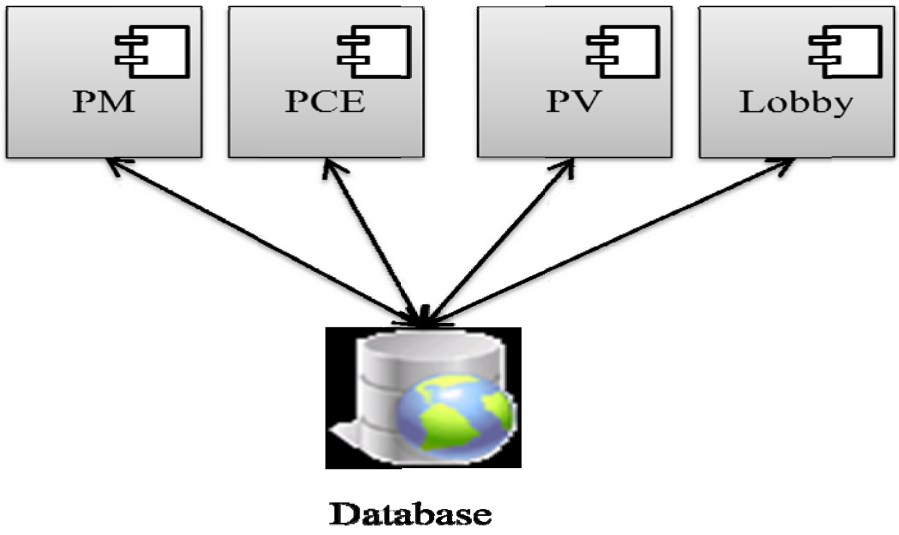
Løsningen som til slutt ble valgt, var å implementere multimediahåndtering ved hjelp av en pluginsmodul eller programvareutvidelsesmodul. Konseptet er at man utvikler små programsnutter som kan kobles til hovedapplikasjonen og har til oppgave å rendre nødvendig HTML for å oppnå ønsket multimediafunksjonalitet. Den største fordelen med dette designet er at man kan legge til nye plugins eller oppdatere eksisterende plugins uten at man trenger å rekompilere og laste opp applikasjonen på nytt. Man trenger riktignok en restart av serveren for at nye plugins skal fungere, men det er lite å betale i forhold til at man får ny funksjonalitet til applikasjonen.

4.3 Overordnet arkitektur i DPG 2.0

Delsystem

Som vist i Figur 2-11 var DPG 1.0 delt inn i to delsystem, DPE og RAT. Det nye systemet fordeler hovedarbeidsoppgavene til tre ulike delsystem. I tillegg er det et mindre delsystem ved navn lobby. (Se Figur 4-1)

- Presentation Content Editor (PCE)
- Presentation Manager (PM)
- Presentation Viewer (PV)
- Lobby



Figur 4-1: Oversikt over delsystemene i DPG 2.0

Årsaken til den nye inndelingen er å få en klarere logisk inndeling av subsystemene, uten at subsystemene får for store ansvarsområder. PV sin oppgave er å vise frem informasjon i en presentasjon PCE sin oppgave er å tilby editering av informasjon i en presentasjon, og PM sin oppgave er å tilby brukeren å endre egenskaper ved en presentasjon. De tre hoveddelsystemene bruker et sentralt repositorium som er implementert som et Java Content Repository [34].

Presentation Viewer (PV)

PV har som oppgave å rendre innholdet i presentasjoner, det vil si prossessere informasjonen i presentasjoner og sende det som ferdig XHTML kode til nettleseren. I tillegg har den støtte for å bygge opp editeringslinker for hver entitet og en link til editoren, hvor den aktuelle presentasjonen figurerer på forsiden.

Presentation Content Editor (PCE)

PCE har som oppgave å hjelpe brukere, med publisher eller administrator rettigheter, å editere innhold i presentasjoner. Med innhold menes alt innhold, ikke bare tekst. Det vil si at den har

egen ressurshåndtering hvor brukerne har mulighet til å laste opp ressurser som skal benyttes i en presentasjon. Den tilbyr også endel andre operasjoner, som å slå av og på views og endre på etiketten til et view.

Presentation Manager (PM)

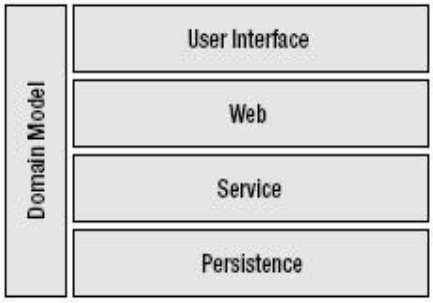
PM tilbyr funksjonalitet for å konfigurere en presentasjon, opprette en ny presentasjon og kopiere en eksisterende presentasjon. Med konfigurasjon menes operasjoner som å endre presentasjonsdetaljer som tittel, beskrivelse og brukergruppe, overkjøre en del komponenter i mønsteret som layout, transformasjoner og stil og å slette presentasjonen.

Lobby

Lobby er et lite delsystem som har to hovedoppgaver. Den første oppgaven går ut på å utføre autentisering og autorisering av brukerne. Den andre oppgaven går ut på å vise inngangsportalen til systemet. Det vil si, vise tilgjengelige presentasjoner for hver enkelt bruker basert på rettighetene til brukeren. I tillegg har inngangsportalen linker til PCE-en og til PM-en.

Lagdeling

DPG 2.0 er bygget opp som en Spring Model View Controller (MVC) applikasjon [35]. En slik applikasjon er vanligvis delt opp i fem lag (se Figur 4-2 hentet fra [35]. Et lag er en modul som jobber med et bestemt problemområde. Et eksempel er persistenslaget som arbeider med lagringsproblematikken. Hvert lag er skilt med interfaces som er en kontrakt for hvordan lagene skal kommunisere seg imellom. Enkelte lag blir bare brukt av laget over seg i hierarkiet, mens for eksempel laget Domain Model kommuniserer med nesten hele applikasjonen.

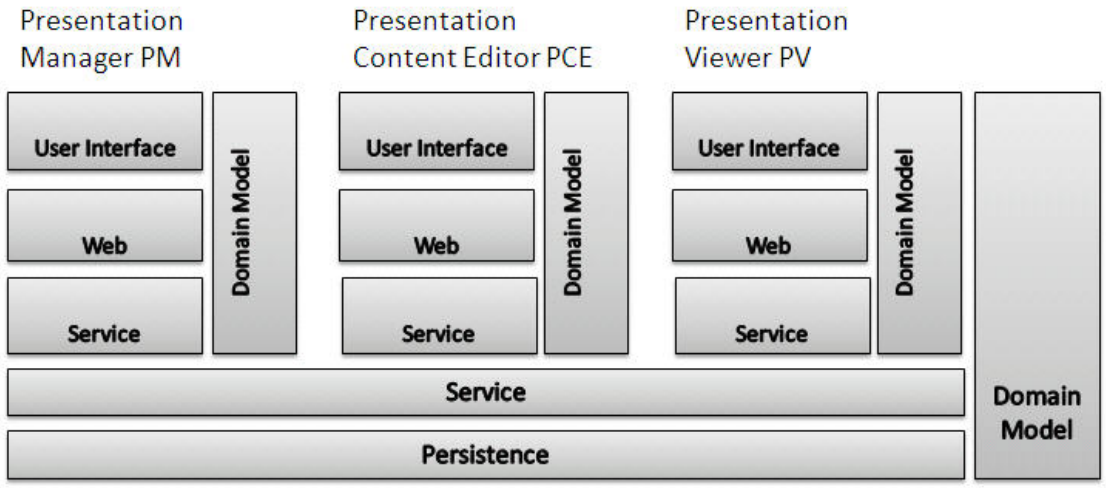


Figur 4-2 Lagdelingen i en Spring MVC applikasjon

Spring rammeverket sin rolle i applikasjonen er at den tilbyr et lag som oversetter mellom HTTP verden og forretningslogikken i domenemodellen. Fordelen er at man kan benytte seg av Plain Old Java Objects (POJOs) [35] til å bygge objektorientert kode i domenemodellen. Bruken av objektorientert utviklingsteknikker gjør systemet testbart og fleksibelt.

Spring hjelper også til med opprettelse av objekter og definering av avhengigheter mellom dem. Dette gjøres i Spring sin `ApplicationContext`, som er et objektregister og integrasjonspunkt. Dette fører igjen til en fleksibel og testbar applikasjon.

DPG 2.0 inneholder, som nevnt tidligere i delkapittelet, tre ulike delsystem. Lagdelingen i applikasjonen ser dermed ut som i Figur 4-3. Meningen er å ha egne User Interface lag, Web lag, Service lag og Domain Model lag for hver delapplikasjon. Grunnen er at det gir en klarere struktur siden hver delapplikasjon har endel funksjonalitet som er spesifikt for det bestemte delsystemet. I tillegg har applikasjonen et Service-, Persistens- og Domain Model lag som inneholder klasser som er felles for hele applikasjonen.



Figur 4-3: Lagdelingen i DPG 2.0. Kun de tre hoveddelsystemene er tatt med.

Persistenslaget inneholder dataaksessobjekter. Her foregår all kommunikasjon mellom applikasjonen og databasen. Persistenslaget tilbyr et grensesnitt som f.eks. servicelaget må implementere for å få tilgang til data i databasen.

Servicelaget oppdaterer datamodellen i applikasjonen. Metodene i dette laget utfører transaksjonelle enheter med arbeid, hvor en kommuniserer med persistenslaget for å bygge en korrekt datamodell utfra en forespørsel. Et eksempel er hvordan servicelaget kommuniserer med dataaksessobjektene og Domain Model-laget for bygge opp en presentasjon av et view.

Domain Model laget inneholder modellen som servicelaget oppdaterer. Domeneobjekter og forretningslogikk er samlet her som POJOs. Eksempler på dette er klasser som inneholder modellen av et mønster.

Weblaget i hver delapplikasjon inneholder klasser som implementerer Spring sitt `org.springframework.web.servlet.mvc.Controller` grensesnitt. En kontroller er ansvarlig for å akseptere `HttpServletRequest` objekt og `HttpServletResponse` objekt. Den kan så utføre en del operasjoner med for eksempel `get` og `post` variablene før den overlater kontrollen til et view. Et eksempel her er hvordan `PresentationController` i delsystemet PV henter ut informasjon fra `get` variabelen for å finne ut hvilken side i hvilken presentasjon som er forespurt.

UserInterfacelaget inneholder informasjon om hvilken rendringsteknologi som skal benyttes. En kontroller i weblaget mapper resultatet av en forespørsel til et view med et logisk navn. I DPG2.0 implementeres grensesnittet `org.springframework.web.servlet.ViewResolver` slik at det logiske navnet blir mappet til en JSP side med samme navn.

5 Presentasjonsmønstre i DPG 2.0

Presentasjonsmønstre er et konsept der man definerer en fast struktur til en side slik at man enkelt kan lage nye presentasjoner basert på denne strukturen. Man skiller mellom struktur og innhold, noe som gjør at strukturen kan gjenbrukes med nytt innhold i en ny presentasjon.

Presentasjonsmønstre inneholder to spesifikasjoner. Den første spesifiserer strukturen dataene skal ha, det vil si hvordan data skal settes sammen for å gi meningsfull informasjon. Den andre spesifiserer hvordan sidene i hver enkelt presentasjon skal se ut. Man kan ha mange spesifikasjoner som definerer presentasjonen av en webside per spesifikasjon av struktur.

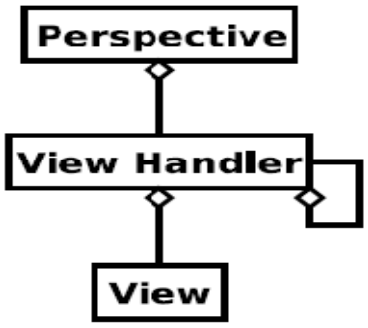
5.1 Presentasjonsmønster utviklet av Alessandro Rossini og Graziano Liberati

Presentasjonsmønstre i DPG 1.0 ble gjennomgått i kapittel 2, men det har vært en utvikling i presentasjonsmønsterspesifikasjonen siden den første versjonen av DPG var implementert.

Alessandro Rossini og Graziano Liberati utviklet en ny spesifikasjon av presentasjonsmønstre som de dokumenterte i [36]. Denne spesifikasjonen var utgangspunkt for presentasjonsmønsterspesifikasjonen i det nye systemet. Hovedmålet med denne mønsterspesifikasjonen var å definere begreper som er logiske, enkle og definerer klare grenser. Her følger en liten gjennomgang av den spesifikasjonen.

Spesifikasjonen består fremdeles av tre deler, men med forskjellig terminologi: Se Figur 5-1 .

- Views
- View handlers
- Perspectives



Figur 5-1: Hvilke deler en mønsterspesifikasjon er delt inn i. Hentet fra [36].

View

Ideen bak `view` er at man utfra domeneområdet hvor mønsteret skal benyttes kan definere ulike entiteter. På den måten får man en logisk abstraksjon til domeneområdet når strukturen

til dataene defineres. Eksempel 5-1 viser et eksempel på hvordan et legeview kan defineres. Her sier man at en lege skal defineres med fornavn, etternavn og legekontor.

```
<view id="lege">
      <fields>
            <field>fornavn</field>
            <field>etternavn</field>
            <field>legekontor</field>
      </fields>
</view>
```

Eksempel 5-1: Oppbyggingen av et view

View handler

Hvert view bli tatt hånd om av en `view-handler`. Her er det mulig å filtrere ut elementer eller å oppdatere XML strukturen. Eksempel 5-2 viser et eksempel på hvordan man definerer en `view-handler`. Først så mappes `view`-et som har id lik lege til `viewhandler`-en. Det neste som skjer er at man filtrerer vekk feltet legekontor. Under filter kommer en ny begrensning, her blir alle som ikke heter Gundersen til etternavn luket bort. I siste steget kan man gjøre ytterligere transformasjoner på XML-treet, det kan være å oppdatere XML-strukturen, filtrere på nytt eller sortere elementene utrfra et gitt kriterium.

```
    <view-handler id="legePresentasjon">
        <associated-views>
            <associated-view>lege</associated-view>
        </associated-views>
        <fields>
            <field>lege.fornavn</field>
            <field>lege.etternavn</field>
        </fields>
        <filters>
            <filter>
                <field>lege.etternavn</field>
                <value>Gundersen</value>
            </filter>
        </filters>
        <transformations>
            <transformation>sorterLeger</transformation>
        </transformations>
    </view-handler>
```

Eksempel 5-2: Oppyggingen av en view-handler

Perspective

Et `perspective` definerer innholdet som skal være på en bestemt side. Et `perspective` setter sammen ulike `view-handlere` som tilsammen utgjør den informasjonen som skal

vises på en side. Eksempel 5-3 viser et eksempel på denne mekanismen. Her defineres først en beskrivelse av siden. Under schema settes referansen til et XSD-dokument som spesifiserer strukturen på XML-treet som genereres. I denne spesifikasjonen er XSD-dokumentet veldig viktig, siden det er eneste kilden presentasjonsdesignerne kan søke etter strukturen på informasjonen som skal presenteres. I composition elementet defineres `view-handler`-ne som skal vises på en side. I dette enkle eksempelet er det bare tatt med en `viewhandler`, men det kan selvsagt være flere. Under view-handler-dependencies kan en sette at en `viewhandler` bare skal vises hvis en annen `view-handler` er valgt.

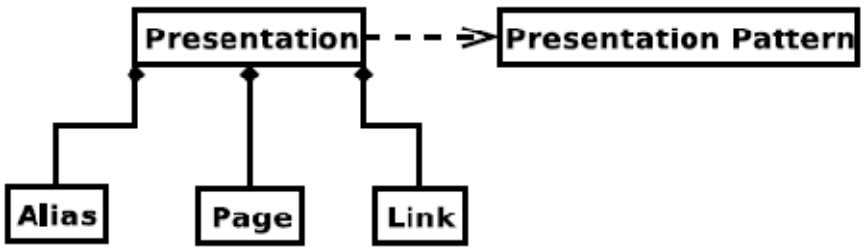
```
<perspective id="legePerspective">
    <description>
      Her vises en fiktiv side som inneholder fornavn og etternavn på leger
      med etternavn Gundersen
    </description>
    <schema>legePageXSD</schema>
    <composition>
        <content>legePresentasjon</content>
    </composition>
    <view-handler-dependencies>
        <view-handler-dependency>
        </view-handler-dependency>
    </view-handler-dependencies>
</perspective>
```

Eksempel 5-3: Oppbyggingen av et perspective

Presentasjonsspesifikasjon

Hver presentasjon har sin egen presentasjonspesifikasjon. Her setter man egenskaper som kan variere fra presentasjon til presentasjon. Også her består spesifikasjonen av tre deler:(Se Figur 5-2)

- Alias
- Page
- Link



Figur 5-2: Oppyggingen av en presentasjonsspesifikasjon

Alias

`Alias` har som oppgave å mappe `views,` som er definert i presentasjonsmønster spesifikasjonen, til entiteter i persistenslaget. I Eksempel 5-4 vises denne funksjonaliteten. Først defineres `view`-et man ønsker å mappe, hvilket persistensteknologi som benyttes og URL til hvor persistenslaget befinner seg. Det neste som skjer, er at man definerer attributten som man ønsker å mappe, i dette tilfellet fornavn. Inne i elementet mapping sier man hvilken databasetabell og hvilken attributt fornavn skal mappes til, i dette tilfelle tabellen legeTabell og attributten fornavn. Dette eksempelet viser oppsett av `alias` elementet når database teknologi er valgt, andre teknologier som feks XML kan også benyttes.

```
<alias
    name="lege"
    kind=”DB"
    url="http://ii.uib.no/jafuDB">
     <attributes>
        <attribute name="fornavn"/>
        <mapping>
            <entity>legeTabell</entity>
            <attribute>fornavn</attribute>
        </mapping>
     </attributes>
</alias>
```

Eksempel 5-4: Oppbyggingen av elementet alias

Page

I elementet `page` kan man definere layout og utseende for en side i presentasjonen slik som det er vist i Eksempel 5-5. Dette er en mapping mellom et `perspective` og en bestemt rendringsteknologi. Det første som skjer, er at man definerer en id til elementet, om siden skal være standardsiden, en beskrivelse av siden, og om siden skal være tilgjengelig i presentasjonen. I elementene perspective og layout skjer mappingen mellom et perspective og rendringsteknologien.

```
<page
     id="legeListe"
     default="true"
     description="Side som viser en legeoversikt"
     available="true">
     <perspective>legePerspective</perspective>
     <layout>legeLayout.jspx</layout>
</page>
```

Eksempel 5-5: Oppbyggingen av elementet page

Link

Navigasjon mellom de forskjellige sidene i en presentasjon blir definert ved hjelp av et `link` element. Det første man gjør er å definere siden man skal linke fra. I elementet destination-pages listes sidene som det skal linkes til.(Se Eksempel 5-6)

```
<link sourcePage="legeListe">
    <destination-pages>
        <page>legeKontorListe</page>
        <page>sykehusListe</page>
    </destination-pages>
</link>
```

Eksempel 5-6: Oppyggingen av elementet link

5.2 Analyse av mønsterspesifikasjon

Som nevnt tidligere var mønsterspesifikasjonen, som er gjennomgått i forrige delkapittel, grunnlag for den nye spesifikasjonen. Her følger en analyse av gode og dårlige sider ved Alessandro Rossini og Graziano Liberati sin mønsterspesifikasjon.

View

Begrepet view ble mye diskutert av medlemmene i prosjektgruppen. Begrepet er ment å definere domeneelementer, som for eksempel en artikkel, noe som prosjektgruppen slett ikke assosierte med view. View betegner mer et utsnitt av informasjon som skal vises etter at transformasjonen har blitt utført.

Utenom selve begrepet view er ideen om å definere domeneelementer en enkel og forståelig måte å strukturere dataene på. Man får en logisk struktur som gjør det enkelt å lese og som letter arbeidet med å utvikle nye mønster.

View handler

Viewhandler er konseptuelt et veldig viktig lag i mønsterspesifikasjonen. Et nivå mellom view og perspective der en har mulighet for filtrering, sortering og ellers endring av XML-strukturen, gir en veldig fleksibel oppbygging. Ved at man kan opprette flere viewhandlere basert på samme data, men som viser informasjonen forskjellig, øker også fleksibiliteten.

Men det er ikke så lurt å ha egne løsninger på ting som det finnes velutprøvd teknologi for å løse. Både filtrering av felt alene og filtrering av felt som inneholder en viss verdi, er oppgaver som bør taes hånd om i transformasjonsarket som er definert i elementet transformation.

Perspective

Perspective modellerer en side i applikasjonen. Konseptet er veldig fleksibelt siden man kan definere hvilke viewhandlere som skal være med på siden.

Det man kan peke på som negativt, er bruken av skjema. Det er ingenting i mønsteret som sier noe om den endelige XML-strukturen til et perspective. Mønsteret definerer en overordnet struktur, men etter at viewhandlerne har utført sine oppgaver, endres denne strukturen. Den eneste muligheten en presentasjonsdesigner kan vite om XML-strukturen et perspective genererer, er gjennom skjemaet som er definert i elementet schema. Dette kompliserer

oppgaven med å lage presentasjoner siden strukturen på innholdet ikke følger logisk fra mønsteret.

Alias

Dette er også en god ide i teorien. Det gir en enorm fleksibilitet at man kan mappe hvert enkelt felt til helt forskjellige datakilder. Men det oppstår også endel problemer og merarbeid når man ikke har et eget persistenslag.

Det mest innlysende er at det blir tidkrevende å lage en presentasjon basert på et mønster som har mange views. Hvert felt krever egen mapping til en datakilde, noe som selvsagt fører til mye arbeid på XML fronten. Selv om man lager en editor for arbeidet, vil det fremdeles være tidkrevende.

Et annet problem er at det er vanskelig å håndtere dataintegritet, feilsøking og tilgjengelighet hvis en har mange datakilder å forholde seg til.

Page

Page har også noen smarte løsninger. En kan foreksempel velge å deaktivere sider i en presentasjon. Dette er en nyttig funksjon hvis en side i presentasjonen ikke er helt ferdig, eller inneholder informasjon som ikke skal være tilgjengelig før en viss dato. En annen fordel er at man har mulighet til å ha egne layoutfiler for hver side i en presentasjon. Det gjør at layout og utseende kan spesifisere forskjellig fra side til side og fra presentasjon til presentasjon.

Men det kan også være tilfeller der man utfra ett bestemt mønster ønsker presentasjoner med samme layout. Man vil for eksempel bare endre farger og logoer. Det har man ikke støtte for her siden man i hver presentasjon må definere egne layout filer for hver side i en presentasjon.

En annen ulempe som angår fleksibilitet er at man ikke har mulighet for å slå av og på viewhandlere i en presentasjon. Denne funksjonaliteten er blant annet hensiktsmessig hvis man har en side i en presentasjon hvor en liten del av siden ønskes å holdes skjult inntil videre.

Link

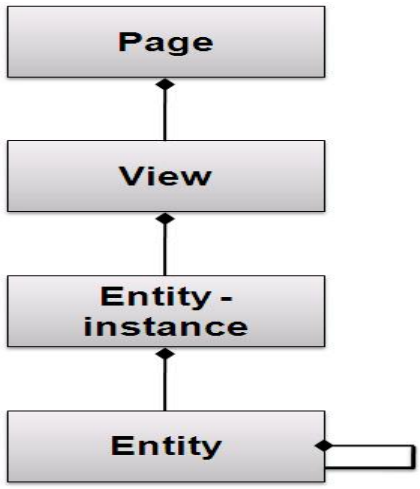
Link er et konsept der man bygger applikasjonens navigasjonhierarki. Det positive er at presentasjonsbyggeren har full kontrol over navigasjonen i applikasjonen. Det vil si at navigasjonen er uavhengig av mønsterspesifikasjonen.

5.3 Mønsterspesifikasjon i DPG 2.0

På bakgrunn av analysen som er foretatt av DPG 1.0 og av Alessandro Rossini og Graziano Liberati sitt forslag til nytt presentasjonsmønster, er det utarbeidet en ny presentasjonsmønster spesifikasjon. I denne seksjonen følger en presentasjon av denne spesifikasjonen.

Presentasjonsmønsteret må spesifisere fire komponenter (se Figur 5-3):

- Entities
- Entity-instances
- Views
- Pages



Figur 5-3: Komponentene som må være med i et presentasjonsmøsnter

Entity

Entiteter skal gjenspeile domeneobjekter i den konteksten som mønsteret er skrevet for. Ser man på elementet `articleEntity` i Eksempel 5-7 er dette en modell av en artikkel. Artikkelen deles her opp i mindre enheter eller felt, som krever spesiell formatering når det skal rendres.

```
<entity id="articleEntity">
    <fields>
      <field type="string" required="true">title</field>
      <field type="xhtml">ingress</field>
      <field type="plugin" pluginConfig="singleVideo">video</field>
      <field type="plugin" pluginConfig="singleImage">image</field>
      <field type="xhtml">content</field>
    </fields>
</entity>

<entity id="listOfArticlesEntity">
   <fields>
      <field type="string" required="true">title</field>
      <field type="entity-list" ref-id="articleEntity">articles</field>
   </fields>
</entity>
```

Eksempel 5-7: Oppbyggingen av et entity element

Attributten `type` sier noe om formatet på informasjonen som er mappet til det bestemte feltet. Det kan ha følgende verdier:

- string — Feltet kan ikke inneholde formatering, bare ren tekst. Editoren viser et tekstfelt som kan editeres når et felt har typen `string`.
- Date — Feltet inneholder en dato i formatet `dd/mm/åååå`. Editoren viser en grafisk presentasjon av en kalender når et felt har typen date.
- File — Feltet inneholder et filnavn. Det er opp til XSLT transformasjonsarket å bestemme hvordan filen skal presenteres. Editoren viser en opplastningsboks når et felt har typen `file`.
- xhtml — Feltet kan inneholde formatering uten at den blir escapet. Eksempel på slik formatering er HTML-taggen `<b>` som viser tekst i fet font. Editoren viser en WYSIWYG (What You See Is What You Get) editor når et felt har typen `xhtml`.
- plugin — Feltet inneholder informasjon som skal rendres på en måte som ikke støttes av hovedrendringsmotoren, men av plugins som er koblet til applikasjonen. Pluginsmodulen er utviklet for å håndtere og fremvise multimediaobjekter. Editoren viser enten en opplastningsboks for multimediaobjekter eller et tekstfelt når et felt har typen `plugin`. Det er under konfigurasjonen av pluginet man bestemmer hva som skal vises i editoren.
- entity-list — Et felt med typen `entity-list` har en litt annen funksjon enn de andre typeattributtene. Ideen er at man i en entitet kan bygge opp en liste med innhold fra en annen entitet. Ser man på entiteten `listOfArticlesEntity` i Eksempel 5-7 bygges det en liste av `articleEntity` entiteter. Dette gjøres ved at attributten `ref-id` linker `articleEntity` til listen av entiteter.
- entity — Et felt med typen entity bygger på samme ideen som et felt med typen entity-list. Forskjellen er at man i en entitet ikke vil ha en liste med en bestemt entitet, men i stedet kun en representant av entiteten det linkes mot. Ser man på Eksempel 5-7 og bytter ut typeattributten entity-list i feltet articles med type attributtet entity, betyr det at elementet articles inneholder kun en articleEntity entitet.

Et `field` element kan også ha attributten `required`. Denne kan inneholde de boolske verdiene `true` eller `false`. Meningen med attributten er at man kan si eksplisitt, hvis verdien er `true`, at man må gi dette feltet innhold.

Attributten `pluginConfig` er det siste attributten som er lovlig i elementet `field`. Denne attributten inneholder et logisk navn på et plugin som skal benyttes for å rendre innholdet som

mappes til dette feltet. Attributten prossesseres kun hvis et `field` element har typen `plugin` eller typen `entity-list.`

Entity-instance

`Entity-instance` er en instans av en entity. Attributten `id` i `entity-instance` elementet peker på et XML-dokument hvor innholdet som skal rendres ligger. Strukturen på dette dokumentet er definert av entiteten som er referert til i elementet `entity` . I Eksempel 5-8 peker attributten id, i `entity-instance` elementet, på et XML-dokument med samme id. Strukturen på dette dokumentet må følge strukturen i `listOfArticleEntity` som er definert i Eksempel 5-7.

```
<entity-instance id="listOfArticlesEntityInstance">
    <entity>listOfArticlesEntity</entity>
</entity-instance>
```

Eksempel 5-8: Oppbyggingen av et entity-instance element

Eksempel 5-9 viser et eksempel på et XML dokument som elementet `entity-instance` i Eksempel 5-8 kan referere til. Dokumentet følger strukturen som er diktert av entiteten `listOfArticleEntity` i Eksempel 5-7. Innholdet som skal rendres legges inn i de ulike elementene.

Eksempelet viser også hvordan `type` attributtene blir brukt i praksis. Ser man på elementet `title` har det `type` lik `string`, dermed kan en bare legge inn ren tekst som innhold.

Elementet `articles` har `type` lik `entity-list`, og som Eksempel 5-7 viste skal dette elementet da inneholde en liste med elementet `articleEntity`.

Elementene `ingress` og `content` har type lik `xhtml`. En kan da pakke innholdet i disse elementene inn i `xhtml` markup.

Elementene `video` og `image` har `type` lik `plugin`. Innholdet i disse elementene blir sendt til plugins for videre prossessering.

```
<listOfArticlesEntity type="rootEntity"
   <title type="string" required="true">Nyheter</title>
   <articles type="entity-list">
      <articleEntity type="subEntity">

         <title type="string">
            LUCAS TO LINE UP IN TONIGHT'S FINAL
         </title>
         <ingress type="xhtml">
            <![CDATA[<p>Brazilian international Lucas has been included in
            Gary Ablett's reserve team for the play-off final against Aston
            Villa at Anfield tonight - a match you can watch live on LFC TV
            from 7.30pm GMT.</p>]]>
         </ingress>
         <video type="plugin">nyhet_1.flv</video>
         <image type="plugin">nyhet_1.jpg</image>
         <content type="xhtml">
            <![CDATA[  <p>The  midfielder  replaces  the  injured  <b>Jay
            Spearing</b> in the team and Emiliano Insua also starts at left
            back. Moroccan international Nabil El Zhar is also back in the
            team after injury.</p>]]>
          </content>
      </articleEntity>

      ...

    </articles>

</listOfArticlesEntity>
```

Eksempel 5-9: Et innholdsdokument som følger strukturen fra Eksempel 5-7

View

`Entity-instances` blir håndtert av `views`, se Eksempel 5-10. Et `view` skal modellere et utsnitt på en side, for eksempel en liste med artikler, meldingsboks osv. Elementet `entity-instance` inneholder en referanse til en bestemt `entity-instance`, som igjen peker på et XML dokument. Elementet `transformation` inneholder en referanse til et XSLT stilark. Stilarket opererer på XML dokumentet og kan utføre blant annet filtrering av elementer, sortering og ikke minst strukturere innhold i XHTML markup. Mulighetene er mange siden XSLT er et kraftig XML basert språk. Det er også viktig å få med at en entity-instance kan bli transformert ulikt i flere forskjellige views.

```
<view id="newsArticleView">
    <description>This view presents all the new articles</description>
    <entity-instance>listOfArticlesEntityInstance</entity-instance>
    <transformation>newsTransformer</transformation>
</view>
```

Eksempel 5-10: Oppbyggingen av et view element

Page

En `page` skal modellere en nettside i applikasjonen (se Eksempel 5-11). Elementet `page` kan ha to forskjellige attributter. `Id` som siden identifiseres av og `default` som kan ha de boolske verdiene `true` eller `false`, som definerer om siden er startside eller ei.

Siden bygges opp av `views` som er definert i elementet `composition`. `Views` inneholder ferdig generert HTML for et bestemt utsnitt av siden. Et view element kan ha to forskjellige attributter . `DefaultView` er en attributt som settes om man vil at viewet skal være standard viewet på siden. `AlwaysVisible` settes om man ønsker at viewet skal være synlig hver gang siden er valgt. Alle `views` som har satt attributtet lik false ender opp i en viewmeny, der bruker må trykke på menyen for å få vist ønsket view.

Siste element som en page kan inneholde er layout. Her legges en referanse til en Velocity-mal [37]. Velocity-malen har til oppgave å definere layouten på siden.

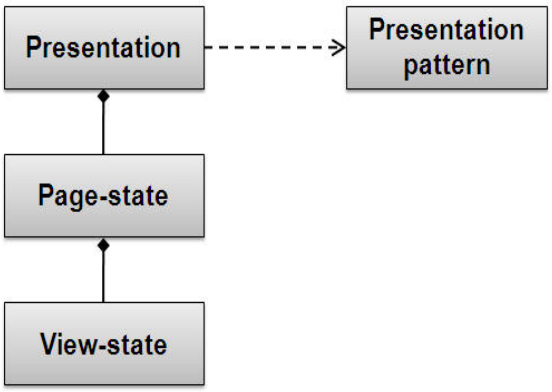
```
<page id="newsPage">
     <description>All the news from the club</description>
     <composition>
       <view defaultView="true">newsArticleView</view>
       <view alwaysVisible="true">lastNewsBoxView</view>
       <view alwaysVisible="false">oldNewsArticleView</view>
       <view alwaysVisible="true">compactTableView</view>
       <view alwaysVisible="true">contactView</view>
       <view alwaysVisible="true">addsView</view>
     </composition>
     <layout>newsPageLayout</layout>
</page>
```

Eksempel 5-11: Oppbyggingen av et page element

5.4 PresentasjonsSpesifikasjon for DPG 2.0

Den nye mønsterspesifikasjonen til DPG 2.0 inneholder også en ny presentasjonsspesifikasjon. Her kan det settes en del egenskaper som ofte er forskjellig fra presentasjon til presentasjon. Følgende elementer må spesifiseres for hver enkelt side i presentasjonen (se Figur 5-4) :

- Page-state
- View-state



Figur 5-4: En visuell presentasjon av presentasjonsspesifikasjonen

I presentasjonsspesifikasjonen er det også spesifiert en del parametre (se Eksempel 5-12). Parametrene inneholder informasjon om hvilke presentasjonsmønster som er brukt, tittelen på presentasjonen, en beskrivelse av presentasjonen og hvilken brukergruppe som skal ha tilgang til presentasjonen.

```
<specification>
      <presentation-pattern>footballPattern</presentation-pattern>
      <title>Liverpool Football Club</title>
      <description>Home page for Liverpool Football Club</description>
      <user-group-id>5</user-group-id>
      ...

</specification>
```

Eksempel 5-12: Spesifikasjon av parametre i presentasjonsspesifikasjonen

Page-state

Eksempel 5-13 viser hvordan man kan definere elementet page-state for en bestemt side. Et page-state element kan ha fire lovlige attributter:

- Id — Id identifiserer siden og må være den samme som id-en til det tilhørende pageelementet i presentasjonsmønsteret.
- Enabled — Enabled kan ha de boolske verdiene true eller false. Her kan en velge om en side skal være tilgjengelig.
- Label — Label er en etikett til siden. Navigasjonen i applikasjonen er autogenerert. Det er en hovedmeny som er lik på alle sider i presentasjonen. Alle page-state elementene i en presentasjon vil ha en link i hovedmenyen med verdien til etiketten.
- Default — Default definerer om siden er startsiden i presentasjonen. Den kan ha de boolske verdiene true eller false.

View-state

Eksempel 5-13 viser også hvordan man definerer elementet view-state for hvert view som en side inneholder. Verdien i elementet view-state er en referanse til det tilsvarende viewet i mønsterspesifikasjonen. View-state kan ha tre lovlige attributter:

- Enable — Enable bestemmer om et view skal være tilgjengelig. En kan slå av og på views alt etter om man vil at de skal vises på siden.
- Label — Label er en etikett for hvert view. Som nevnt tidligere, er navigasjonen i applikasjonen autogenerert. På hver side i applikasjonen vil det være en viewmeny som inneholder linker til views hvor `alwaysVisible` er satt til `false` i mønsterspesifikasjonen. Linkene vil ha verdien til attributtet label.
- Default — Default bestemmer om et view skal være standard viewet på en side. Dette er en overkjøring av default attributtet i mønsteret.

```
<page-state id="newsPage" enabled="true" label="News">

  <view-state enabled="true" label="News">newsArticleView</view-state>
  <view-state enabled="true" label="Last News">lastNewsBoxView</view-state>
  <view-state enabled="true" label="Archive">
      oldNewsArticleView
  </view-state>
  <view-state enabled="true" label="Table View">
      compactTableView
  </view-state>
  <view-state enabled="true" label="Contact Us">contactView</view-state>
  <view-state enabled="true" label="Adds">addsView</view-state>
</page-state>
```

Eksempel 5-13: Hvordan pagestate og viewstate elementene bygges opp.

5.5 Multimedia i presentasjonsmønster

I DPG 2.0 blir multimediafunksjonalitet implementert ved hjelp av en pluginmodul. Denne løsningen setter også en del føringer til mønsterspesifikasjonen. Det er to forskjellige tilfeller der man i mønsteret må ha et begrep der man sier at informasjonen som blir mappet til dette feltet, skal rendres av plugins.

Det første tilfellet er hvis det kun er informasjon som mappes til et enkelt felt som skal håndteres av et plugin. Eksempel på dette kan være en enkel film, bilde eller PDF-fremvisning.

Det andre tilfellet er hvis det er informasjon som er mappet til en liste som skal behandles av et plugin. Det vil si at innholdet i hele listen skal taes hånd om av et plugin. Eksempel på dette kan være en spilleliste med filmer som skal vises i en og samme spiller.

Som nevnt under gjennomgangen av entiteter tidligere i kapittelet, har `type` attributten til et felt i en entitet blitt utvidet til også å kunne ha verdien plugin. En spesifiserer da at informasjonen som mappes til dette feltet, skal rendres av et plugin. Dette dekker det første tilfellet.

Det andre tilfellet, hvor man skal ha mulighet til å gi en liste med informasjon til et plugin, blir håndtert litt annerledes. Når man definerer at et felt skal inneholde en liste, har man allerede satt `type` attributtet lik `entity-list`, så har man ikke mulighet å benytte dette `type` atrtibuttet til å si at listen skal behandles av et plugin. Pluginmotoren sjekker da i stedet om attributtet `pluginconfig` er tildelt en verdi hver gang et felt med type lik `entity-list` blir prossessert.

Ved bare å sette `type` lik `plugin` har man bare definert at informasjonen som mappes til feltet skal håndteres av et plugin. Men man har ikke sagt noe om hvilket plugin som skal benyttes. Det er her attributtet `pluginconfig` kommer inn i bildet. Attributtet sier riktignok ikke hvilket eksakt plugin som skal benyttes, men inneholder en logisk verdi som mappes til et bestemt plugin i en konfigurasjonsfil. Hvert mønster har en slik konfigurasjonsfil der en i tillegg til å mappe felt i mønsteret til bestemte plugins, kan definere parametre som pluginet benytter seg av. En oppnår da en generisk struktur med stor fleksibilitet.

Et scenario kan være at man har flere felt i mønsteret som benytter seg av et plugin med det logiske navnet `videospiller`. Så utvikles det et nytt plugin som er bedre egnet til å ta hånd om multimediaobjektene. I stedet for at en da må endre på alle feltene i mønsteret hvor `pluginconfig` er satt til `videospiller,` kan en bare endre konfigurasjonsfilen og mappe det logiske navnet til det nye pluginet.

En annen fordel med en slik konfigurasjonsfil er at en kan definere en parameterliste for hvert element i denne filen. Hvert element blir identifisert med det logiske navnet som er definert i mønsteret. Det vil si at hvert plugin kan ha ulike parameterverdier utfra hvor de blir benyttet i mønsteret. Med andre ord kan et bestemt plugin mappes til mange logiske navn med egne parameterlister. En viktig forutsetning er at pluginet implementeres slik at det tar hensyn til samlingen med parametre som sendes til pluginet.

5.6 Multimedia eksempel

I dette avsnittet vises et eksempel som belyser sammenhengen mellom mønsterdefinisjonen, pluginkonfigurasjonsfilen, innholdsfilen og hva som blir resultatet i presentasjonen. Utgangspunktet er at det skal vises et bildegalleri på en side, og et plugin har som oppgave å utføre denne oppgaven.

Mønsterdefinisjon med plugin

Kodesnutten i Eksempel 5-14 viser definisjonen av de nødvendige entitetene i mønsteret. Entiteten `imageCollectionEntity` inneholder blant annet en liste med `imagelistEntity` entiteter. Dette er uttrykt ved at feltet `imageEntities` har `type` attributtet lik `entity-list` og attibuttet `ref-id` lik `imageListEntity`. I tillegg er attributtet `pluginConfig` satt til verdien `photogallery`. Dette betyr at listen av

`imageListEntity` entiteter skal prossesseres av et plugin som er mappet til den logiske verdien `photogallery`.

```
<entity id="imageListEntity">
     <fields>
         <field type="string">type</field>
         <field type="string" required="true">title</field>
         <field type="file">image</field>
     </fields>
</entity>

<entity id="imageCollectionEntity">
     <fields>
        <field type="string" required="true">title</field>
        <field type="string" required="true">description</field>
        <field type="entity-list" ref-id="imageListEntity"
               pluginConfig="photogallery" >imageEntities
        </field>
     </fields>
</entity>
```

Eksempel 5-14: Definisjonen av imageCollectionEntity i mønsteret

Definisjon av parametre til plugin-et

Mappingen mellom det logiske navnet `photogallery`, som var definert i mønsteret, og det eksakte pluginet skjer i `pluginPattern` filen. Hvert mønster har, som nevnt tidligere, en egen fil ved navn `pluginPattern` der denne mappingen foregår. Som Eksempel 5-15 viser, blir det logiske navnet `photogallery` mappet til et plugin ved navn `ThickboxPictureGalleryJQueryPlugin`.

Hvert plugin-config element kan inneholde ulike parametre. Parametrene inneholder verdier som skreddersyr pluginet til å rendre resultatet på en bestemt måte. Parametrene gjenspeiler ofte egenskaper som en utvikler vil ha mulighet til å endre på når pluginet skal benyttes i ulike kontekster. Parametrene i eksempelet dikterer høyde og bredde til forhåndsvisningen av bildene, antall bilder som skal forhåndsvises per rad, tittel til bildegalleriet og om det skal være en ramme rundt bildegalleriet.

```
<plugin-config id="photogallery"
               plugin-name="ThickboxPictureGalleryJQueryPlugin">
      <param name="thumbnailHeight">100</param>
      <param name="thumbnailWidth">80</param>
      <param name="numberOfPicturesInRow">2</param>
      <param name="title">Liverpool FC Gallery</param>
      <param name="tableBorder">0</param>
</plugin-config>
```

Eksempel 5-15: Hvordan man binder det logiske plugin-navnet fra mønsteret opp mot et eksakt plugin.

Innholdsfil som rendres av plugin

For å komplementere eksempelet vises også et utdrag fra XML dokumentet, hvor innholdet er strukturert (se Eksempel 5-16). Dokumentet er bygget opp slik som mønsteret dikterte strukturen. Elementet `imageCollectionEntity` inneholder en liste av `imageListEntity` elementer. I hvert slikt element ligger tittelen til bildet og filen som

inneholder bildet. Listen blir koblet av innholdsdokumentet og sendt videre til pluginet for videre prossessering.

```
<imageCollectionEntity type="rootEntity"
      entityPath="imageCollectionEntityInstance_imageCollectionEntity"
      createdBy="brsseb" updatedBy="brsseb" createdOn="25/04/2008 16:24"
      updatedOn="25/04/2008 16:24">
      <title type="string" required="true">Image Gallery</title>
      <description type="string" required="true">
            Images form this liverpool site
      </description>
      <imageEntities type="entity-list"
           entityPath="imageCollectionEntityInstance_imageCollectionEntity_
                       imageEntities_imageListEntity">
            <imageListEntity type="subEntity" entityInstanceId="1234"
                  entityPath="imageCollectionEntityInstance_
                  imageCollectionEntity_imageEntities_imageListEntity_1234"
                  createdBy="Ingvaldsen" createdOn="28/04/2008 12:00"
                  updatedBy="Ingvaldsen" updatedOn="28/04/2008 12:00">

                  <title type="string">Xabi Alonso</title>

                  <image type="file">alonso.jpg</image>

            </imageListEntity>

            ...
      </imageEntities>
</imageCollectionEntity>
```

Eksempel 5-16: Et innholdsdokument som svarer til ImageCollectionEntity

Resultat av rendringen

Resultatet av rendringen som skjer i pluginet, vises i Figur 5-5. Verdiene til parametrene i pluginPattern-filen kommer tydelig frem her gjennom overskriften, antall bilder per rad, bildestørrelsen og at det ikke er en ramme rundt galleriet. Det som ikke kommer frem av bildet, er hva som skjer med et bilde når brukeren trykker på det. Pluginet sammarbeider med et CSS-stilark og et javascript som viser en større versjon av bildet i forgrunnen på siden.

Figur 5-5: Resultatet som plugin-et genererer i nettleseren

6 Presentasjon av Presentation Viewer PV

I dette kapittelet følger en detaljert beskrivelse av PresentationViewer PV. I denne delapplikasjonen foregår rendring av presentasjoner, og det er her pluginmodulen benyttes for å rendre HTML kode for multimediaobjekter. Første delen av kapittelet gir en kort innføring i hvordan PV er bygget opp, samt en redgjørelse av hva som foregår i de viktigste klassene. Andre delen av kapittelet gjennomgår rendringsprosessen som foregår i PV.Ved hjelp av sekvensdiagram sammenlignes rendingsprosessen til en forespørsel der pluginmodulen ikke er i bruk, med rendringsprosessen til en forespørsel hvor plugin-modulen hjelper til å med å generere HTML kode for et multimediaobjekt.

6.1 Hovedoppgavene til Presentation Viewer

Presentation Viewer PV er subsystemet som har ansvaret for å rendre sider i presentasjoner.Det vil si å generere HTML-kode for alle presentasjoner som er opprettet i DPG-en. I PV utføres følgende hovedoppgaver:

- DPG 2.0 kan håndtere mange forskjellige presentasjoner. PV trenger dermed informasjon av brukeren som forteller hvilken presentasjon som skal vises. Brukeren kan også oppgi en bestemt side i presentasjonen som skal vises og hvilket view som skal vises på siden. Hvis ikke dette er spesifisert, vil standardsiden og standard viewet vises. Et view kan inneholde en liste av elementer, PV-en har funksjonalitet for å vise kun ett element i en en slik liste. Dette defineres med en unik sti til elementet. PV får tak i all denne informasjonen ved hjelp av parametre som hentes fra forespørselobjektet.
- For hvert `View` som skal rendres identifiseres og hentes det tilhørende XML dokumentet. Entitetene som definerer strukturen på innholdet blir så analysert for å sjekke om noen av elementene skal rendres av plugins. Hvis så er tilfelle, oppdateres XML dokumentet, innholdet i elementet erstattes med ferdig rendret HTML fra et plugin. Dersom ingen elementer skal rendres av plugins, forblir XML dokumentet uforandret. Dokumentet sendes i neste omgang videre til transformasjon ved bruk av XSLT stilark.
- Hver side i presentasjonen har en egen viewmeny. Denne menyen består av linker til views som ikke er definert til å være `alwaysVisible=true`. PV autogenererer et XML document basert på denne informasjonen som igjen sendes til transformasjon ved bruk av XSLT stilark.
- Alle presentasjoner har en hovedmeny. Denne består av linker til hver `page` som er definert i mønsteret. PV autogenererer et XML document basert på denne informasjonen som igjen sendes til transformasjon ved bruk av XSLT stilark.
- Hver presentasjon har sin egen navigasjonsmeny. Dette er i realiteten en meny med link for utlogging, link til `Lobby`-en hvor en kan bytte presentasjon og link til PCE-en. PV autogenererer også her et XML dokument som inneholder denne informasjonen og som sendes videre til transformasjon ved bruk av XSLT stilark.
- Hver `page` i mønsteret har en Velocity sidemal. PV-en gir templatet ansvaret for å plassere `view`-ene i passende HTML containere. Hvert mønster har i tillegg en

Velocity hovedsidemal som er strukturert som en vanlig HTML side. PV-en gir dette templatet ansvaret for å sette resultatet fra de enkelte sidenes sidemal sammen med resultatet fra transformasjonene av hovedmenyen, viewmenyen og navigasjonsmenyen. I tillegg blir det satt inn link til CSS stilark, tittel til presentasjonen og beskrivelse av presentasjonen. Resultatet blir ferdig rendret HTML for en side som ble forespurt i forespørselen.

6.2 Overordnet arkitektur

I Figur 6-1 vises et klassediagram som presenterer klassene og pakkene som benyttes for å bygge opp funksjonaliteten i PV. Pakkene utenfor `pv` pakken inneholder kun klasser som er relevant for PV. Diagrammet er også forenklet med at klassene som ligger utenfor `pv` pakken er kun referert til på pakkenivå. Her følger en redgjørelse av hvilke klasser som er ansvarlig for å utføre hovedoppgavene i PV.

I `controllers`-pakken er det en klasse som heter `PresentationController`. Applikasjonen er, som nevnt tidligere, bygget ved hjelp av Spring rammeverk, se [33]. I Spring definerer man kontrollere for å håndtere forespørsler og for å bygge opp en respons. Denne klassen er da ansvarlig for å hente ut nødvendig parameterinformasjon fra forespørselobjektet og sende dette videre til servicelaget for å generere korrekt innhold til responsen.

I `service` pakken inne i `pv` pakken befinner det seg en klasse som heter `PresentationViewerService`. Det er denne klassen som mottar parameterinformasjon fra `PresentationController` klassen. Utfra verdiene til parametrene, har denne klassen ansvaret for å delegere videre de ulike oppgavene som skal utføres for å oppnå en fullstendig og korrekt respons.

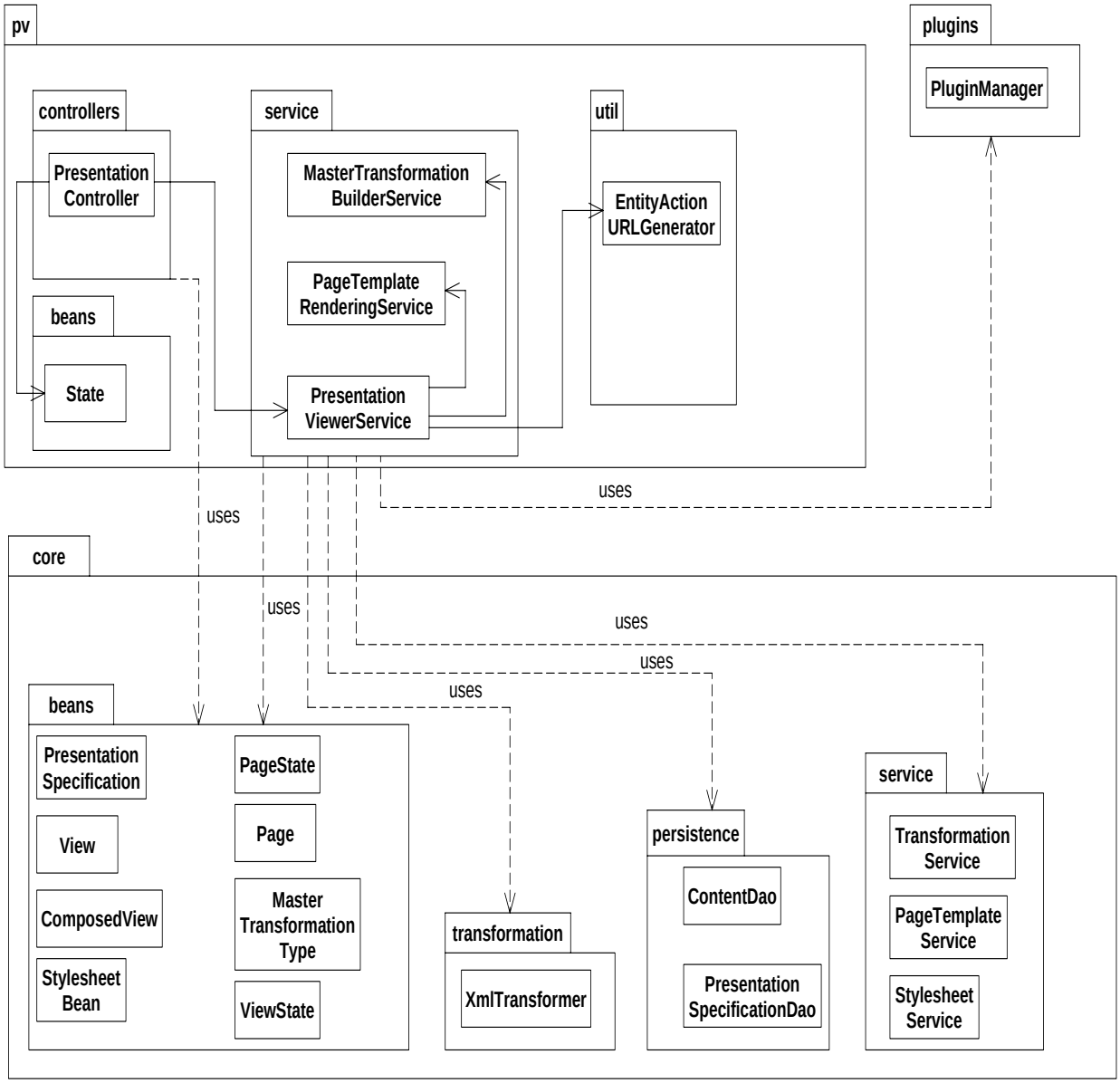
Klassen `ContentDao` i `persistence` pakken har ansvaret for å hente XML dokumentet til hvert enkelt view. Den får nødvendig parameterinformasjon fra klassen `pv.service.PresentationViewerService` til å utføre oppgaven. XML dokumentene blir så sendt til klassen `plugins.PluginManager`, hvor eventuell rendring ved hjelp av plugins foregår.

Klassen `core.service.TransformationService` slår opp korrekt stilark til hvert enkelt view basert på parameterinformasjon fra `pv.service.PresentationViewerService`. Transformeringen av XML dokumentet skjer i klassen `core.transformation.XmlTransformer.` Metoden som utfører denne operasjonen får inn XML dokumentet, stilarket og eventuelle parametre som skal benyttes i stilarket fra `pv.service.PresentationViewerService`.

Når innholdet i views-ene er ferdig rendret, er det de ulike menyene som skal rendres. Som nevnt tidligere, er dette hovedmeny, viewmeny og navigasjonsmeny. Klassen `pv.service.MasterTransformationBuilderService` har ansvaret for å generere korrekte XML dokumenter til menyene. På sammen måten som for views brukes klassen `core.service.TransformationService` til å finne frem stilarket som skal benyttes i transformeringen. Klassen `core.transformation.XmlTransformer` tar seg av selve transformasjonen.

Klassen `core.service.PageTemplateService` er ansvalig for å hente frem korrekt velocity-mal for organisering av views-ene samt hovedsidemalen. Klassen `pv.service.PageTemplateRenderingService` utfører rendringen som organiserer views-ene og rendringen som setter alle delene sammen til en fullstendig HTML side. Resultatet av denne operasjonen retuneres til klassen `pv.controllers.PresentationController` som igjen hekter det på responsobjektet.

Klassen `pv.util.EntityActionURLGenerator` er ansvarlig for å generere edit og delete linker til hvert enkelt view. I tillegg er klassen ansvarlig for å generere edit, delete og add linker til hvert element i en liste.



Figur 6-1: Klassediagram til delsystemet Presentation Viewer PV

6.3 Forskjellene i rendringsprosessen med og uten plugins aktivitet

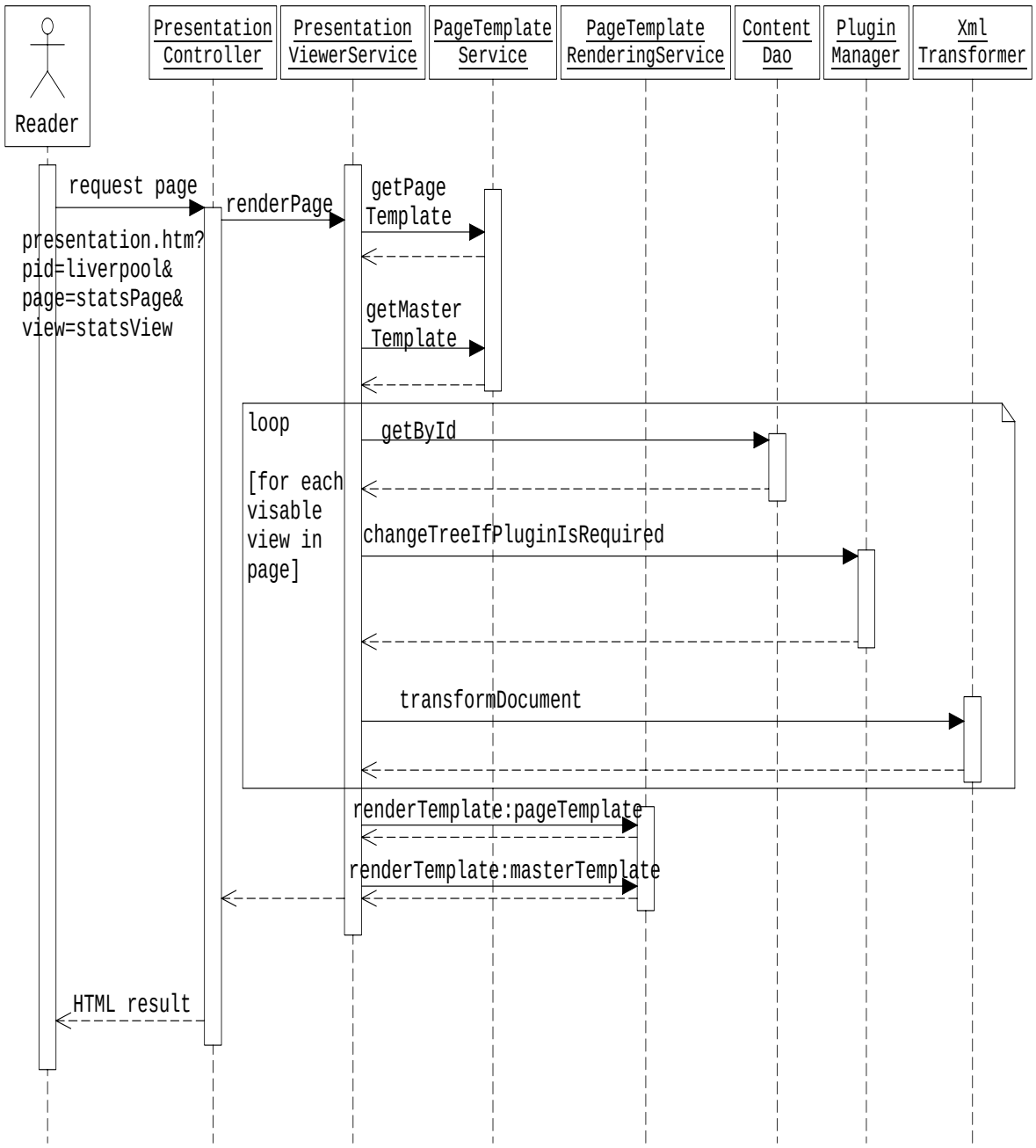
I dette delkapittelet klargjøres hva som skjer når det kommer en forespørsel til PV. Det vil bli vist to ulike eksempler på en forespørsel som kan inntreffe. Dette gjøres for å vise forskjellene i rendringsprosessen når en forespørsel spør om et view som inneholder ren tekst og når en forespørsel spør om et view som inneholder ett eller flere multimediaobjekter.

Rendringsprosessen uten pluginaktivitet

Ved hjelp av sekvensdiagrammet i Figur 6-2 gis her en innføring til syklusen til en forespørsel, hvor ingen elementer krever at innholdet prosesseres av et plugin. I dette tilfellet er det en bruker som har rollen `Reader` som sender en forespørsel om å vise `statsView`-et på siden `statsPage` som ligger i presentasjonen ved navn `liverpool`. Som vist i figuren sendes denne informasjonen som get-parametre. `PresentationController` henter disse parametrene fra forespørselobjektet og sender dem videre i metodekallet `renderPage` til klassen `PresentationViewerService`. Som vist i figuren, delegerer denne klassen arbeidet som skal utføres videre til spesialiserte klasser som utfører de forskjellige enhetene med arbeid. Det første som skjer er at klassen `PageTemplateService` identifiserer og retunerer en velocity mal for sidemal og for hovedsidemal.

Neste steg er å rendre de ulike view-ene som page-en inneholder. Her er det forespurt om et enkelt view, men det kan likevel være flere views som må rendres. Dette bestemmes, som nevnt tidligere, av mønsterspesifikasjonen, der man kan spesifisere at et view alltid skal være synlig når den bestemte siden rendres. For å tydeliggjøre poenget med eksempelet, vil kun view-et som er forespurt bli beskrevet. `PresentationViewerService` sender en forespørsel til `ContentDao` om å identifisere og retunere XML dokumentet som gjenspeiler innholdet til `statsView`-et. XML dokumentet som retuneres sendes så til PluginManager som analyserer det og retunerer det uforandret tilbake til `PresentationViewerService`. View-et er nå klart til å transformeres fra XML til HTML. Denne transformeringsoppgaven delegeres til `XmlTransformer` klassen som utfører transformeringen ved hjelp av XSLT-stilark.

`PresentationViewerService` klassen har nå bare to oppgaver igjen å delegere videre. Det er å gruppere viewene i forhold til hverandre samt og bygge opp hele HTML-siden. Dette gjøres ved to kall til `renderTemplate` metoden til klassen `PageTemplateRenderingService`. Det ene kallet sender sidemalen, som er definert i mønsteret i page elementet statsPage, som et av parametrene til metoden. Det andre sender hovedmalen masterTemplate, som er felles for alle sidene i mønsteret, som parameter til metoden. `PresentationViewerService` har nå rendret ferdig siden som er forespurt og kan retunere innholdet til `PresentationController`, som igjen kan hekte innholdet på responsobjektet som retuneres til brukeren.



Figur 6-2: Sekvensdiagram som viser hva som skjer i PV når det ikke er plugin aktivitet

Rendringsprosessen med pluginaktivitet

Multimedia eksempelet i delkapittel 5.5 viste samspillet mellom mønsterdefinisjonen, pluginkonfigurasjonsfilen, innholdsfilen og hva som blir resultatet av presentasjonen for brukeren. Ved hjelp av sekvensdiagrammet i Figur 6-3, følger her en beskrivelse av rendingsprosessen til PV, sett i et multimediaperspektiv. Utgangspunktet er det samme som i

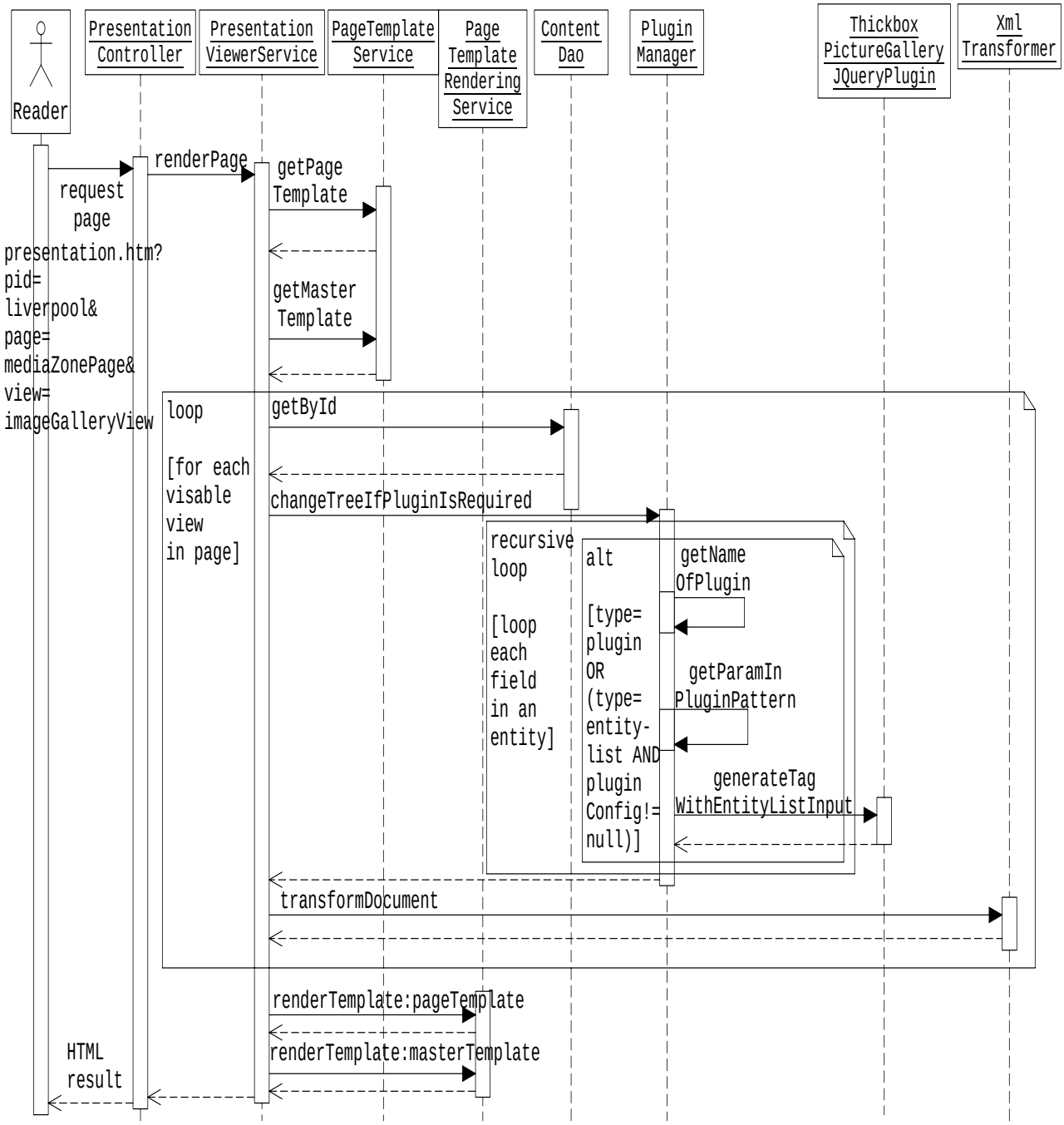
delkapittel 5.5, det skal vises et bildegalleri på en side, og et plugin har som oppgave å utføre denne operasjonen.

Det er viktig å merke seg at det er bare når views inneholder multimeidaobjekter at rendringsprosessen endres. Finnes det andre views på en side som ikke inneholder multimediaobjekter, blir disse rendret på samme måte som i Figur 6-2.

Forespørselen i Figur 6-3 viser at reader-en vil at view-et `imageGalleryView` skal vises frem på siden `mediaZonePage` som befinner seg i presentasjonen `liverpool`. Løkken, i klassen `PresentationViewerService` , prosesserer alle synlige views på siden og henter ut XML dokumentene til view-ene ved kall til metoden med navn `getByID`. Strukturen til hvert view, som er spesifisert i mønsteret, blir så analysert av `PluginManager` for å sjekke om entitet-en i view-et inneholder felt som skal rendres av et plugin. Denne sjekken foregår i den rekursive løkken som startes i metoden `changeTreeIfPluginIsRequired.` Når løkken starter prosessering av view-et `imageGalleryView,` vil analysen i `pluginmanager` avdekke at view-et inneholder entitet-en `ImageCollectionEntity` som igjen har et felt `imageEntities` hvor `type` er lik `entity-list` og `pluginConfig` er satt til verdien `photogallery`. Feltet som inneholder listen skal da rendres av et plugin, men før det må `PluginManager` gjøre endel preprosessering.

Som sekvensdiagrammet viser, kalles den lokale metoden `getNameOfPlugin`. Denne metoden åpner `pluginPattern.xml` og henter det fulle navnet på plugin-et som skal benyttes, basert på det logiske navnet som er definert i mønsteret gjennom attributten `pluginconfig.` Det neste som skjer er at `PluginManager` henter ut alle parametre som plugin-et skal ta hensyn til. Dette hentes også fra `pluginPattern.xml` filen. `PluginManager` har da tilstrekkelig informasjon til å kalle korrekt plugin og sende med blant annet de fastsatte parametrene og innholdsdokumentet.

I dette eksempelet, er det `ThickboxPictureGalleryJQueryPlugin,` som får oppgaven med å oppdatere elementet `imageEntities` i innholdsdokumentet med ferdig rendret HTML kode for et bildegalleri. `PluginManager` sender så det oppdaterte XML dokumentet tilbake til `PresentationViewerService`. Dokumentet skal så sendes til transformasjon ved kallet til metoden `transformDocument` som befinner seg i `XmlTransformer` klassen. Ettersom pluginet `ThickboxPictureGalleryJQueryPlugin` har tatt seg av rendringen av elementet `imageEntities,` er det viktig at det spesifiseres at dette elementet ikke skal tranformeres videre på noen måte, men i stedet benyttes uforandret. Resten av rendringsprosessen er lik den som er vist i Figur 6-2.



Figur 6-3: Sekvensdiagram som viser hva som skjer i PV når det er plugin aktivitet.

7 Arkitektur til pluginmodul

Dette kapittelet gir en beskrivelse av arkitekturen til plugin-modulen. Valget av plugin arkitektur ble, som nevnt i delkapittel 4.2, foretatt på bakgrunn av at man ønsket en fleksibel arkitektur. Det er ønskelig at man kan legge til støtte for nye multimediatyper, samt oppdatere eksisterende kode for multimediaobjekter, uten at man trenger å rekompilere og laste opp hovedapplikasjonen på nytt. Plugin-arkitektur løser dette ved å tilby en sentralisert runtime konfigurasjon av plugins.

7.1 Mønster for plugins

Martin Fowler beskriver i boken Patterns of enterprise application architecture [21] et mønster for å bygge en plugin arkitektur. Mønsteret er brukt som inspirasjon for plugin-modulen i DPG 2.0, men det er gjort visse tilpasninger som følge av bruksområdet til applikasjonen.

Mønsteret innebærer at man først oppretter såkalte Separated Interfaces, hvor en definerer oppførsel som vil ha forskjellig implementasjon basert på kjøremiljøet. Separated Interfaces er et konsept der man definerer et grensesnitt i en bestemt pakke, men implementerer det i en annen pakke. På denne måten kan en klient, som trenger avhengigheten til grensesnittet, være totalt uvitende til implementasjonen. Det er altså disse grensesnittene plugin-ene implementerer.

Videre brukes Factory-mønsteret, med noen få endringer, til instansiering av plugins. Dette mønsteret gjør det mulig å instansiere konkrete objekter mens en bare avhenger av deres grensesnitt. Men med en factory metode bestemmes hvilke objekter som skal retuneres ved hjelp av testing av lokale miljøvariabler. Det vil si at linkingen mellom grensesnitt og klassen som implementerer grensesnittet skjer i factory metoden. Dette fører til at man må utvide factory-metoden med en ny test hvis det kommer en ny implementasjon av grensesnittet.

En plugin factory krever at link instruksjonene spesifiseres på ett bestemt punkt, slik at det er enkelt og oversiktlig å håndtere konfigurasjonen. Samtidig er det viktig at linkingen til implementasjonen av grensesnittene skjer dynamisk under kjøring, i stedet for under kompileringen. Da slipper man å bygge applikasjonen på nytt hver gang man endrer konfigurasjonen. Link instruksjonene kan defineres i en vanlig tekstfil. Da vil factory-en lete i filen etter en forespurt implementasjon av et bestemt grensesnitt og returnere denne.

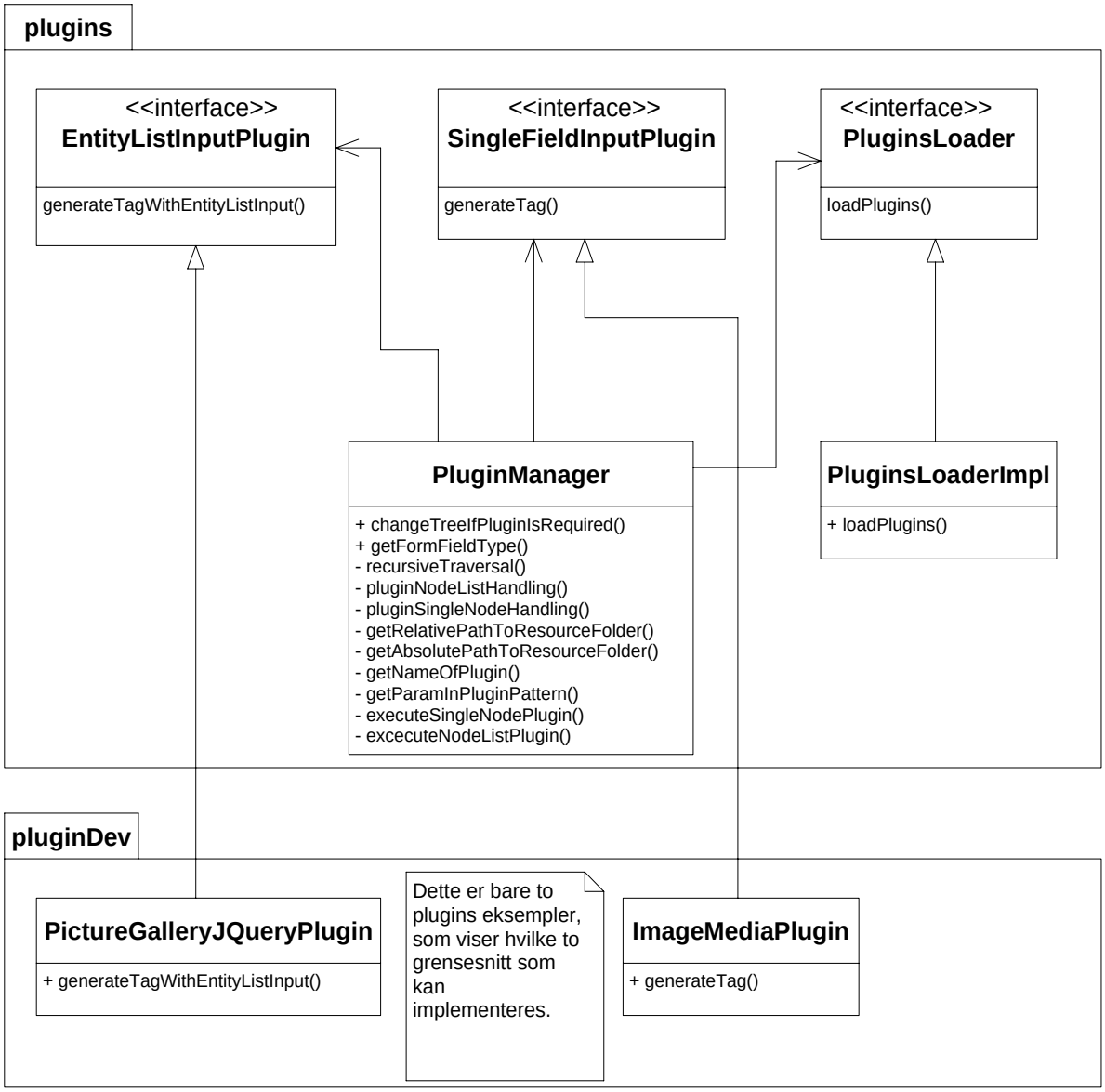
Plugin arkitektur fungerer best i språk som støtter reflection. Dette er fordi factory-en da kan opprette objekter uten avhengigheter dem i mellom ved kompilering. Når en bruker reflection, er det viktig at konfigurasjonsfilen inneholder mapping mellom grensesnittnavn og klassenavn til klassen som implementerer grensesnittene. Factory-en trenger da ikke å bli endret når man legger til ny implementasjon av grensesnittene.

7.2 Pluginsmodul i DPG 2.0

I dette delkapittelet følger en presentasjon av pluginsmodulen i DPG 2.0 og en beskrivelse av hvordan teorien i forrige delkapittel er satt ut i prakisis.

Oppbyggingen av plugins-modulen

Klassediagrammet i Figur 7-1 viser hvordan modulen er bygget opp.



Figur 7-1: Klassediagram for plugin-modulen.

I pakken plugins ligger det to grensesnitt ved navn `EntityListInputPlugin` og `SingleFieldInputPlugin`. Grensesnittene bygger på konseptet om Separated Interfaces. `PluginManager`, som har avhengigheter til grensesnittene, trenger ikke å vite noe om implementasjonen av dem. Det er disse grensesnittene som kan implementeres av plugins. Applikasjonen da har to steder en kan koble seg på med et programvaretillegg eller plugin. Det er hva man ønsker å utvikle som bestemmer hvilket grensesnitt man skal implementere.

Den store forskjellen på grensesnittene er at man i `SingleFieldInputPlugin` kun får tilgang til verdien i et bestemt element fra et XML innholdstre. I `EntityListInputPlugin` får man tilgang til en liste av like elementer fra et XML innholdstre. Med andre ord implementerer man f.eks. `SingleFieldInputPlugin` grensesnittet hvis man vil utvikle et plugin for å vise et enkelt bilde, mens man må implementere `EntityListInputPlugin` hvis det er ønskelig å utvikle et plugin som genererer et bildegalleri utfra en liste med bilder.

Grensesnittet `PluginsLoader` definerer en plugins-factory. Ved å bruke grensesnitt til factory-en kan man, ved hjelp av objektinstansieringen til rammeverket Spring [33] ,enkelt bytte ut implementasjonen av factory-en ved å endre i konfigurasjonsfilen til Spring. Dette skaper en fleksibel arkitektur ved at man fjerner avhengigheten mellom implementasjonen av plugin factory-en og `PluginManager`.

`PluginsLoaderImpl` implementerer grensesnittet `PluginsLoader`. Dette er da implementasjonen av factory-en til pluginmodulen. Link-instruksjonene, som angir linken mellom grensesnittnavn og klassenavn, defineres i en bestemt XML-fil. Factory-en leser så denne konfigurasjonsfilen og oppretter objekter ved bruk av reflection, basert på konfigurasjonen.

Konfigurasjon av plugins

Eksempel 7-1 viser et utdrag fra en XML konfigurasjonsfil. Link instruksjonene blir definert i et `plugin` element. Et slikt element definerer det fulle klassenavnet til plugin-et, hvilket grensesnitt klassen implementerer, en identifikator til plugin-et og hvilket skjemaelement som skal benyttes i editoren. Linken mellom klassenavn og grensesnitt benyttes under instansieringen av pluginsene slik at pluginklassen blir instansiert med det korrekte grensesnittet. Identifikatoren er nødvendig siden det er mange plugins som kan implementere samme grensesnitt. Denne identifikatoren blir benyttet som nøkkel når hovedapplikasjonen skal hente frem et konkret plugin fra samlingen av plugins. Det er derfor viktig at identifikatoren er unik.

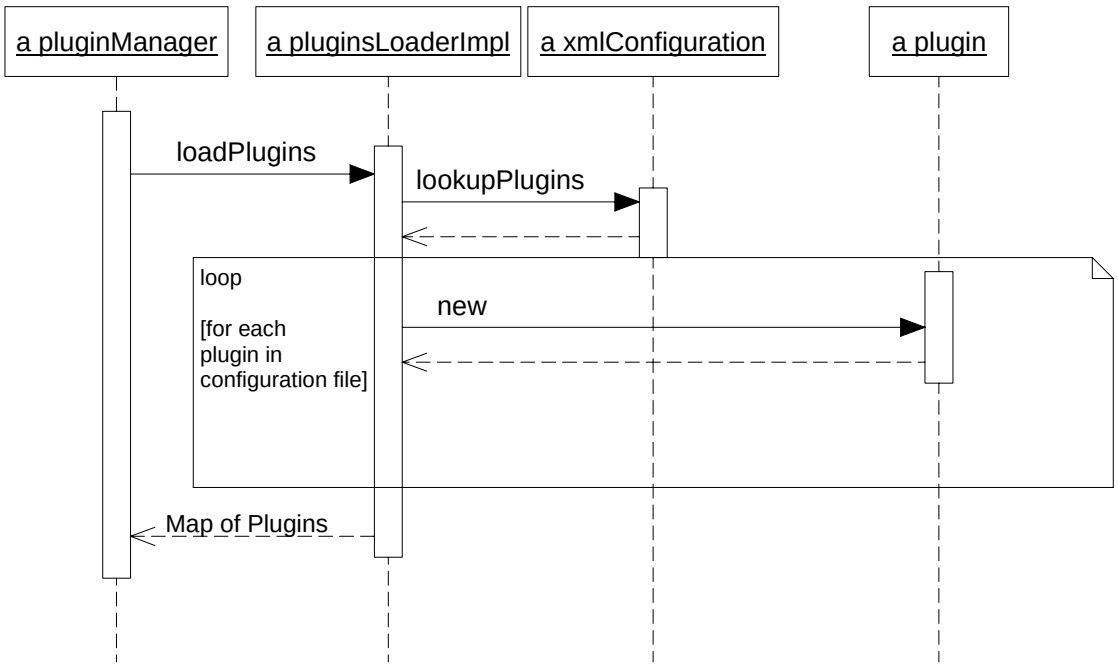
I editoren skal det blant annet være mulig å editere og legge til elementer som skal rendres av et plugin. Innholdet som ligger i disse elementene kan være filer eller strenger. Eksempel på en fil kan være en bildefil som skal rendres av et plugin. En streng kan være en id-streng til en film som allerede er tilgjengelig på Internet (f.eks. på nettstedet www.youtube.com), men skal inkluderes i en presentasjon i DPG-en ved hjelp av et plugin. Et plugin kan ikke endre på typen inndata fra mønster til mønster, av den grunn kan man spesifisere hvilket skjemaelement som skal vises i editoren i konfigurasjonsfilen til plugin-et. Som Eksempel 7-1 viser, definerer elementet `formfield-type` skjemaelementet som skal benyttes for et bestemt plugin.

```
<?xml version="1.0" encoding="UTF-8"?>
<application>
  <app-name>DPG2.0</app-name>
  <plugins>
     <plugin>
       <!--fully qualified classname-->
       <class-name>
          entityListPlugin/PictureGalleryJQueryPlugin.class
       </class-name>
       <!—Implemented interface-->
       <interface>EntityListInputPlugin</interface>
       <!—Id of the plugin -->
       <plugin-name>PictureGalleryJQueryPlugin</plugin-name>
       <!—Used by the editor to specify what type of form element to use
          for this plugin -->
       <formfield-type>file</formfield-type>
     </plugin>

     <plugin>
       <class-name>image/ImageMediaPlugin.class</class-name>
       <interface>SingleFieldInputPlugin</interface>
       <plugin-name>ImageMediaPlugin</plugin-name>
       <formfield-type>file</formfield-type>
     </plugin>

      ...

   </plugins>

</application>
```

Eksempel 7-1: Utdrag fra plugin-konfigurasjonsfil for DPG2.0

Instansiering av plugins

Figur 7-2 viser et sekvensdiagram som modellerer hvordan objektene sammarbeider om å utføre oppgaven med å instansiere plugins. Instansen av `PluginManager` sender en forespørsel til `pluginsLoaderImpl` objektet om å laste inn alle installerte plugins. Objektet `pluginsLoaderImpl` benytter seg av XML rammeverket til Apache Commons [38] for å lese konfigurasjonsfilen og sender da en forespørsel om å lese XML filen. En løkke startes så for å instanisere alle plugin-ene som er definert i konfigurasjonsfilen. Alle plugin-ene blir så samlet i en datastruktur og retunert til instansen av `PluginManager`.



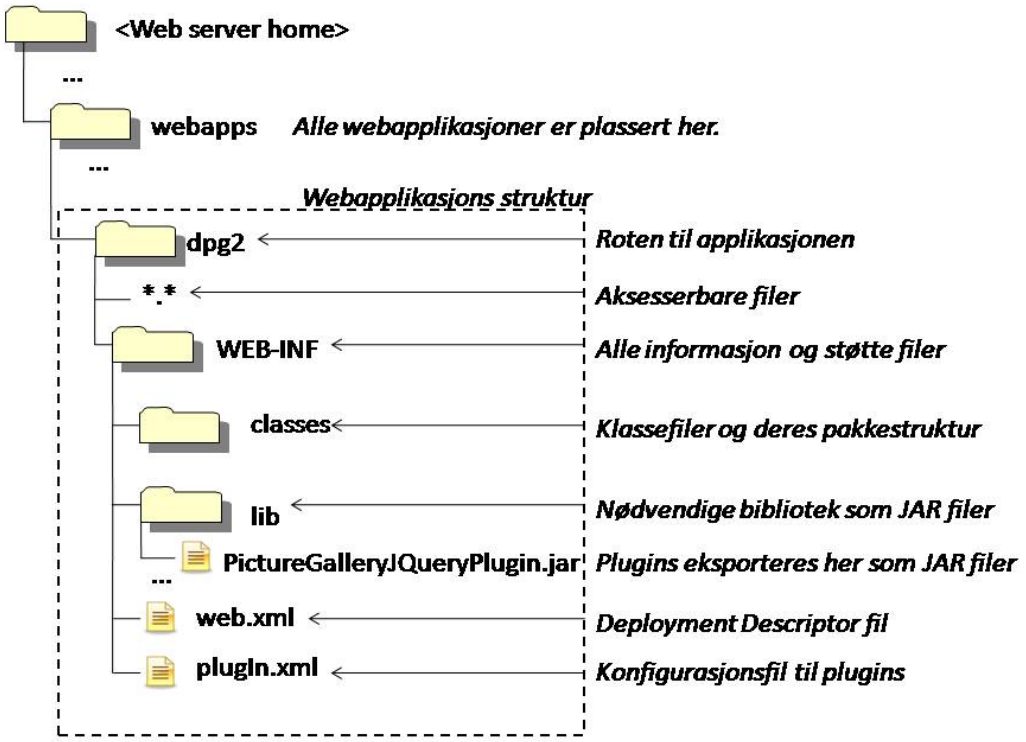
Figur 7-2: Sekvensdiagram som viser stegene i instaniseringen av plugins

Installasjon av plugins

Som nevt tidligere i delkapittelet, er grensesnittene som plugin-klasser kan implementere bygget opp som Separated Interfaces. PluginManager er avhengig av grensesnittene, men trenger ikke å bry seg om implementasjonen av dem. Dette er en veldig viktig faktor i utbyggingen av en pluginmodul. Årsaken er at man ikke vil jobbe med kildekoden til hovedapplikasjonen når man skal laget et plugin. For det ville ført til at man måtte rekompilere og laste opp applikasjonen på nytt hver gang en plugin ble utviklet. Løsningen som er valgt for pluginmodulen til DPG 2.0 er at grensesnittene, som definerer tilkoblingpunkt, legges inn i en Java ARchieve (JAR) [39] fil. Denne JAR filen vil da fungere som et bibliotek som kan benyttes av et hvilket som helst uavhengig Java prosjekt. En er da totalt uavhengig av kildekoden til hovedapplikasjonen i utviklingen av nye plugins.

Et ferdig utviklet plugin må installeres eller kobles til hovedapplikasjonen. Det er da to operasjoner som må gjennomføres. Det første er å eksportere pluginet til en lokasjon som ligger i `classpath`-en [40] til hovedapplikasjonen (DPG 2.0). Dette er nødvendig for at `factory`-en skal kunne instansiere plugins-ene. Løsningen som er valgt for denne pluginsmodulen, er at plugins-ene eksporteres som JAR filer til DPG 2.0 sin biblioteksmappe ved navn `lib` (se Figur 7-3). Alle klasser i `classes` mappen til webapplikasjonen, også `Factory`-en `PluginsLoader`, kan benytte seg av bibliotekene som ligger i `lib` mappen. Dersom et plugin benytter seg av et eksternt bibliotek, må også dette biblioteket eksporteres til DPG 2.0 sin `lib` mappe. Klassediagrammet i Figur 7-1 illustrerer at plugins-ene ligger i en annen pakke. Dette er korrekt, men pakken trenger ikke å ligge i same pakkestruktur som klassene i hovedapplikasjonen.

Den andre operasjonen som må gjennomføres, er å definere link instruksjonene mellom klassenavn og grensesnitt, samt gi plugin-et en identifikator. Dette gjøres, som nevnt tidligere, i delkapittelet i en XML fil. Denne XML filen har navnet `plugIn.xml` og befinner seg i `WEB-INF` mappen til webapplikasjonen DPG 2.0 (Se Figur 7-3). Når de to operasjonene er utført, er plugin-et koblet til, men man må starte serveren på nytt for at plugin-et skal aktiveres. Grunnen til dette er at alle plugins-ene instansieres i oppstarten til webapplikasjonen.



Figur 7-3 Hvor pluginkomponenter ligger i strukturen til webapplikasjonen DPG2.0.

Flere plugins kan, som nevnt tidligere, implementere samme grensesnitt. Under instansieringen av plugins legges plugins-ene i en datastruktur som retuneres til klassen `PluginManager` når prosessen er ferdig. `PluginManager` har som oppgave å håndtere bruken av plugins. Men også utvelgelsen av hvilket plugin som skal benyttes i enhver situasjon, skjer dynamisk. I delkapittel 5.5 nevnes en konfigurasjonsfil ved navn `pluginPattern.xml`. Hvert mønster i applikasjonen har en slik konfigurasjonsfil, og det er her linkingen mellom den logiske verdien i attributtet `pluginConfig` i mønsterdefinisjonen og identifikatoren til hvert enkelt plugin foregår. `PluginManager` slår opp i denne XML filen hver gang den blir forespurt om å benytte et plugin og kaller metoden som er spesifisert av grensesnittet i den korrekte pluginklassen.

PluginModulen utfører både operasjonen med å innstallere plugins og håndteringen av hvilket plugin som skal benyttes til en hver tid dynamisk. Hva som utføres i pluginsmodulen ved hver forespørsel, dikteres av konfigurasjonen i de to konfigurasjonsfilene plugIn.xml og

pluginPattern.xml. Dette skaper stor fleksibiltet i modulen og gjør operasjoner som for eksempel oppdatering av hvilket plugin som skal benyttes til rendring av et spesiellt felt, endring av parametre som plugins benytter seg av og installasjon av nye plugins til en enkel oppgave .

8 Plugins i DPG 2.0

I dette kapittelet følger en presentasjon av grensesnittene plugins-ene kan implementere, hvordan man bygger opp et plugin og tilslutt en presentasjon av de plugins-ene som er implementert og installert i DPG 2.0.

8.1 Grensesnittene som plugins kan implementere

Klassen som representerer factory-en, `PluginsLoaderImpl`, kan instanisere klasser som enten implementerer grensesnittet `SingleFieldInputPlugin` eller grensesnittet `EntityListInputPlugin`. De to grensesnittene er tilkoblingspunkter hvor en kan koble på en programvareutvidelse, også kalt plugin. Forskjellen på grensesnittene er hva de tilbyr utvikleren å jobbe med. Implementerer man `SingleFieldInputPlugin`, får man bare inn et element som tilsvarer et felt i en entitet, mens i `EntityListInputPlugin` får man inn en liste av like elementer som gjenspeiler en liste av entiteter.

SingleFieldInputPlugin grensesnittet

Eksempel 8-1 viser hvordan grensesnittet `SingleFieldInputPlugin` er bygget opp. Grensesnittet har en metode, som må implementeres, ved navn generateTag(). Som vist i eksempelet har metoden mange parametre. Parameteret `idOfResource` inneholder verdien som ligger i elementet i innholdsdokumentet. Her vil det typisk ligge et filnavn til et multimediaobjekt eller en id-streng til en ressurs som for eksempel er stasjonert på et annet nettsted. Datastrukturen `parameterMap` inneholder parametre som er konfigurert i filen `pluginPattern.xml`. Disse parametrene må taes hensyn til av pluginet og betegner for eksempel høyde og bredde til bilder.

Strengverdiene i pid, page, view, entityPath og patternId inneholder henholdsvis presentasjonen hvor pluginet skal benyttes, view-et hvor HTML fra plugin-et skal benyttes, den unike stien til entiteten hvor feltet som skal rendres befinner seg og mønsteret som presentasjonen bygger på. Disse parametrene kan for eksempel brukes til å bygge opp linker i systemet eller til filtrering av felt i pluginet.

Dataaksessobjektet resourceDao er tilgjengelig for at utvikleren skal kunne hente filer eller opprette nye filer fra persistenslaget. Strengene `UrlToFilesRequiredByPlugin`, `UrlToPluginGeneratedFiles` og `UrlToMediaResources` inneholder URL-er til henholdsvis filer som pluginet trenger for å rendre korrekt resultat, filer som plugin-et selv oppretter og til multimediafiler som skal rendres av pluginet. Returverdien til metoden må ha typen Element som har sin opprinnelse fra XML biblioteket JDOM (se [41]). Dette biblioteket må importeres i klassene som skal implementere plugins.

```
/**
     *
     * @param idOfResource The identifier of the resource that will be
              displayed
     * @param parameterMap Map of parameters that the will customize the
              output of the plugin
     * @param pid The presentation id
     * @param page The page Id
     * @param view The id of the view
     * @param entityPath Path to a single element in the list listOfNodes
     * @param resourceDao Data Access object that can be used for CRUD
              operations in the persistence layer
     * @param patternId The id of the pattern
     * @param UrlToFilesRequiredByPlugin URL to files required by the
              plugin. For example a flashfile for the flashvideoplayer.
     * @param UrlToPluginGeneratedFiles URL to files generated by the
              plugins. For example a playlist file generated by a flash
              mediaplayer plugin
     * @param UrlToMediaResources URL to multimediafiles that are going to
              be displayed. For example a video file.
     * @return The XML element that contains HTML for rendering a
               multimedia object.
     */
public interface SingleFieldInputPlugin {

     public Element generateTag(String idOfResource,
             Map<String,String>parameterMap, String pid,String page,
             String view, String entityPath, ResourceDao resourceDao,
             String patternId, String UrlToFilesRequiredByPlugin,
             String UrlToPluginGeneratedFiles, String UrlToMediaResources);
}
```

Eksempel 8-1: Grensesnittet SingleFieldInputPlugin

EntityListInputPlugin grensesnittet

Eksempel 8-2 viser oppbyggingen av grensesnittet `EntityListInputPlugin`. Også dette plugin-et har en metode som må implementeres og har navnet `generateTagWithEntityListInput()`. Denne metoden har mange av de samme parametrene og samme returverdi som metoden `generateTag()` i `SingleFieldInputPlugin` grensesnittet. Men det er en parameter som er forskjellig, og det er `listOfNodes`. Dette er en datastruktur som inneholder en liste av like elementer som skal prosesseres av et plugin. Eksempel 8-3 viser et eksempel på slike elementer. En liste kan for eksempel inneholde mange trackEntity-elementer. I dette eksempelet benyttes listen til å lage en musikkspilleliste. Hvert trackEntity-element inneholder informasjon som tittel, filnavn til sangen, bilde som skal vises i spilleren og en beskrivelse av sangen.

Under konstruksjonen av mønster er det viktig at man bygger opp strukturen på elementene slik som plugin-et krever. Med andre ord må man bestemme seg for hvilket plugin som skal benyttes i en gitt kontekst før en definerer strukturen på elementene i mønsteret. Hvis man i Eksempel 8-3 endrer strukturen på trackEntity elementene i mønsteret, vil ikke det samme plugin-et rendre et ønsket resultat.

```
/**
     *
     * @param listOfNodes List of equal elements that will be processed by
              a plugin
     * @param nameInPluginPattern The logical name of the plugin
     * ... Se grensesnittet for SingleFieldInputPlugin for beskrivelse av
         de resterende parametrene
     */
public interface EntityListInputPlugin {

    public Element generateTagWithEntityListInput(List listOfNodes,
            ResourceDao resourceDao,String nameInPluginPattern,
            Map<String,String> parameterMap,String pid, String page,
            String view, String entityPath, String patternId,
            String UrlToFilesRequiredByPlugin,
            String UrlToPluginGeneratedFiles, String UrlToMediaResources);
}
```

Eksempel 8-2: Grensesnittet EntityListInputPlugin

```
...
<trackEntity>
     <title type="string">Ferry cross the mersey</title>
     <location type="file">Ferry_Cross_The_Mersey.mp3</location>
     <image type="file">liverpool.jpg</image>
     <annotation type="string">Ferry cross the mersey</annotation>
</trackEntity>

<trackEntity>
     <title type="string">You'll never walk alone</title>
     <location type="file">Liverpool_FC.mp3</location>
     <image type="file">liverpool.jpg</image>
     <annotation type="string">You'll never walk alone</annotation>
</trackEntity>

...
```

Eksempel 8-3: En liste av like elementer som kan prossesseres av et plugin

8.2 Metoder for å bygge opp HTML i plugins

Hovedoppgaven til plugins er å generere HTML kode for fremvisning av multimediaobjekter i en nettleser. Oppbyggingen av HTML i plugins kan utføres på to forskjellige måter.

Bygge opp HTML elementer som en streng

Den første og mest primitive måten går ut på representere HTML elementer som en streng. En oppretter da et JDOM rotelement og setter HTML-strengen som verdien til dette elementet. Dette er en enkel metode, men den er også litt problematisk. For det første får man ingen hjelp til å skrive velformet HTML. Det er for eksempel veldig lett å glemme å lukke elementer, glemme anførselstegn på attributter, også videre.

Resultatet som metoden i plugin-et returnerer, erstatter verdien i det prosesserte elementet fra XML-dokumentet som representerer innholdstreet. Innholdstreet blir i neste steg sendt videre

til transformasjon i et XSLT-stilark. Hvis HTML-koden som plugin-et produserte er lagret i en streng har man ingen mulighet til å utføre transformasjon på HTML-koden. Dette er fordi XSLT-kode, som utfører transformasjonen, prosesserer XML-elementer. Den eneste muligheten som finnes er å skrive ut HTML-koden i elementet uforandret ved bruk av XSLT-kode som vist i Eksempel 8-4. Utgangspunktet for eksempelet er at man har fått en match på elementet som har attributten `type` lik plugin. Koden sier her at man skal skrive ut HTML-strengen som ligger i rotelementet som plugin-et produserte. I tillegg sier man at innholdet ikke skal escapes, noe som i så fall ville ødelagt HTML-koden.

```
<xsl:value-of select="./*" disable-output-escaping="yes" />
```

Eksempel 8-4: XSLT kode for å skrive ut HTML, fra et plugin, som er lagret i en streng.

Bygge opp HTML elementer som et XML dokument

Den andre metoden, som kan benyttes for å bygge opp HTML-kode, går ut på å representere HTML-elementer som JDOM [41]-elementer. Dette er en tryggere metode å bruke for å utvikle korrekte plugins. Her trenger man ikke å tenke på å for eksempel lukke elementer eller å huske anførselstegn på attributter. Grunnen til dette er at man bygger en struktur av objekter som kobles sammen. Hvert objekt representerer et HTML element.

Rotelementet som inneholder objektrepresentasjon av HTML elementer retuneres og kobles på XML dokumentet som representerer innholdstreet. Under transformasjonen har en nå mulighet til å manipulere resultatet fra plugin-et ytterligere. Et eksempel på hvor dette kan være nyttig, er når man har en entitet som blant annet inneholder en liste med bildeentiteter som vist i Eksempel 8-5. Listen av bildeentiteter kan inneholde bilder fra mange ulike kategorier. Kategorien til hvert bilde er her representert ved feltet `type`. Ønsker man for eksempel at bildegalleriet kun skal vise bilder fra en bestemt kategori, kan man ved hjelp av XSLT filtrere bort bilder fra de andre kategoriene. På den måten samarbeider plugin-et og transformasjonsarket om å generere et resultat som skal sendes videre til nettleseren.

```
<entity id="imageListEntity">
   <fields>
      <field type="string">type</field>
      <field type="string" required="true">title</field>
      <field type="file">image</field>
   </fields>
</entity>

<entity id="imageCollectionEntity">
      <fields>
         <field type="string" required="true">title</field>
         <field type="string" required="true">description</field>
         <field type="entity-list" ref-id="imageListEntity"
                pluginConfig="photogallery" >imageEntities</field>
      </fields>
</entity>
```

Eksempel 8-5: Her kan være nyttig å samarbeide med et XSLT stilark om filtrering av kode generert av et plugin

Bruker man JDOM-elementer i oppbyggingen av HTML-elementer kan man ikke bruke XSLT-koden fra Eksempel 8-4 for å skrive ut HTML koden. Grunnen til det er at man ved bruk av den metoden bare vil kunne skrive ut eventuell tekst i rotelementet og ikke

elementene som ligger inne i rotelementet. Man må i stedet bruke XSLT-koden som Eksempel 8-6 viser. Her tar man en dyp kopi av alle elementene og skriver dem ut uten å escape noen av karakterene. Resultatet vil bli HTML kode som kan retuneres til nettleseren.

```
<xsl:copy-of select="./*"/>
```

Eksempel 8-6: XSLT kode for å skrive ut HTML, fra et plugin, som er lagret som elementer

I plugins-ene som er utviklet for DPG 2.0 er begge metodene, som er beskrevet ovenfor, benyttet. Erfaringer viser at plugins som skal generere mye HTML-kode bør bygges ved hjelp av XML-elementer. Dette er fordi det er veldig enkelt å glemme å lukke elementer eller å glemme anførselstegn for attributter, også videre. Skal man lage et enkelt plugin som generer et HTML-element for å vise et enkelt bilde, er det nok enklere og raskere å bygge opp HTML-koden som en streng.

8.3 To kategorier med multimedia-plugins

Plugins-ene sin hovedoppgave er å gi DPG 2.0 støtte for å vise multimediaobjekter i presentasjonene. Hva som må til for å vise multimediaobjekter varierer og er opp til plugins-ene å bestemme. Den mest brukte metoden er å benytte seg av plugins som er installert i nettleseren. Eksempler på dette er Quicktime-plugin, Pdf-plugin og Windows media player-plugin. Men det finnes også andre metoder, som for eksempel å benytte seg av mediaspillere som er utviklet med Flash-teknologi, eller å benytte ressurser som allerede er tilgjengelige på nettet eller utvikle visningen av multimediaobjektet selv.

Når det kommer til utvikling av plugins, er det naturlig å dele dem i to kategorier. Den første kategorien er de plugins-ene som tar inn et multimediaobjekt og kun refererer til korrekt avspillingsmetode ved hjelp av HTML. Denne kategorien kalles primitive plugins. Den andre kategorien inneholder plugins, hvor det i tillegg til å referere til korrekt avspillingsmetode, også må gjøres endel prosessering i plugin-et for å oppnå ønsket resultat. Denne kategorien kalles komplekse plugins.

Primitive plugins

Denne kategorien inneholder plugins hvor man kun bygger opp HTML kode som refererer til korrekt avspillingsmetode for multimediaobjektet. Et eksempel på et slikt plugin er PdfPlugin (Se Eksempel 8-7). I eksempelet ser vi hvordan et HTML object-element representeres ved hjelp av XML. I object elementet settes mime-typen til PDF-filen. Dette hjelper nettleseren å finne korrekt plugin som skal benyttes for å vise PDF-filen. I tillegg må URL-en til PDF-filen settes og eventuelle parametre. Som eksempelet viser, skjer det ingen prosessering utover å bygge opp HTML som kaller korrekt avspillingsmetode for PDF-filen.

```
    ...
    //Bygger opp et HTML object element som et XML element.
      Element root =new Element("object");

          //Setter type attributten til object elementet lik mime-typen
          til en PDF-fil.
          root.setAttribute("type", "application/pdf");

          //Setter dataattributten lik URL til PDF fil
          root.setAttribute("data", UrlToMediaResources+fileName);
          root.setAttribute("width", width);
          root.setAttribute("height", height);

         //Hvis nettleseren ikke har installert et PDF-plugin vises i
         stedet en link til PDF filen
          Element paramOne =new Element("a");
             paramOne.setAttribute("href", UrlToMediaResources+fileName);
             paramOne.setText(UrlToMediaResources+fileName);

      root.addContent(paramOne);

      return root;
  }
    ...
```

Eksempel 8-7: Oppbygging av et primitivt plugin

Komplekse Plugins

Denne kategorien inneholder plugins som utfører endel prosessering for at ønsket resultat skal vises i nettleseren. Med prosessering menes foreksempel å generere en spilleliste til en mediaspiller, bygge opp HTML som gir tekst fra en tekstfil en ny stil når den vises i nettleseren eller bygge opp HTML elementer som gjør at bilder kan vises ved hjelp av eksterne bildegalleri.

Et eksempel på et slikt plugin er JavaToHtmlPlugin(Se Eksempel 8-8). Plugin-et henter ut java-filen, som er referert til av parameteren `fileName`, fra persistenslaget. Filen prosesesseres ved hjelp av biblioteket JavaToHtml [42] og resultatet er HTML som gir innholdet fra filen en stil som gjør at utseende i nettleseren blir det samme som utseende i en javaeditor. Prosesseringen skjer i linje 1 til 7. I linje 8 og 9 bygges HTML elementet som skal sendes videre til nettleseren.

Som vist i eksempelet, har utvikleren store muligheter til å lage komplekse plugins. En har tilgang til persistenslaget som gjør at en kan preprosessere multimediafiler før visning, og en kan generere nye filer, som må taes hensyn til av applikasjoner som viser mediaobjektet i nettleseren. En kan også, som plugin-et JavaToHtml viser, utvikle visningen av multimediaobjektet på egenhånd, det vil si at en ikke bruker en ekstern applikasjon til å vise resultatet i nettleseren.

```
        ...
         //Finner javafilen som skal prosesseres i persistenslaget til
         applikasjonen
      1.  ResourceFile resourceFile=
                resourceDao.getPresentationResourceById(pid, null,
                fileName, resourceType, ResourceDao.DO_GET_BINARY_CONTENT);

        //Oppretter en InputStream som inneholder det binære innholdet til
        javafilen
      2. InputStream is=
                 new ByteArrayInputStream(resourceFile.getBinaryContent());

        //Oppretter en referanse til en javakilde.
      3. JavaSource source=null;
        ...
        //Parser strømmen slik at den nå svarer til en Javakilde
      4.  source = new JavaSourceParser().parse(is);
        ...

        //Oppretter en konverter og skriver javakilde objektet som HTML i
          en StringWriter
      5. JavaSource2HTMLConverter converter = new
                                        JavaSource2HTMLConverter();
        ...
        //StringWriter hvor den konverterte javakoden legges.
      6  StringWriter writer = new StringWriter();

        ...
        //Her skjer selve konverteringen
      7  converter.convert(source,...,writer);
        ...

        //Setter HTML som konverteren genererte inn i et HTML div element
        (Representert som XML) og retunerer dette elementet.
      8  Element root = new Element("div");

      9  root.setText(... writer.toString()...);


        return root;
    }
 ...
```

Eksempel 8-8: Prosessering i et komplekst plugin

8.4 Plugins som er utviklet til DPG 2.0

På bakgrunn av kravene til multimedia, som er definert i delkapittel 4.1, er det laget plugins som gjør at DPG 2.0 kan oppfylle disse. I Tabell 8-1 vises en oversikt over pluginsene i venstre kolonne og en referanse til en ny plugintabell, som beskriver plugins-ene mer inngående, i høyre kolonne. Plugins-ene er en viktig del av denne oppgaven.

PluginNavn	Referanse til tabell
AardvarkMapPlugin	Plugin 12: Plugin som genererer HTML for fremvisning av et kart.
AudioMp3Plugin	Plugin 1: Plugin som genererer HTML for avspilling av MP3 musikkfiler
FlashMediaPlugin	Plugin 10: Plugin som genererer HTML for fremvisning av SWF (Shockwave Flash) filer
FlashYoutubeVideoPlugin	Plugin 6: Plugin som genererer HTML for fremvisning av SWF (Shockwave Flash) spilleren til nettstedet www.youtube.com.
ImageMediaPlugin	Plugin 3: Plugin som genererer HTML for fremvisning av et bilde.
JavaToHtmlPlugin	Plugin 13: Plugin som genererer HTML for fremvisning av javakode i nettleseren med syntaksmerking.
MusicPlayerWithPlayListPlugin	Plugin 2: Plugin som genererer HTML for avspilling av MP3 filer som ligger i en spilleliste.
PdfPlugin	Plugin 11: Plugin som genererer HTML for fremvisning av PDF filer.
PictureGalleryClassicPlugin	Plugin 4: Plugin som genererer HTML for fremvisning av bilder i et bildegalleri.
ThickboxFilterTypePictureGalleryJQueryPlugin og ThickboxPictureGalleryJQueryPlugin	Plugin 5: To plugins som benytter seg av eksterne Javascript og CSS-filer for å vise bilder i et bildegalleri
VideoMediaPlugin	Plugin 7: Plugin som genererer HTML for fremvisning av videoer.
VideoPlayerWithPlayListPlugin	Plugin 8: Plugin som genererer HTML for fremvisning av en videospiller med en spilleliste.
XSPFMediaPlayerPlugin	Plugin 9: Plugin som genererer HTML for fremvisning av en videospiller med en spilleliste ved bruk av en ekstern videospiller utviklet i flash.

Tabell 8-1 Oversikt over plugins-ene og en referanse til hvor en kan finne mer informasjon om dem.

Hvordan ta i bruk plugins

Når et plugin er installert, er det presentasjonsmønsteret og den tilhørende pluginkonfigurasjonfilen som bestemmer hvor det skal brukes. Dette skaper stor fleksibilitet i plugins modulen og fører til at man enkelt kan bytte ut plugins med nye oppdaterte plugins. I Eksempel 8-9 vises et eksempel på hvordan man tar i bruk et plugin. I mønsterdefinisjonen settes `type` attributten lik plugin og attributten `pluginConfig` til en logisk verdi som gjenspeiler plugin-et som skal benyttes. I konfigurasjonsfilen pluginPattern.xml bindes den logiske verdien som er definert i mønsteret opp mot et reelt plugin. Det eneste som gjenstår da er å gi en referanse til multimediaobjektet i innholdsdokumentet.

```
<!--Mønster definisjon -->
<entity id="articleEntity">
            <fields>
              ...
              <field type="plugin" pluginConfig="singleImage">image</field>
            </fields>
</entity>


<!--pluginPattern.xml XML som binder logisk pluginnavn fra mønsteret opp
mot et reelt plugin -->
<plugin-config id="singleImage" plugin-name="ImageMediaPlugin">
    ...
</plugin-config>


<!--Innholdsdokument der en definerer hvilket bilde som skal vises i
nettleseren-->
<articleEntity ...>
      ...
      <image type="plugin">nyhet_1.jpg</image>
</articleEntity>
```

Eksempel 8-9: Oppsett for å ta i bruk plugins

9 Multimedia i ulike presentasjonsmønstre

Dette kapittelet reflekterer over hensikten med å lage støtte for multimedia i DPG 2.0. Det vil her bli gjennomgått to mønstre som er utviklet for å vise at multimedia er en stor bidragsyter til å få frem innholdet i et budskap. Mønstrene dekker ulike domeneområder som igjen viser at multimedia er en viktig ingrediens i et hvilket som helst CMS.

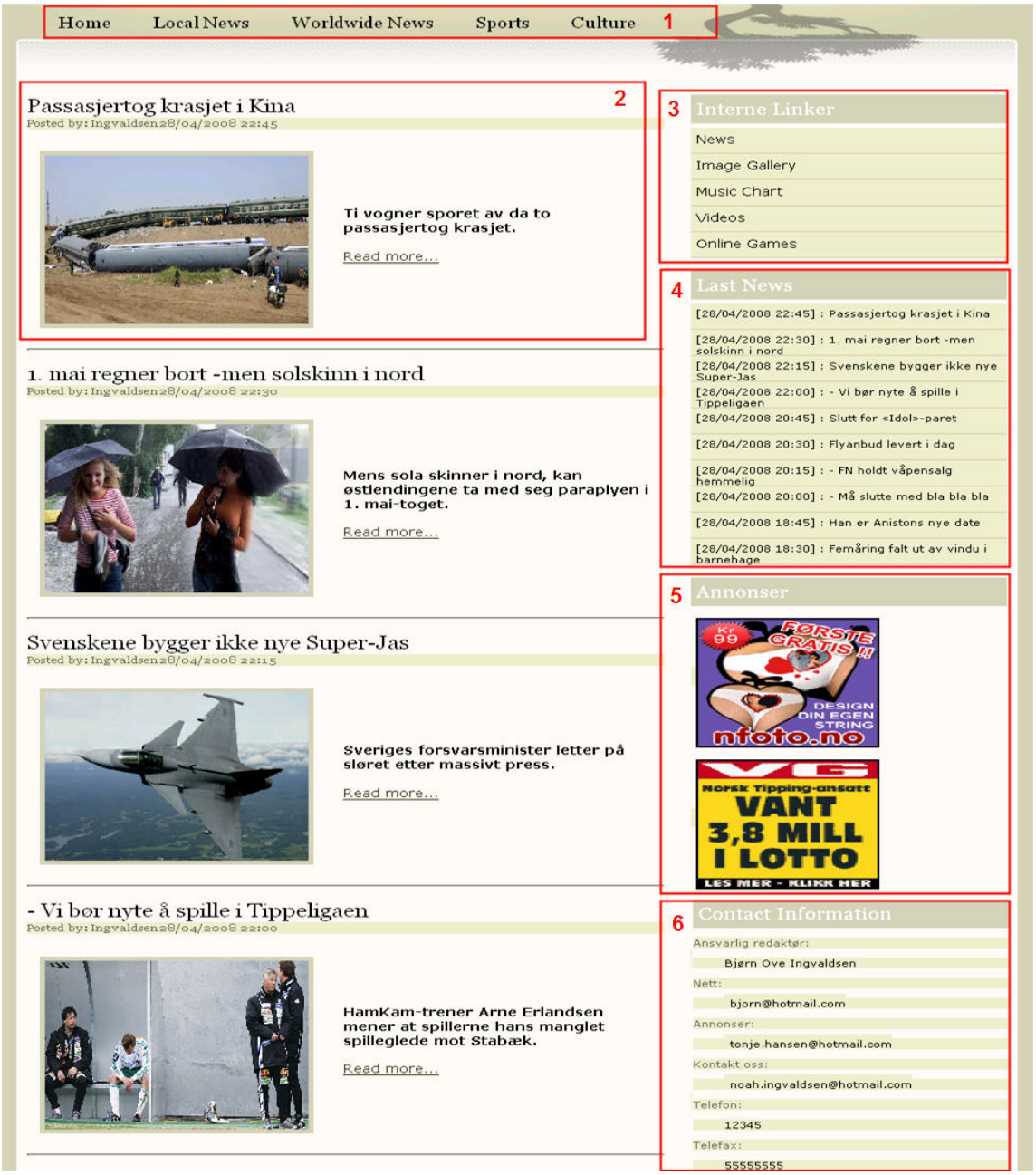
9.1 Paper Pattern (Avismønster)

Avismønsteret er et mønster som gjenspeiler en mal til en online avis. Mønsteret er noe forenklet, men nytteverdien til multimedia kommer klart frem. En demopresentasjon av mønsteret er laget og brukes som eksempel for å belyse de viktigste momentene. Figur 9-1 viser en forenklet versjon av startsiden i presentasjonen og skal hjelpe til å beskrive oppbyggingen av mønsteret.

Oppbygging av mønsteret

Mønsteret er bygget opp på følgende måte:

- Inneholder fem page-elementer eller sider som modellerer følgende (Se hovedmeny i boks 1 i Figur 9-1):
    - En startside hvor siste nytt uansett kategori vises.
    - En side for innenriksnyheter.
    - En side for utenriksnyheter.
    - En side for sportsnyheter.
    - En side for kulturnyheter.
- Hvert page-element inneholder diverse views, som enten dukker opp i viewmenyen (se boks 3 i Figur 9-1) eller som alltid er synlig på siden. Alle page-elementene i avismønsteret har noen få views som er felles og som alltid er synlige på siden. Dette er views for å vise reklame og kontaktinformasjon. Se henholdsvis boks 5 og 6 i Figur 9-1.
- Hvert page-element gjenspeiler en nyhetskategori, utenom page elemenet som representerer startsiden som inneholder det siste nye på nyhetsfronten uansett kategori. Alle page-ene har et standard view som viser nyhetene i sin kategori slik som boks 2 i Figur 9-1 viser. I tillegg har hvert page-element et view som viser en nyhetsboks med de siste ti nyheter i sin kategori (se boks 4 i Figur 9-1), et view som viser et bildegalleri med bilder fra nyheter i sin kategori, og tilslutt et view som viser videoer fra alle nyhetene uansett kategori (se boks 3 i Figur 9-1).
- Page elementet som representerer startsiden har i tillegg et view som tilbyr spill og et view som viser en musikk hitliste (Se boks 3 i Figur 9-1).
- Navigasjonsmenyen og banner er ikke tatt med på bildet.



Figur 9-1: Forsiden av en presentasjon som bygger på avismønsteret

Multimedia i mønsteret

Multimedia er benyttet for ulike formål i avismønsteret. Eksempler på dette er spill og musikkhitliste til underholdning, bilder og videoer som hjelper å fremme budskapet i en artikkel. Her følger en gjennomgang av de viktigste entitetene som benytter seg av plugins for å rendre HTML for fremvisning av multimediaobjekter.

ArtikkelEntitet

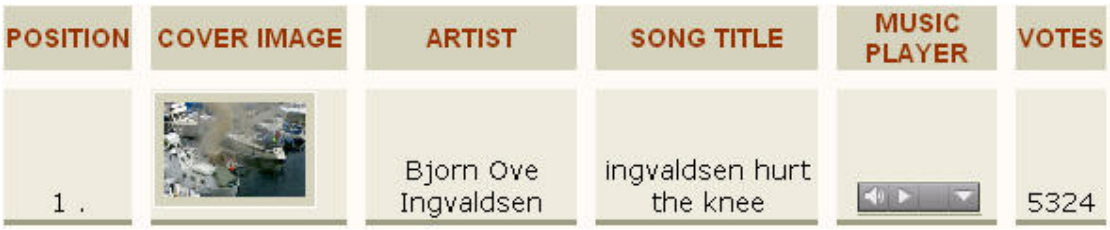
En artikkel kan inneholde tittel, ingress, bilde, video og selve innholdet representert som tekst. I boks 2 i Figur 9-1 vises hvordan en artikkel presenteres på startsiden. Trykker man på

linken til artikkelen, vises hele artikkelen som vist i Figur 9-2. Her vises hvordan multimedia benyttes for å fremme budskapet til artikkelen. Bildet og videoen gir artikkelen mer tyngde ved å utrykke innholdet på en annen måte enn ren tekst. Dette gir leseren alternativer og skaper interesse hos leseren. Rendring av HTML for bilde skjer i ImageMediaPlugin mens rendring av HTML for video skjer i VideoMediaPlugin.

Figur 9-2: Hvordan en artikkel i avismønsteret ser ut.

Musikkentitet

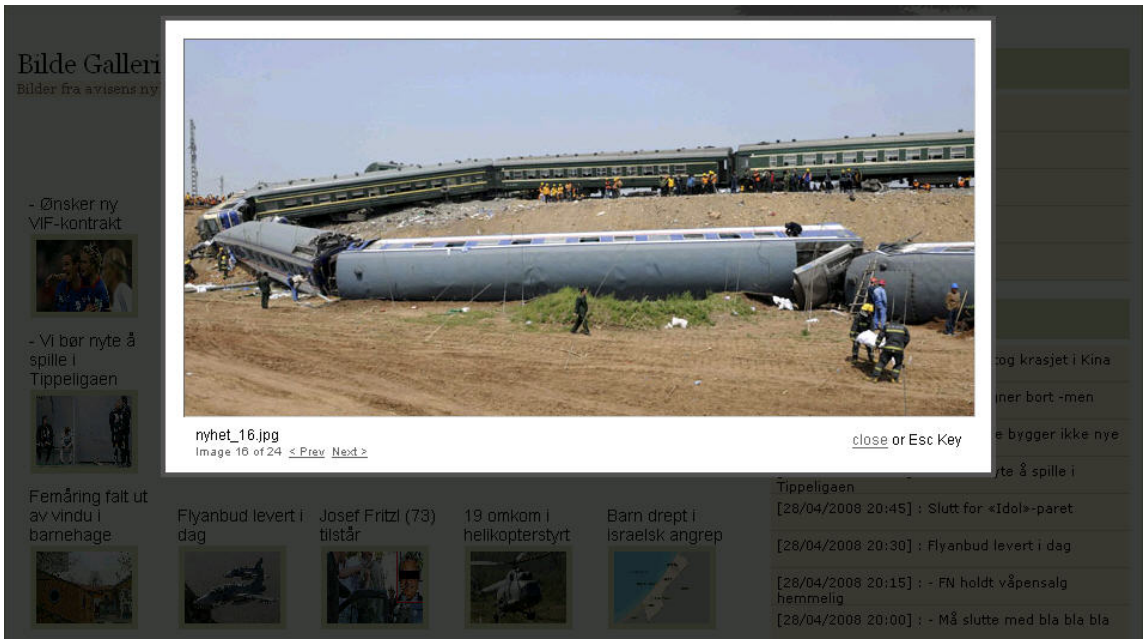
Boks 3 i Figur 9-1 viser en viewmeny med blant annet en link til en musikkhitliste. Et musikkelement i en hitliste kan inneholde artistnavn, tittel på sangen, bilde, lydsnutt og antall stemmer som sangen har fått. Figur 9-3 viser en presentasjon av den første sangen i avisens musikkhitliste. Figuren viser hvordan multimediaobjekter hjelper å få frem informasjon som ren tekst ikke har mulighet til å utrykke. Rendring av HTML for bilde skjer ved hjelp av ImageMediaPlugin, mens rendring av HTML for musikkspilleren skjer ved hjelp av AudioMp3Plugin.



Figur 9-3: Et av musikkelementetne i avismønsteret sin hitliste

Bildegalleri entitet

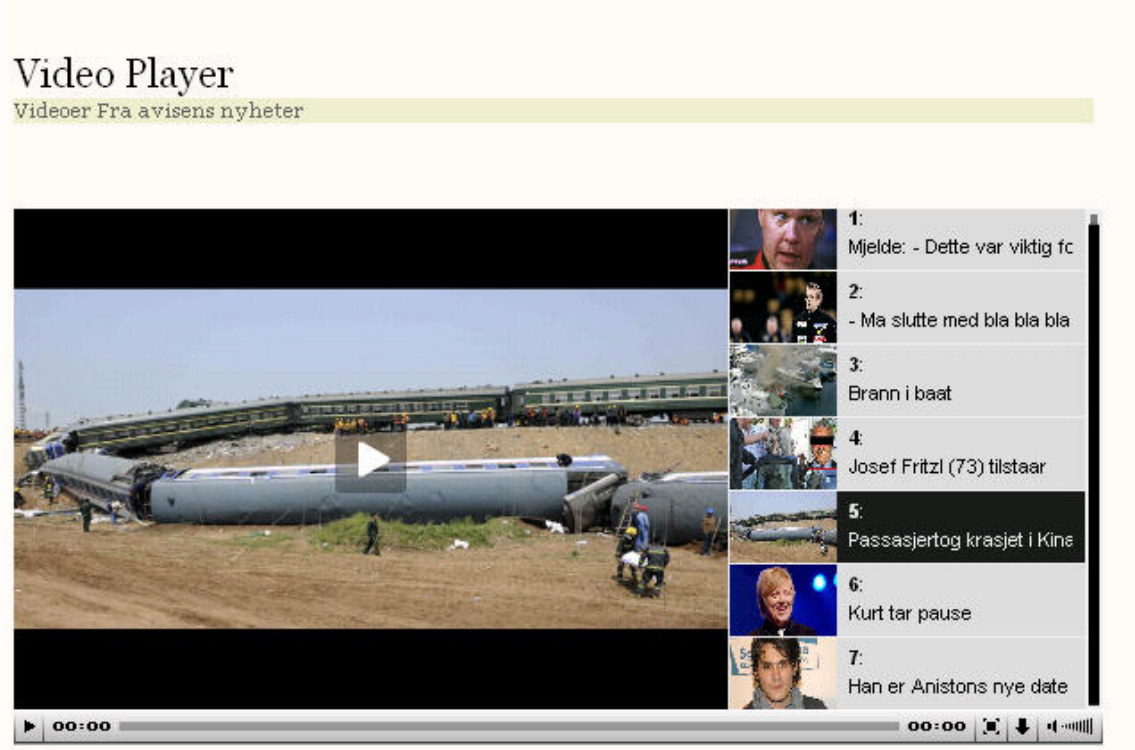
Et bildegalleri inneholder en samling av bildeentiteter. Hver bildeentitet har en tittel og et bilde. Figur 9-4 viser hvordan et bilde fra galleriet presenteres i nettleseren. Et bildegalleri som inneholder bilder, samt tittelen til nyheten det er hentet fra, gir leseren en rask visuell presentasjon av nyhetene. Rendring av HTML for hele bildegalleriet skjer ved bruk av `ThickboxFilterTypePictureGalleryJQueryPlugin`.



Figur 9-4: Bruken av bildegalleri i avismønsteret

Videosamling Entitet

I avismønsteret er videosamlingen representert ved en videospiller med en tilhørende spilleliste med videoer fra nyhetene. Figur 9-5 viser hvordan spilleren vises i nettleseren. Videopresentasjon av nyheter er en god måte og formidle nyheter på og gir leserne et nytt alternativ til hvordan de ønsker nyhetene presentert. I tillegg kan video være et godt tilleggssupplement til en tekstbasert nyhetsartikkel. Rendring av HTML for fremvisning av videospilleren utføres av XSPFMediaPlayerPlugin.



Figur 9-5: Bruken av videoavspiller i avismønsteret

Andre entiteter som benytter seg av multimedia

I avismønsteret er det også andre entiteter som benytter seg av multimedia. View-et, som tilbyr onlinespill til leserne, inneholder en entitet som gjør bruk av FlashMediaPlugin til rendringen av flash-spillet. Reklameview-et inneholder en entitet som benytter seg av ImageMediaPlugin for å vise bilder som skal fange leserens oppmerksomhet.

9.2 Football Pattern (Fotballmønster)

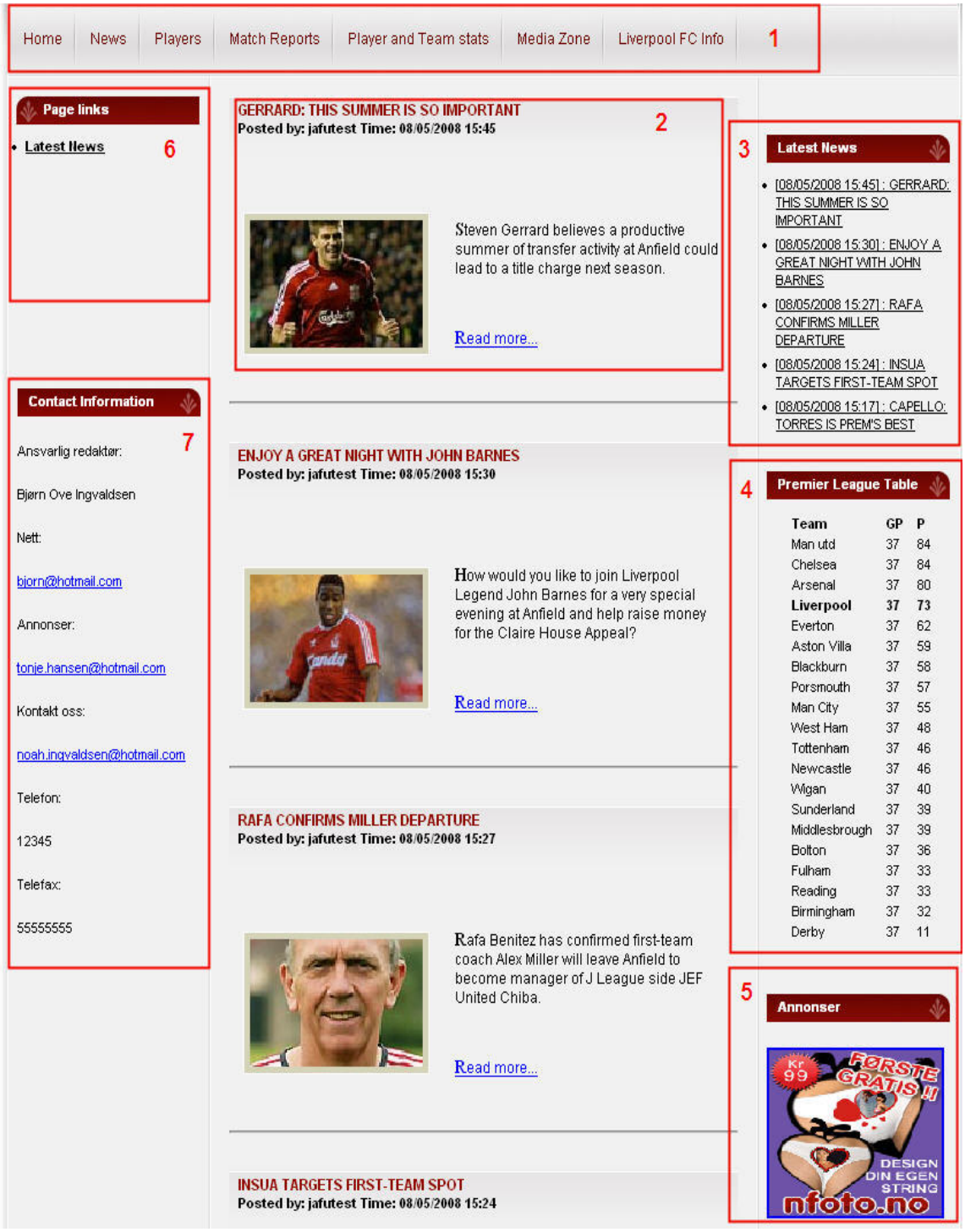
Fotballmønster er et mønster som gjenspeiler en mal for presentasjon av en fotballklubb på internet. Også dette mønsteret har god nytte av å ta i bruk multimedia for gi informasjon til brukeren på flere forskjellige måter. Men mønsteret bruker også multimedia til å få frem et budskap i kontekster hvor tekst ikke kan gi samme informasjon på en tilfredstillende måte. Startsiden til en demopresentasjon av dette mønsteret er vist i Figur 9-6. (Skjermbildet er forenklet noe ved at banneren og navigasjonmenyen på toppen av siden er fjernet).

Oppbygging av mønsteret

Mønsteret er bygget opp på følgende måte:

- Inneholder syv page-elementer eller sider som modellerer følgende: (Se hovedmeny i boks 1 i Figur 9-6)
  - Startside ved navn `Home`.
  - Nyhetsside ved navn `News`.
  - Side for presentasjon av spillere, ved navn `Players`.

    - Side for presentasjon av kamprapporter, ved navn `Match Reports.`
    - Side for presentasjon av spiller og lag-statestikk, ved navn `Player and Team` stats.
    - Side for samling av multimediaobjekter, ved navn `Media Zone.`
    - Side for for presentasjon av fakta om fotballklubben, ved navn `Liverpool FC Info.`
- Felles for hvert page-element er fire views som alltid er synlige. Det er view for å vise de siste nyhetene, view for å vise en tabell, view for å vise reklame og view for å vise kontaktinformasjon. View-ene er markert i Figur 9-6 i henholdsvis boks 3,4,5 og 7.
- Startsiden inneholder et view som viser de fem siste nyhetene representert slik som boks 2 i Figur 9-6 viser.
- Nyhetssiden inneholder de samme view-ene som startsiden, men har i tillegg et view som viser eldre nyhetsartikler. En kan med andre ord få tilgang til hele arkivet med nyheter.
- Siden for presentasjon av spillere inneholder et view som presenterer spillerne i spillerstallen.
- Siden for presentasjon av kamprapporter inneholder et standard view som viser en kamprapport fra den siste kampen som er spilt. I tillegg inneholder siden et view som viser et arkiv med kamprapporter fra alle kampene som er spilt.
- Siden for presentasjon av spillere og lagstatistikk inneholder et standard view som viser statistikk for hvordan laget har prestert inneværende sesong. I tillegg er det view som viser statistikk for hvordan spillerne har prestert inneværende sesong, view som viser en liste over hvem som har gjort mål og et view som viser serietabellen i full versjon.
- Siden for samling av multimediaobjekter inneholder et standard view som viser en videospiller med en spilleliste. I tillegg er det view som inneholder et bildegalleri og et view som viser en musikkspiller med tilhørende spilleliste.
- Side for presentasjon av fakta om fotballklubben inneholder et view som presenterer fotballklubben og historiske resultater som er oppnådd.



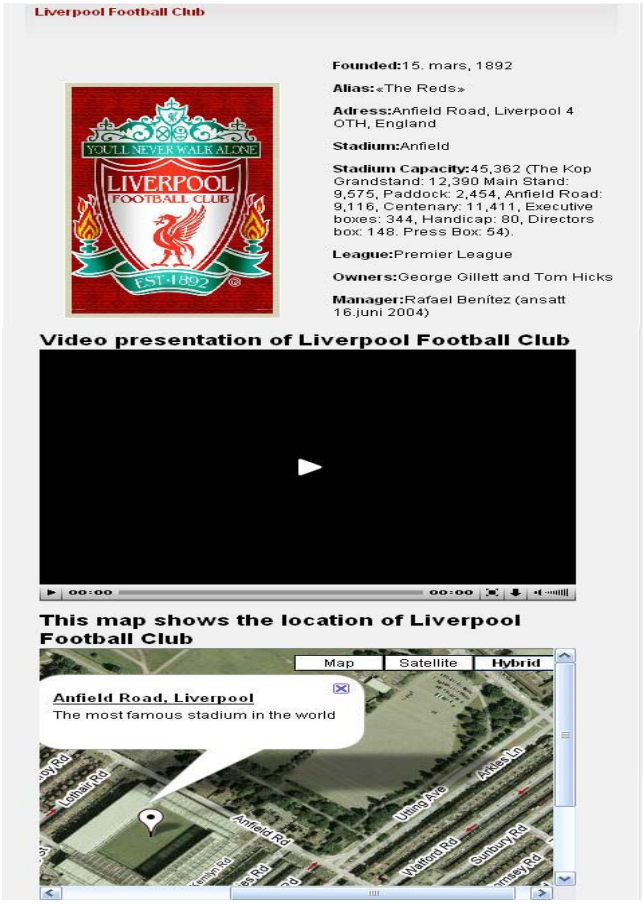
Figur 9-6: Forsiden av en presentasjon som bygger på fotballmønsteret

Multimedia i mønsteret

Multimedia er også i dette mønsteret benyttet for ulike formål. Eksempler på dette er video for å presentere fotballklubben og kart for å vise hvor fotballklubben hører hjemme. Her følger en gjennomgang av noen av entitetene som benytter seg av plugins for å rendre HTML for fremvisning av multimediaobjekter.

Fakta om fotballklubb entitet

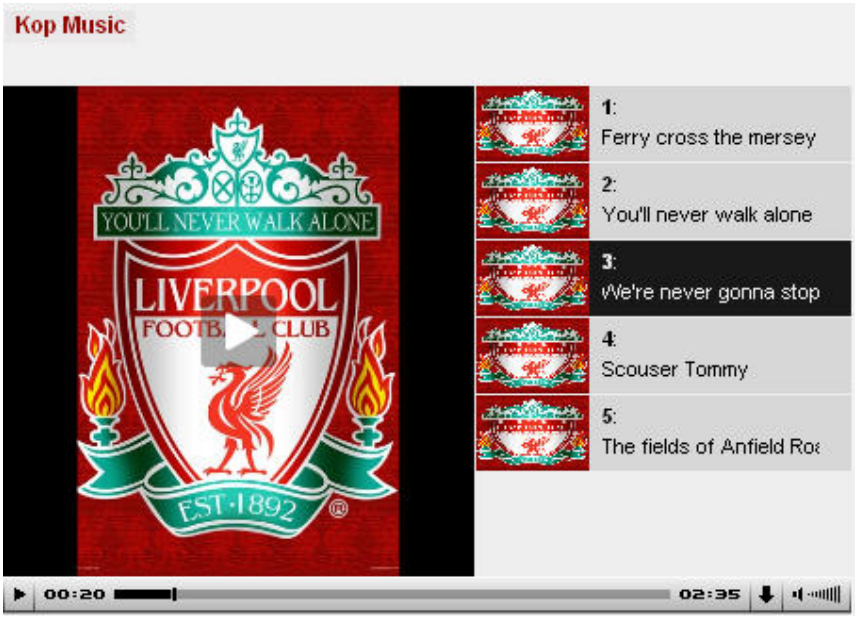
Denne entiteten inneholder felt som presenterer fakta om en fotballklubb. I tillegg til mange tekstbaserte felt, har entiteten tre felt som viser multimedia. Det er felt som viser logoen til klubben, en videopresentasjon av klubben, og et kart som viser hvor klubben befinner seg. Figur 9-7 viser hvordan siden, som inneholder fakta om en fotballklubb, ser ut i demopresentasjonen. Bildet av logoen til klubben og videopresentasjonen gir brukerne av siden tilleggsinformasjon om klubben som vil være umulig å gjengi eksakt ved kun å bruke tekst. Kartet hjelper også brukeren med å finne frem til hvor klubben hører hjemme ved å vise en grafisk presentasjon av området. Plugins-ene som benyttes for å rendre HTML for visning av multimediaobjektene er ImageMediaPlugin, VideoMediaPlugin og AardvarkMapPlugin.



Figur 9-7: Hvordan fakta om en fotballklubb kan se ut i fotallmønsteret

MusikkSamling Entitet

I fotball mønsteret er musikksamlingsentiteten representert ved en musikkspiller og en tilhørende spilleliste. Se Figur 9-8 viser hvordan spilleren ser ut i demopresentasjonen. Spillelisten inneholder supportersanger og gir brukeren mulighet å oppdatere seg på klubbens sanger. Musikkspilleren skaper interesse hos brukerne, noe som igjen fører til at trafikken på siden vil øke. Plugin-et som benyttes for å rendre HTML slik at spilleren vises i nettleseren heter XSPFMediaPlayerPlugin.



Figur 9-8: Bruken av en musikkavspiller i fotballmønsteret

Andre entiteter som benytter seg av multimedia

Det er mange andre entiteter i fotballmønsteret som benytter seg av multimedia for å fremme et budskap. Men på bakgrunn av at entitetene er mer eller midre lik entitetene som er presentert tidligere i kapittelet, nevnes de kun med få ord i dette avsnittet.

View-ene som viser nyhetsartikler inneholder en artikkelentitet som har felt for å vise bilder og video. Denne entiteten er den samme som artikkelentiteten i avismønsteret. View-ene som presenterer spillere og kamprapporter benytter også entiteter som har felt for bilde og video. Hensikten er den samme som for en artikkel, en skaper mer tyngde i innholdet ved å utrykke det på en annen måte enn ren tekst. Dette gir leseren alternativer og skaper interesse hos leseren.

View-ene som viser en videospiller med spilleliste og et bildegalleri inneholder entiteter som er helt lik henholdsvis videosamlings-entiteten og bildegalleri-entiteten i avismønsteret. Hensikten med videospilleren er å samle alle videoene fra en presentasjon på et sted, slik at brukerne enkelt kan finne frem til videoer som er av interesse. Hensikten med bildegalleriet er mye det samme. Her gir en brukerne mulighet å se bilder av spillere eller viktige hendelser på ett og samme sted.

10 Evaluering og forslag til videre arbeid

10.1 Måloppnåelse

Hovedmålet til denne oppgaven var å utvikle støtte for multimedia i presentasjonsmotoren til den nye DPG2.0. For å nå dette målet, var det endel delmål som måtte oppfylles. Disse målene er repetert under:

1. Det må utføres en grundig analyse av DPG 1.0 for å belyse løsninger som bør forbedres og løsninger som er gode og som bør inngå i det nye systemet.
2. Det må fastsettes hvilke krav man skal ha til multimediafunksjonalitet.
3. Utvikle en ny presentasjonsmønsterspesifikasjon som utfra analysen forbedrer den gamle spesifikasjonen og i tillegg gjør nødvendige utvidelser med hensyn til multimedia.
4. Det må utføres en analyse av hvordan multimediafunksjonalitet skal implementeres i det nye systemet.
5. Det må implementeres en ny motor som rendrer presentasjoner, hvor multimediafunksjonaliteten implementeres i henhold til analysen i punktet ovenfor.
6. Funksjonalitet som gjenspeiler kravene til multimedia må implementeres.
7. Det må utføres en analyse av nytteverdien til multimedia i DPG2.0.

Delmål 1 er oppnådd ved et samarbeid mellom Karianne Berg, Bjørn Christian Sebak og undertegnede. Resultatet er dokumentert i kapittel 3 og danner grunnlaget for denne oppgaven.

Delmål 2 er oppnådd ved hjelp av en analyse av hva som systemet må støtte. Her går det mest på hvilke mediatyper som skal taes hensyn til i DPG-en. Et overordnet resultat av analysen er dokumentert i delkapittel 4.1

Delmål 3 er oppnådd ved et samarbeid mellom Bjørn Christian Sebak, og undertegnede. Resultatet er en helt ny presentasjonsmønsterspesifikasjon hvor en blant annet kan definere bruken av multimedia. Det var multimediadelen i spesifikasjonen som undertegnede hadde hovedansvar for. Ellers var undertegnede sterkt involvert i avalysen av mønsterspesifikasjonen til Alessandro Rossini og Graziano Liberati og fungerte deretter som en samtalepartner og ideutveksler for Bjørn Christian Sebak som hadde hovedansvaret for resten av presentasjonsmønsterspesifikasjonen. I kapittel 5 er hele løsningen på presentasjonsmønsterspesifikasjonen presentert.

Delmål 4 er oppnådd ved hjelp av en analyse av hvordan multimediafunksjonalitet skal implementeres. Resultatet av analysen ble et standpunkt om at multimediafunksjonalitet skulle implementeres ved hjelp av en pluginmodul. Hovedgrunnen til dette er at man ønsker en fleksibel struktur hvor det er enkelt å legge til støtte for nye mediatyper og å oppdatere støtten til eksisterende mediatyper. Dette er dokumentert i delkapittel 4.2

Delmål 5 er oppnådd ved et samarbeid med Bjørn Christian Sebak. Motoren kalles Presentation Viewer PV hvor undertegnede hadde hovedansvar for å utvikle pluginmodulen til denne motoren.

Delmål 6 er oppnådd ved utvikling av ulike plugins som har ansvar for å bygge opp HTML for en bestemt mediatype. Det kan være flere ulike plugins for hver mediatype alt etter hvordan en ønsker at visningen skal se ut.

Delmål 7 er oppnådd ved hjelp av en analyse av bruken av multimedia i avismønsteret og fotballklubbmønsteret. Mønstrene er utviklet av undertegnede med det formålet å vise nytteverdien av å integrere multimedia i DPG2.0.

10.2 Problemer underveis i prosjektet

Etter at arbeidet med oppgaven startet, viste det seg at det oppsto noen uventede problemstillinger. Det var spesielt to ulike problemstillinger som krevde ekstra oppmerksomhet.

Det første problemet som viste seg var at ikke alle nettleserne fulgte W3C [43] sin HTML 4.01 [44] og XHTML 1.0 [45] spesifikasjon. Dette førte til problemer med å vise frem multimediaobjekter i alle nettleserne. Løsningen på problemet varierte med typen til multimediaobjektet og er dokumentert i katalogen som viser en oversikt over de ulike plugins-ene (Se Tabell 8-1). Inspirasjon til løsningen ble funnet i boken HTML, XHTML, & CSS, Sixth Edition: Visual QuickStart Guide [18].

Det andre problemet oppsto da Karianne Berg integrerte persistenslaget som hun skriver om i oppgaven sin [7]. Før hun integrerte persistenslaget, lå alle multimediafilene under roten til webserveren og kunne hentes frem ved hjelp av URL som pekte direkte til filen. Etter at hun integrerte det nye persistenslaget, var multimediafilene flyttet inn i en database. Samtidig benyttet hun en egen Java-klasse til å identifisere multimediafilene som ble forespurt. Hver forespørsel om en ressurs går til en egen Java Servlet [46] som utfra GET parametre i URL-en hentet frem den korrekte ressursen. Eksempel 10-1 viser en URL for å referere til mediaspilleren som benyttes i XSPFMediaPlayerPlugin. Her ser vi at Java Servlet-en som refereres til av getPluginResource.htm får inn to GET parametre som skal identifisere ressursen.

```
http://localhost:8080/dpg2/getPluginResource.htm?pluginName=XSPFVideoPlayer
Plugin&resourceId=mediaplayer.swf
```

Eksempel 10-1: URL til ressurser etter at det nye persistenslaget var implementert.

Problemet oppstår siden ulike mediaspillere som er implementert i Flash bruker en parameter til å definere flere egenskaper. Alle egenskapene skilles med et & tegn som er samme tegnet som benyttes for å skille GET parametre. Eksempel 10-2 illustrerer problemet. `Flashvars` er en attributt som hører til `object` elementet som lages i XSPFMediaPlayerPlugin. Denne attributten definerer flere egenskaper som URL til spilleliste, bredde til skjermen som viser en video, bakgrunnsfarge, forgrunnsfagre, og om man skal kunne laste ned mediaobjektet. Eksempelet viser her at det er umulig for flash applikasjonen å skille egenskapene fra hverandre siden det brukes GET parametre for å identifisere spillelisten. URL-en til spillelisten er definert under file. Men som det vises, brukes det & tegn inne i denne URL-en for å identifisere ressursen. Dette ødelegger strukturen til attributten flashvars siden '&' tegnet skal skille ulike egenskaper fra hverandre. Resultatet er at Flash applikasjonen ikke finner spillelisten. Se XSPFMediaPlayerPlugin i plugins-katalogen for å se bruken av flashvars.

```
flashvars="file=http://localhost:8080/dpg2/getPresentationResource.htm?pid=
jafuAvis&type=PLUGIN_RESOURCE&fileId=videoList_playlist.xml&displaywidth=40
0&backcolor=0xEEEEEE&frontcolor=0x000000&showdownload=true
```

Eksempel 10-2: En attributt i Object elementet som setter verdi til flere egenskaper.

Det finnes ikke en fullstendig løsning på dette problemet, men enkelte Flash applikasjoner tilbyr tolking av escapede tegn i URL-er. Tegnene det er snakk om er &, = og ?. Ved å escape karakterer i URL-er klarer flashapplikasjonen å skille de ulike egenskapene fra hverandre. Se Eksempel 10-3.

```
%3F= '?'
%3D= '='
%26= '&'

flashvars="file=http://localhost:8080/dpg2/getPresentationResource.htm%3Fpi
d%3DjafuAvis%26type%3DPLUGIN_RESOURCE%26fileId%3DvideoList_playlist.xml
```

Eksempel 10-3: Hvordan spesial karakterer blir escapet i URL-en til spillelisten

10.3 Forslag til videreutvikling

Det er enda en del nyttig funksjonalitet som ikke er implementert. Her kommer noen nyttige tips om hva undertegnede mener bør videreutvikles.

Nye plugins

Det kommer stadig nye mediatyper og nye behov for multimedia. Bruken av plugins for å støtte ulike mediatyper gjør jobben med å utvide multimediafunksjonalitet til en enkel oppgave. Løsningen er veldig fleksibel og utvidbar siden man ikke trenger å endre eller legge til noen kode i hovedapplikasjonen. Det eneste man trenger å gjøre, er å utvikle et plugin som kobles til hovedapplikasjonen på ett av de to mulige tilkoblingspunktene. Tilslutt er det konfigurasjon av plugIn.xml som gjenstår før plugin-et er klar til bruk.

Oppdatering av eksisterende plugins er også en enkel oppgave. Alle plugins pakkes som JAR-filer og legges i lib-mappen på serveren. Vil man oppdatere et plugin, er det bare å erstatte JAR-filen som representerer plugin-et med en ny JAR-fil med et oppdatert plugin.

Opplasting av plugins

Per i dag må plugins som skal installeres manuelt legges som en JAR-fil i lib mappen på serveren. I tillegg må en oppdatere plugins.xml med informasjon om plugin-et for at man skal kunne ta det i bruk. Her vil det være hensiktsmessig å la manageren tilby en opplastingsfunksjon for plugins. Og i tillegg tilby brukeren et skjema, som samsvarer med elementene i plugin.xml, hvor han kan fylle inn nødvendig informasjon om plugin-et.

Plugins-håndtering

I mønsteret er det, for et felt som skal taes hånd om av et plugin, definert en logisk id til et plugin. Denne id-en blir i XML filen pluginPattern.xml linket opp mot et eksakt plugin. Skal en for eksempel bytte plugin som skal benyttes for et bestemt felt, må en oppdatere denne pluginPattern.xml filen mauelt. Også her bør manageren tilby brukeren et grensesnitt for å manipulere denne filen.

Eclipse plugin for utvikling av multimediaplugins

Forfatteren av denne oppgaven har tidligere jobbet en del med Eclipse [47] plugins. Assistert hjelp i utvikligen og installasjon av plugins kan oppnåes ved hjelp av Eclipse plugins. En kan for eksempel opprette en veiviser i eclipse plugin-et som går gjennom de nødvendige stegene for å utvikle og installere et plugin.

10.4Utviklingsmetode

Fofatteren av denne oppgaven har vært medlem av en prosjektgruppe hvor hvert medlem har ansvar for et spesifikt problemområde. Hovedmålet til prosjektet var å utvikle en ny DPG, mens hovedmålet til denne oppgaven har vært å integrere multimediafunksjonalitet i det nye sytemet.

En av de største fordelene med å jobbe i en prosjektgruppe er at man får mulighet til å være med å utvikle et større system. Selv om man har sitt eget problemområde får man også en viss innsikt i hva de andre deltagerene arbeider med, noe som fører til at man også øker kompetansen sin på andre områder.

Utover det faglige lærer man selvsagt å samarbeide med andre medlemmer i utviklingen av et system. Man må ta hensyn til andre deltagere, og man må levere resultater til avtalte tidspunkt. Alt dette er nyttige erfaringer å ta med seg videre inn i arbeidslivet.

Bjørn Christian Sebak er et prosjektmedlem som undertegnede har samarbeidet mye med. I begynnelsen av prosjektet var det utprøving av endel ideer som hadde hovedfokus. Det ble da jobbet med å prototype [48] løsninger på ny presentasjonsmøsnterspesifikasjon. Ulike løsninger ble utprøvd før en kom til en konklusjon, og den endelige mønsterspesifikasjonen så dagens lys. Etter hvert som konturen av den nye mønsterspesifikasjonen tok form, ble det foretatt ny prototyping på annen funksjonalitet.

Prototyping er også en utviklingsmetode som undertegnede har benyttet i stor grad under utviklingen av plugins. En av årsakene til at denne metoden var verdifull, var at nettleserne ikke alltid fulgte HTML 4.01 standarden. Ved å utvikle prototyper av plugins-ene fikk jeg mer innsikt i problemområdene og kunne løse de mest kritiske problemene før jeg brukte tid på å implementere en fullversjon av plugin-et.

Referanser

[1] K. Cruickshanks, "Verktøy for generering av XML-baserte presentasjoner: JPGen - Java presentasjons generator," Masteroppgave, Universitetet i Bergen, Institutt for informatikk, 2004.

[2] C. M. Berg, "Statusrapport for presentasjonsmønstermotor (PMM)," Universitetet i Bergen, Institutt for informatikk, 2004.

[3] B. Boiko, *Content Management Bible*. Wiley Publishing, 2005.

[4] Y. Espelid, "Dynamic Presentation Generator," Masteroppgave, Universitetet i Bergen, Institutt for informatikk, Bergen, 2004.

[5] A. Rossini and G. Liberati, "Presentation Pattern Guide," Universitetet i Bergen, Institutt for informatikk, 2006.

[6] B. C. Sebak, "Dynamic Presentation Generator 2.0 - Utvikling av ny dynamisk presentasjonsgenerator og presentasjonsmønsterspesifikasjon," Masteroppgave, Universitetet i Bergen, Institutt for informatikk, Bergen, 2008.

[7] K. Berg, "Persistensproblematikk i DPG2.0," Masteroppgave, Universitetet i Bergen, Institutt for informatikk, Bergen, 2008.

[8] K. S. Løvik, "Webucator 3.0 Brukerhåndtering og aksesskontroll i DPG 2.0," Masteroppgave, Universitetet i Bergen, Institutt for informatikk, Bergen, 2008.

[9] J. Nordbotten, *Multimedia Information Retrieval Systems*. (not published yet), 2008.

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

[11] Enonic. Progressive web content management for Java - Enonic. [Online]. www.enonic.com

[12] Sun Microsystems. The Java Community Process (SM) program - JSRs: Java Specification Requests - detail JSR-170. [Online]. http://www.jcp.org/en/jsr/detail?id=170

[13] Senselogic AB. Senselogic. [Online]. www.senselogic.se

[14] db4objects. db4o :: Native Java and .NET Object Database :: Open Source. [Online]. www.db4o.com

[15] GNU Project. (2008, Feb.) GNU Project. [Online]. http://www.gnu.org/licenses/gpl.html

[16] Zope.org. Zope.org. [Online]. www.zope.org

[17] K. A. Mughal, "Presentation Patterns: Composing Web-based presentations," Universitetet i Bergen, Institutt for informatikk, 2003.

[18] E. Castro, *HTML, XHTML, & CSS, Sixth Edition: Visual QuickStart Guide*. Peachpit Press, 2006.

[19] Sun Microsystems. JavaServer Pages Technology. [Online]. http://java.sun.com/products/jsp/

[20] The Apache software fundation. Apache Tomcat. [Online]. http://tomcat.apache.org/

[21] M. Fowler, *Patterns of Enterprise Application Architecture*. Pearson Education, Inc, 2003.

[22] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of*

Reusable Object-Oriented Software. Addison-Wesley, 1994.

[23] The World Wide Web Consortium (W3C). Extensible Markup Language (XML). [Online]. http://www.w3.org/XML/

[24] Apache Software Foundation. (1999-2006) Apache Tomcat 6.0 - Realm Configuration HOW-TO. [Online]. http://tomcat.apache.org/tomcat-6.0-doc/realm-howto.html

[25] PostgreSQL. PostgreSQL. [Online]. http://www.postgresql.org/

[26] W3C The World Wide Web Consortium (W3C). HTTP - Hypertext Transfer Protocol. [Online]. http://www.w3.org/Protocols/

[27] Wikipedia. Wikipedia- URL. [Online]. http://en.wikipedia.org/wiki/Uniform_Resource_Locator

[28] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlaad, *Security patterns, Integrating securit and systems engineering*. John Wiley & sons,Ltd, 2006.

[29] K. Arnold, G. James, and D. Holmes, *The java programming language*, Third edition ed.. Addison Wesley, 2000.

[30] Microsystems, Sun. Javadoc. [Online]. http://java.sun.com/j2se/javadoc/

[31] L. Ø. Lervik, "Utvikling av nye presentasjonsmønstre," Masteroppgave, Universitetet i Bergen, Institutt for informatikk, Bergen, 2006.

[32] W3C The World Wide Web Consortium (W3C). The Extensible Stylesheet Language Family (XSL). [Online]. http://www.w3.org/Style/XSL/

[33] Spring Framework. Spring Framework. [Online]. http://www.springframework.org/about

[34] K. Berg, "Kravspesifikasjon for DPG 2.0," Universitetet i Bergen, Institutt for informatikk, 2008.

[35] L. Seth and D. Keith, *Expert Spring MVC and Web Flow*. Apress, 2006.

[36] G. Liberati, "INF219 Work Report," Universitetet i Bergen, Institutt for informatikk, 2006.

[37] Apache Software Foundation. The Apache Velocity project. [Online]. http://velocity.apache.org/

[38] The Apache Software Foundation. Apache Commons. [Online]. http://commons.apache.org/configuration/

[39] Wikipedia. Wikipedia- JAR (fileformat). [Online]. http://en.wikipedia.org/wiki/JAR_(file_format)

[40] Wikipedia. Wikipedia- Classpath (Java). [Online]. http://en.wikipedia.org/wiki/Classpath_(Java)

[41] JDOM. JDOM. [Online]. http://www.jdom.org/

[42] Java2Html. [Online]. http://www.java2html.de/

[43] W3C The World Wide Web Consortium (W3C). W3C. [Online]. http://www.w3.org/

[44] W3C The World Wide Web Consortium (W3C). HTML 4.01 Spesifikasjon. [Online]. http://www.w3.org/TR/REC-html40/

[45] W3C The World Wide Web Consortium (W3C). XHTML 1.0 Spesifikasjon. [Online]. http://www.w3.org/TR/2000/REC-xhtml1-20000126/

[46] Sun Microsystems. Sun Developer network. [Online]. http://java.sun.com/products/servlet/

[47] Eclipse. Eclipse. [Online]. http://www.eclipse.org/home/categories/languages.php

[48] Wikipedia. Wikipedia- Software prototyping. [Online]. http://en.wikipedia.org/wiki/Software_prototyping

[49] J. Wijering. JW FLV MEDIA PLAYER 4.0. [Online]. http://www.jeroenwijering.com/?item=JW_FLV_Media_Player

[50] The XIPH open source community. XSPF. [Online]. http://xspf.org/

[51] Wikipedia. Wikipedia- SWF. [Online]. http://en.wikipedia.org/wiki/SWF

[52] Aardvark Map. Aardvark Map. [Online]. http://www.aardvarkmap.net/

PluginsKatalog

Audio plugins

Her følger en presentasjon av plugins som dekker kravet om rendre HTML for å tilby avspilling av lyd.

AudioMp3Plugin

Egenskaper	Verdi
Kategori	Primitive plugins
Grensesnitt	SingleFieldInputPlugin
Ansvarsområde	Generere HTML for avspilling av MP3 musikkfiler
Filtyper/ Mediatyper	MPEG-1 Audio Layer 3 MP3 (.mp3)
Parametre	`height` Høyde på quicktimespilleren. `width` Bredde på quicktimespilleren. `autoplay` Definerer om avspillingen skal skje automatisk. `controller` Definerer om spilleren skal vise kontrollerknapper.
Beskrivelse	Plugin-et bygger opp et HTML `object` element som kaller Quicktime plugin-et til nettleseren. Elementet kan skreddersys ved hjelp av parametrene som sendes inn til plugin-et.
Utviklingsmetode	Bygger opp HTML elementer ved hjelp av JDOM
Externe bibliotek	JDOM
Filer plugin-et genererer	Ingen
Filer plugin-et er avhengig av	Ingen
Eventuell browserproblematikk	Internet Explorer IE benytter seg av den ustandardiserte attributten `classid` i `object` elementet for å kalle korrekt avspillings plugin i nettleseren. Problemet er at dette attributten ikke er standard og dermed ikke taes hensyn til av de andre nettleserne. Løsningen blir å nøste et nytt `object` element inne i det første slik som eksempelet viser. Reglene er slik at hvis en nettleser ikke klarer å tolke det ytterste `object` elementet, vil den prøve å tolke `object` elementet på nivå to, altså det nøstete `object` elementet. Siden det bare er IE som benytter seg av attributtet classid vil alle andre nettlesere tolke `object` elementet på nivå to. Her brukes det standardiserte attributten `type`, som angir mime-typen til filen, for å kalle korrekt avspillingsplugin i nettleseren. Det ytterste `object` elementet blir tolket av IE. Det innerste `object` elementet blir pakket inn i IE sin `conditional comment.` Dette gjemmer elementet for IE nettlesere.
Eksempel	`<object classid="clsid:02bf25d5-8c17-4b23-bc80-d3488abddc6b" codebase="http://www.apple.com/qtactivex/qtplugin.cab" height="16" width="90">` `<param name="src" value="<<URL>>/longestpath.mp3">`

	<param name="controller" value="true"> <param name="autoplay" value="false"> <!--[if !IE]>--> <object data="<<URL>> longest-path.mp3" height="16" type="audio/mpeg" width="90"> <param name="controller" value="true"> <param name="autoplay" value="false"> </object> <!--<![endif]--> </object>
Skjermbilde	

Plugin 1: Plugin som genererer HTML for avspilling av MP3 musikkfiler

MusicPlayerWithPlayListPlugin

Egenskaper	**Verdi**	
Kategori	Komplekse plugins	
Grensesnitt	EntityListInputPlugin	
Ansvarsområde	Generere HTML for avspilling av MP3 filer som ligger i en spilleliste.	
Filtyper/ Mediatyper	MPEG-1 Audio Layer 3 MP3 (.mp3)	

Parametre	`height`
	`width`
	`imageHeight`
	`imageWidth`
	`title`
	`playlistTitle`
	`playlistWidth`
	`playlistImageWidth`
	`playlistImageHeight`
	`playerHeader`
	`autoplay`
	`controller`
		Høyden på Quiktime spilleren.
		Bredden på Quicktime spilleren.
		Høyden på bildet som vises over spilleren.
		Bredden på bildet som vises over spilleren.
		Tittelen til plugin-et.
		Tittelen til spillelisten.
		Bredden til spillelisten.
		Bredden til bildet som vises i spillelisten.
		Høyden til bildet som vises i spillelisten.
		Ledetekst til spilleren.
		Definerer om spilleren skal starte avspilling automatisk.
		Definer om kontroller knapper skal vises på spilleren.

Beskrivelse	Plugin-et får inn en liste av like elementer fra innholdsdokumentet som gjenspeiler en musikkspilleliste. Denne listen blir presentert i et HTML tabellelement. Listen er utformet som linker og aktiverer en bestemt sang i en Quicktime spiller når det trykkes på linken til denne sangen.	
Utviklingsmetode	Bygger opp HTML elementer ved hjelp av JDOM.	
Externe bibliotek	JDOM	
Filer plugin-et genererer	Ingen	
Filer plugin-et er avhengig av	Ingen	
Eventuell browserproblematikk	Dette plugin-et har samme problem som AudioMp3Plugin. Se browserproblematikken i Plugin 1	
Eksempel	`<table cellpadding="0" cellspacing="0">` `<caption>`	

```
    <h1>DPG 2 Media Player</h1>
  </caption>
  <tr>
    <th>Naa avspilles: Good old Harold</th><th></th>
  </tr>
  <tr>
    <td valign="bottom">
        <table cellpadding="0" cellspacing="0">
          <tr>
            <td><img src="<<URL>> logo.jpg" ... >
            </td>
          </tr>
          <tr>
            <td valign="bottom">
             <object
               <<Object element for Quicktime
                  musikkspiller se Plugin 1>>
             </object>
            </td>
          </tr>
        </table>
      </td>
      <td valign="top">
         <div style="border:1px  solid; width:220px;
              height:216px; overflow:auto;">
           <table border="1" rules="rows" width="200">
              <caption>
                 <h2>Spilleliste</h2>
              </caption>
              <tr>
                <td><a href=...>
                     <img ... >
                       <<bildelink til sangen>>
                    </a>
                </td>
                <td><a href=...>
                       <<tekstlink til sangen>> </a>
                </td>
              </tr>
              ... <<Flere spillelisteelementer>>
```

Skjermbilde

Plugin 2: Plugin som genererer HTML for avspilling av MP3 filer som ligger i en spilleliste.

Her følger en presentasjon av plugins som dekker kravet om å rendre HTML for å vise bilder.

ImageMediaPlugin

Egenskaper	Verdi
Kategori	Primitive plugins
Grensesnitt	SingleFieldInputPlugin
Ansvarsområde	Generere HTML for fremvisning av et bilde.
Filtyper/Mediatyper	Joint Photographic Experts Group JPEG (.jpg .jpeg) Portable Network Graphics PNG (.png) Graphics Interchange Format GIF (.gif)
Parametre	`height` Høyden til bildet. `width` Bredden til bildet. `tagClass` Klassen til HTML elementet img. `align` Justering av bildet i HTML containeren.
Beskrivelse	Plugin-et bygger opp et HTML img element. Elementet kan skreddersys ved hjelp av parametrene som sendes inn til plugin-et.
Utviklingsmetode	Bygger opp HTML elementer ved hjelp av JDOM.
Externe bibliotek	JDOM
Filer plugin-et genererer	Ingen
Filer plugin-et er avhengig av	Ingen
Eventuell browserproblematikk	Ingen
Eksempel	`<img class="align_left" width="150" height="270" src=" <<sti>>liverpool.jpg" >`
Skjermbilde	

Plugin 3: Plugin som genererer HTML for fremvisning av et bilde.

PictureGalleryClassicPlugin

Egenskaper	Verdi
Kategori	Komplekse plugins
Grensesnitt	EntityListInputPlugin
Ansvarsområde	Generere HTML for fremvisning av bilder i et bildegalleri.
Filtyper/Mediatyper	Joint Photographic Experts Group JPEG (.jpg .jpeg) Portable Network Graphics PNG (.png) Graphics Interchange Format GIF (.gif)
Parametre	`thumbnailHeight` Høyde til miniatyrbildene til bildegalleriet.

	`thumbnailWidth`	Bredde til miniatyrbildene til bildegalleriet.
	`title`	Tittel til bildegalleriet.
	`numberOfPicturesInRow`	Antall miniatyrbilder som skal vises per rad i bildegalleriet.
Beskrivelse	Plugin-et bygger opp en HTML tabell, der hver av cellene inneholder et miniatyrbilde med en link til det orginale bildet, en tittel til bildet og en beskrivelse. Forskjellen fra ThickboxPictureGalleryJQureyPlugin er at det ikke er benyttet ekstra stilark og javascript for å bedre på utseende. Trykkes det på et bilde åpnes et nytt vindu i nettleseren som viser bildet.	
Utviklingsmetode	Bygger opp HTML elementer ved hjelp av JDOM.	
Externe bibliotek	JDOM	
Filer plugin-et genererer	Ingen	
Filer plugin-et er avhengig av	Ingen	
Eventuell browserproblematikk	Ingen	

Eksempel	<table ...>
	  <caption>
	   Course Pattern Gallery
	  </caption>
	  <tr>
	  <td>
	   liverpool
	  nice
	  <a href=<<URL>>/logo.jpg" ...>
	  <img height="80" src="<<URL>> logo.jpg"
	  width="100" ...>
	
	  </td>
	  ... <<Flere bilder>>
	  </tr>
	  ... <<Flere rader som inneholder bilder>>
	</table>	

Skjermbilde		

Plugin 4: Plugin som genererer HTML for fremvisning av bilder i et bildegalleri.

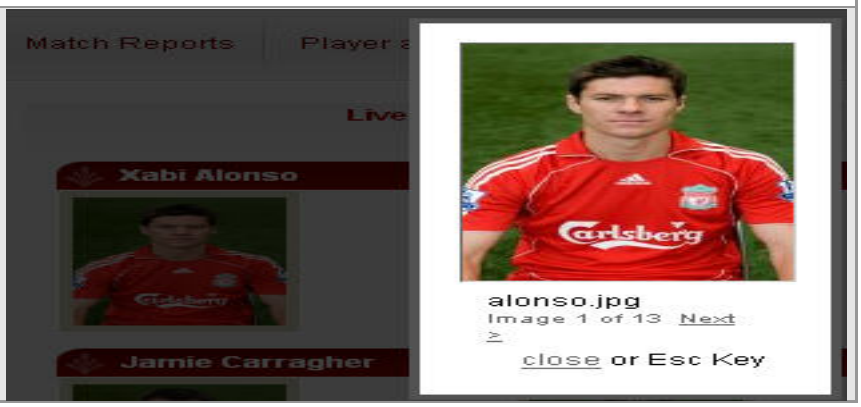
ThickboxFilterTypePictureGalleryJQueryPlugin og ThickboxPictureGalleryJQureyPlugin

Egenskaper	Verdi
Kategori	Komplekse plugins
Grensesnitt	EntityListInputPlugin
Ansvarsområde	Generere HTML for fremvisning av bilder i et bildegalleri
Filtyper/Mediatyper	Joint Photographic Experts Group JPEG (.jpg .jpeg) Portable Network Graphics PNG (.png) Graphics Interchange Format GIF (.gif)
Parametre	`thumbnailHeight` Høyden til miniatyrbildene til bildegalleriet. `thumbnailWidth` Bredden til miniatyrbildene til bildegalleriet. `Title` Tittel til bildegalleriet. `numberOfPicturesInRow` Antall miniatyrbilder som skal vises per rad i bildegalleriet. `cssName` Navn på CSS stilark som skal benyttes til å style bildegalleriet. `tableBorder` Størrelsen på rammen til HTML tabellen som representerer bildegalleriet.
Beskrivelse	Plugin-et bygger opp en HTML tabell, der hver av cellene inneholder et miniatyrbilde med en link til det orginale bildet og en tittel til bildet. Plugin-et setter også inn et javascript og et CSS stilark som hjelper til å bedre utseendet til bildegalleriet. Forskjellen mellom ThickboxFilterTypePictureGalleryJQueryPlugin og ThickboxPictureGalleryJQureyPlugin er at ThickboxFilterTypePictureGalleryJQueryPlugin tar hensyn til et felt ved navn `type`. Plugin-et lager en HTML tabell som inneholder bilder for hver verdi i som ligger i `type` feltet. På den måten kan man legge bilder fra et innholdsdokument inn i kategorier. XSLT transformasjonsarket kan da velge ut hvilken kategori som skal vises på en bestemt side. Et eksempel kan være en avis som har samlet alle sine bilder i et innholdsdokument. Men på sportsiden vil man bare ha sportsbilder i galleriet. Da bruker man ThickboxFilterTypePictureGalleryJQueryPlugin-et for å lage tabell med bilder for hver kategori, og så får XSLT transformasjonsarket ansvaret for å vise den korrekte tabellen med bilder på hver enkelt side.
Utviklingsmetode	Bygger opp HTML elementer ved hjelp av JDOM
Externe bibliotek	JDOM
Filer plugin-et genererer	Ingen
Filer plugin-et er avhengig av	thickbox.css, thickbox.js og jquery-1.2.3.js
Eventuell browserproblematikk	Ingen

Eksempel	`<div>`
	`<script language="javascript" src="thickbox.js" type="text/javascript"></script>`
	`<link href="thickbox.css" media="screen" rel="stylesheet" type="text/css">`
	`<table align="center" border="0">`
	`<caption>`

```
            <h2>Picture Gallery</h2>
         </caption>
         <tr>
            <td valign="bottom">
               <h3>Xabi Alonso</h3>
               <a class="thickbox" href="alonso.jpg"
                  rel="gallery" title="alonso.jpg">
                 <img alt="alonso.jpg" height="100"
                      hspace="2" src="alonso.jpg"
                      title="click to view" width="80">
               </a>
            </td>
            ...//td elementer
         ... //tr elementer

      </table>
</div>
```

Skjermbilde



Plugin 5: To plugins som benytter seg av eksterne Javascript og CSS-filer for å vise bilder i et bildegalleri

Video plugins

Her følger en presentasjon av plugins som dekker kravet om å rendre HTML for å vise video.

FlashYoutubeVideoPlugin

Egenskaper	Verdi
Kategori	Primitive plugins
Grensesnitt	SingleFieldInputPlugin
Ansvarsområde	Generere HTML for fremvisning av SWF (Shockwave Flash) spilleren til nettstedet www.youtube.com.
Filtyper/ Mediatyper	Id streng som peker på en video som er lastet opp til nettstedet www.youtube.com
Parametre	`height` Høyden på flash spilleren `width` Bredden på flash spilleren
Beskrivelse	Plugin-et bygger opp et HTML object element som kaller Shockwave Flash mediaspilleren til nettstedet www.youtube.com. Navnet på videoen hentes fra innholdsdokumentet og hektes på en URL til nettstedet.
Utviklingsmetode	Bygger opp HTML elementer ved hjelp av JDOM
Externe bibliotek	JDOM
Filer plugin-et genererer	Ingen

Filer plugin-et er avhengig av	Ingen
Eventuell browserproblematikk	Ingen
Eksempel	`<object data="http://www.youtube.com/v/RIXKKoBnnYc" height="256" type="application/x-shockwave-flash" width="320"> <param name="movie" value="http://www.youtube.com/v/RIXKKoBnnYc"> </object>`
Skjermbilde	

Plugin 6: Plugin som genererer HTML for fremvisning av SWF (Shockwave Flash) spilleren til nettstedet www.youtube.com.

VideoMediaPlugin

Egenskaper	**Verdi**
Kategori	Primitive plugins
Grensesnitt	SingleFieldInputPlugin
Ansvarsområde	Generere HTML for fremvisning av videoer.
Filtyper/Mediatyper	Quicktime (.mov) Audio Video Interleave AVI (.avi) MP4 (.mp4) 3GP (.3GP) Moving Picture Experts Group Phase 1 MPEG-1 (.mpg) Windows Media Video WMV (.wmv) Flash Video (.flv)
Parametre	`height` Høyden til mediaspilleren. `width` Bredde til mediaspilleren. `autoplay` Angir om filmen skal starte automatisk. `controller` Angir om spilleren skal vise kontrollknapper. `displayWidth` Definerer størrelsen på vinduet hvor filmen vises. Kun i bruk hvis filformatet er Flash Video. `bgcolor` Definerer bakgrunnsfargen til flash mediaspilleren.
Beskrivelse	Plugin-et bygger opp HTML elementer for fremvisning av video med formatene som er nevt ovenfor. Utfra filekstensjonen velger plugin-et hva som må rendres for at videoen skal kunne spilles av i nettleseren. For eksempel ved visning av en Quicktime film, må plugin-et generere HTML

	som nettleseren tolker slik at den velger sitt eget quicktime plugin for avspilling av filmen. Er det derimot en Flash Video fil må plugin-et generere et object element som refererer til en bestemt flash video avspiller, som i dette tilfellet er mediaplayer.swf.
Utviklingsmetode	Bygger opp HTML elementer ved hjelp av JDOM.
Externe bibliotek	JDOM
Filer plugin-et genererer	Ingen
Filer plugin-et er avhengig av	mediaplayer.swf [49]
Eventuell browserproblematikk	Plugin-et refererer til tre forskjellige mediaspillere avhengig av filformatet på filmen: • Nettleserens Quicktime plugin. Her finner man samme browserproblematikk som for AudioMp3Plugin(se Plugin 1) . • Nettleserens Windows Media Player plugin. Microsoft vil at man skal bruke attributten `classid` i `object` elementet når man skal benytte seg av dette plugin-et. Men siden dette attributten ikke er standard vil andre nettlesere enn Microsoft Explorer ignorere object elementet når dette attributten brukes. Løsningen blir at man bygger opp et standard object element med `type` attributten lik mime typen til Windows Media File (`type="video/x-ms-wmv"`). På denne måten vil alle nettlesere som følger standarden spille av videoen korrekt. Inne i object elementet bygger man opp en link til videoen. Når Microsoft Explorer ser denne linken til en Windows Media File, vil nettleseren spille av filen med Windows Media Player plugin-et. • JW FLV MEDIA PLAYER 4.0 [49] .Her definerer man et HTML `object` element som rot element. Inne i `object` elementet defineres et HTML `embed` element. Grunnen til dette er at man vil at videospilleren skal vises i ulike nettlesere. `Object` elementet sørger for at Internet Explorer viser videoen, mens `embed` elementet er nødvendig for at andre nettlesere skal kunne vise videoen. I tillegg til å referere til spilleren må man også angi videoen som skal spilles av. Dette gjøres ved å opprette et `param` element ved navn `flashvars` inne i `object` elementet. Stien til video og skjermstørrelsen ligger i `value` attributten til dette elementet.

Eksempel	Video format:
	• Quicktime. I eksempelet i Plugin 1 vises hvordan man skriver HTML kode for å benytte seg av Quicktime plugin-et til avspilling av MP3 filer. Forskjellen når man skal benytte Quicktimeplugin-et til avspilling av videofiler er i hovedsak mime typen. Den er for video lik `video/quicktime` .
	• WindowsMediaPlayer.
	`<object data="<<URL>>/bailey.mpg"`
	`height="300" type="video/x-ms-wmv"`
	`width="400">`
	`<param name="src"`
	`value="<<URL>>/bailey.mpg">`

```
        <param name="autostart"
           value="false">
        <param name="controller"
           value="true">
        <param name="qtsrcdontusebrowser"
           value="true">
        <param name="enablejavascript"
           value="true">
        <a href="<<URL>>
           bailey.mpg">
        </a>
      </object>
```

- JW FLV MEDIA PLAYER.

```
<object data="<<URL>>/
              mediaplayer.swf"
        flashvars="file=<<URL>>
        liverpool_history.flv&amp;
        displaywidth=400&amp;
        backcolor=0xEEEEEE "
        height="300"
        type="application/x-shockwave-
             flash"
        width="400">
        <param name="movie"
        value="<<URL>>/
              mediaplayer.swf">
        <param name="flashvars"
              value="file=<<URL>>/
            liverpool_history.flv&amp;
            displaywidth=400&amp;
            backcolor=0xEEEEEE ">
        <param name="bgcolor"
              value="#E6E6E6">
        <embed
         flashvars="file=<<URL>>/
         liverpool_history.flv&amp;
         displaywidth=null"
         height="300"
         src="<<URL>>/
         resourceId=mediaplayer.swf"
         type="application/x-
              shockwave-flash"
         width="400">
        </embed>
    </object>
```

Skjermbilde	 Skjermbildet viser spilleren som plugin-et velger når videoen er en Flash Video (.flv) fil

Plugin 7: Plugin som genererer HTML for fremvisning av videoer.

VideoPlayerWithPlayListPlugin

Egenskaper	**Verdi**	
Kategori	Komplekse plugins	
Grensesnitt	EntityListInputPlugin	
Ansvarsområde	Generere HTML for fremvisning av en videospiller med en spilleliste.	
Filtyper/Mediatyper	Quicktime (.mov) Audio Video Interleave AVI (.avi) MP4 (.mp4) 3GP (.3GP) Moving Picture Experts Group Phase 1 MPEG-1 (.mpg) Windows Media Video WMV (.wmv) Flash Video (.flv)	
Parametre	`height`	Høyden til mediaspilleren.
	`width`	Bredden til mediaspilleren.
	`autoplay`	Angir om filmen skal starte automatisk.
	`controller`	Angir om spilleren skal vise kontrollknapper
	`title`	Angir den overordnede tittelen
	`playlistTitle`	Angir tittel til spillelisten
	`playlistWidth`	Angir bredden på spillelisten
	`playlistImageWidth`	Angir bredden på spillelistebildet.
	`playlistImageHeight`	Angir høyden på spillelistebildet
	`playerHeader`	Angir ledetekst over videospilleren
Beskrivelse	Plugin-et genererer en spilleliste og refererer til en passende videospiller som spiller av den valgte videoen fra spillelisten. HTML-koden for å kalle korrekt videospiller er den samme koden som VideoMediaPlugin (se Plugin 7) benytter seg av. Spillelisten bygges ved hjelp av en HTML tabell og inneholder linker som peker på bestemte elementer i listen fra innholdsdokumenet som plugin-et får inn som parameter.	
Utviklingsmetode	Bygger opp HTML elementer ved hjelp av JDOM.	
Externe bibliotek	JDOM	
Filer plugin-et genererer	Ingen	

Filer plugin-et er avhengig av	mediaplayer.swf [49]
Eventuell browserproblematikk	Browserproblematikken er den samme som for VideoMediaPlugin (Se Plugin 7) Årsaken til dette er at koden for å referere til en mediaspiller er helt lik i begge plugins-ene.
Eksempel	<table cellpadding="0" cellspacing="0"> <caption> Dette er en spiller som viser apple filmer </caption> <tr> <th>Naa avspilles: apple</th> </tr> <tr> <td valign="bottom"> <object><<Object element som referer til en bestemt mediaspiller (Se Plugin 7 for å se koden)>> </object> </td> <td valign="top"> <table border="1" rules="rows" width="150"> <caption> playlist </caption> <tr> <td> <a href="<<Bildelink i spillelisten>>” </td> <td> <a href="<<Tekstlink i spillelisten>> "> apple </td> </tr> ...<<Flere spillelisteelementer>> </table> </td> </tr> </table>

Skjermbilde	

Plugin 8: Plugin som genererer HTML for fremvisning av en videospiller med en spilleliste.

XSPFMediaPlayerPlugin

Egenskaper	Verdi
Kategori	Komplekse plugins
Grensesnitt	EntityListInputPlugin
Ansvarsområde	Generere HTML for fremvisning av en videospiller med en spilleliste.
Filtyper/Mediatyper	Flash Video (.flv) MPEG-1 Audio Layer 3 MP3 (.mp3)
Parametre	`height` Høyden til mediaspilleren. `width` Bredden til mediaspilleren. `displayWidth` Bredden på vinduet som viser videoen. `bgcolor` Angir bakgrunnsfargen til mediaspilleren.

Beskrivelse	Plugin-et genererer en spilleliste og refererer til mediaplayer.swf [49] ,ved hjelp av object elementet i HTML, for avspilling av mediaelementene i spillelisten. Spillelisten genereres som et XML dokument og lagres i en XML fil i persistenslaget. Mediaspilleren får så en referanse til denne XML filen. Når det forekommer en forespørsel til nettleseren om å vise mediaspilleren, parser mediaspilleren XML spillelisten og bygger opp en spilleliste.
	Formatet på spillelisten følger standarden XSPF, som er et XML format for deling av spillelister. [50]. Her er et eksempel:
	`<playlist version="1" xmlns="http://xspf.org/ns/0/">`
	`  <title>Tittel </title>`
	`  <info>http:/xspf.org/xspf-v1.html</info>`
	`  <trackList>`
	`  <track>`
	`  <title>Tittel på mediaobjektet</title>`
	`  <location>URL til mediaobjektet</location>`
	`  <image>URL til bilde som skal vises i spilleren</image>`
	`  </track>`
	`  ... Et track element for hvert mediaobjekt som skal vises i spillelisten`
	`  </trackList>`
	`</playlist>`

Utviklingsmetode	Bygger opp HTML elementer ved hjelp av JDOM.
Externe bibliotek	JDOM
Filer plugin-et genererer	*_playlist.xml
Filer plugin-et er avhengig av	mediaplayer.swf [49]
Eventuell browserproblematikk	Mediaspilleren som brukes i dette plugin-et er den samme som benyttes som flash mediaspiller i VideoMediaPlugin. Se **Error! Reference source not found.** for browserproblematikk

Eksempel	`<object data="<<URL>>/ mediaplayer.swf"`
	`flashvars="file=<<URL>>/videoList_playlist.xml`
	`&displaywidth=400&backcolor=0xEEEEEE"`
	`height="300"`
	`type="application/x-shockwave-flash"`
	`width="610">`
	`<param name="movie"`
	`value="<<URL>>/mediaplayer.swf">`
	`<param name="flashvars"`
	`value="file=<<URL>>/videoList_playlist.xml&`
	`displaywidth=400&backcolor=0xEEEEEE`
	
	`<embed allowscriptaccess="always"`
	
	`flashvars="file=<<URL>>/videoList_playlist.xml&`
	`displaywidth=400&backcolor=0xEEEEEE"`
	`height="300" src="<<URL>>/mediaplayer.swf"`
	`type="application/x-shockwave-flash"`
	`width="610">`
	`</embed>`
	`</object>`

Skjermbilde	

Plugin 9: Plugin som genererer HTML for fremvisning av en videospiller med en spilleliste ved bruk av en ekstern videospiller utviklet i flash.

Her følger en presentasjon av et plugin som dekker kravet om rendre HTML for å vise flash animasjoner.

FlashMediaPlugin

Egenskaper	Verdi
Kategori	Primitive plugins
Grensesnitt	SingleFieldInputPlugin
Ansvarsområde	Generere HTML for fremvisning av SWF (Shockwave Flash) filer. Dette er et filformat for multimedia som er utviklet av Macromedia. [51]
Filtyper/ Mediatyper	Shockwave Flash (.swf)
Parametre	`height` Høyden på flash spilleren `width` Bredden på flash spilleren
Beskrivelse	Plugin-et bygger opp et HTML object element som kaller en mediaspiller med formatet SWF. Navnet på spilleren hentes fra innholdsdokumentet og hektes på en URL til persistenslaget hvor spilleren befinner seg.
Utviklingsmetode	Bygger opp HTML elementer ved hjelp av JDOM
Externe bibliotek	JDOM
Filer plugin-et genererer	Ingen
Filer plugin-et er avhengig av	Ingen
Eventuell browserproblematikk	Ingen

Eksempel	`<object data="<URL>/penalty_game.swf"`
	`height="400"`
	`type="application/x-shockwave-flash"`
	`width="583">`
	`<param name="movie" value="<URL>/penalty_game.swf">`
	`</object>`

Skjermbilde	

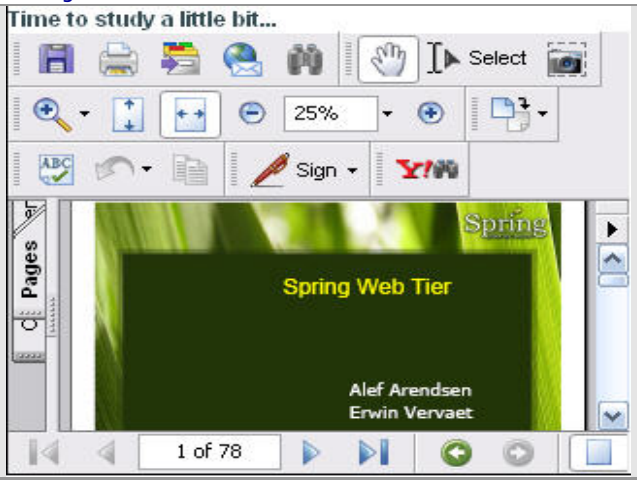
Plugin 10: Plugin som genererer HTML for fremvisning av SWF (Shockwave Flash) filer

Her følger en presentasjon av et plugin som dekker kravet om rendre HTML for å vise en grafisk presentasjon av PDF filer.

PdfPlugin

Egenskaper	Verdi
Kategori	Primitive plugins
Grensesnitt	SingleFieldInputPlugin
Ansvarsområde	Generere HTML for fremvisning av PDF filer.
Filtyper/Mediatyper	Portable Document Format PDF (.pdf)
Parametre	`height` Høyden til PDF plugin-et `width` Bredden til PDF plugin-et
Beskrivelse	Plugin-et bygger opp et HTML `object` element som igjen har ansvar å kalle PDF plugin-et til nettleseren. Hvis det under noen omstendigheter ikke finnes et slikt plugin, eller nettleseren ikke har en oppdatert versjon av plugin-et, vises en link til PDF dokumentet.
Utviklingsmetode	Bygger opp HTML elementer ved hjelp av JDOM.
Externe bibliotek	JDOM
Filer plugin-et genererer	Ingen
Filer plugin-et er avhengig av	Ingen
Eventuell browserproblematikk	Firefox må minimum ha `adobe reader` plugin i versjon 7.08. Firefox vil terminere hvis det vises flere PDF dokumenter på samme side.

Eksempel	`<object`
	`data="<<URL>>/Springinthewebtier.pdf"`
	`height="256"`
	`type="application/pdf" width="320">`
	`<a href="<<URL>>Springinthewebtier.pdf">`
	`Springinthewebtier.pdf`
	`</a>`
	`</object>`

Skjermbilde	

Plugin 11: Plugin som genererer HTML for fremvisning av PDF filer.

Her følger en presentasjon av et plugin som dekker kravet om rendre HTML for å vise kart.

AardvarkMapPlugin

Egenskaper	Verdi
Kategori	Primitive plugins
Grensesnitt	SingleFieldInputPlugin
Ansvarsområde	Generere HTML for fremvisning av et kart.
Filtyper/Mediatyper	Id streng som peker på et kart som er generert av nettstedet [52].
Parametre	`height` Høyden til HTML elementet `iFrame`. `width` Bredden til HTML elementet `iFrame`. `frameborder` Tykkelsen på rammen rundt `iFrame` elementet. `scrolling` Angir om det skal vises Rullegardin eller ei i `iframe` Elementet. `marginwidth` Definerer avstand til høyre og venstre fra `iFrame` elementet mot andre elementer. `marginheight` Definerer avstand til topp og bunn fra `iFrame` elementet mot andre elementer.
Beskrivelse	Plugin-et bygger opp et HTML `iFrame` element hvor kilden er en URL til plasseringen av kartet. Id strengen som identifiserer et bestemt kart hentes fra innholdsdokumentet og erstatter * i basis URL-en http://www.aardvarkmap.net/mapi/*.
Utviklingsmetode	Bygger opp HTML elementer ved hjelp av JDOM.
Externe bibliotek	JDOM
Filer plugin-et genererer	Ingen
Filer plugin-et er avhengig av	Ingen
Eventuell browserproblematikk	Ingen
Eksempel	`<iframe frameborder="0" height="300" marginheight="0" marginwidth="0" scrolling="auto" src="http://www.aardvarkmap.net/mapi/YFIMHCFI" width="413"></iframe>`

Skjermbilde	

Plugin 12: Plugin som genererer HTML for fremvisning av et kart.

Java til HTML plugins

Her følger en presentasjon av et plugin som dekker kravet om rendre HTML for å vise javakode med syntaksmerking i nettleseren.

JavaToHtmlPlugin

Egenskaper	Verdi
Kategori	Komplekse plugins
Grensesnitt	SingleFieldInputPlugin
Ansvarsområde	Generere HTML for fremvisning av javakode i nettleseren med syntaksmerking. Med andre ord, vise javakode slik som en javaeditor vil vise den.
Filtyper/ Mediatyper	Java fil (.java)
Parametre	`height` Høyden på HTML rotkontaineren `width` Bredden på HTML rotkontaineren `class` Klassen til HTML rotkontaineren
Beskrivelse	Plugin-et prosesserer en javafil og bygger opp HTML elementer med en stil som samsvarer med java syntakstmerking
Utviklingsmetode	Bygger opp HTML elementer i en streng og pakker dette inn i et JDOM element.
Externe bibliotek	JDOM, java2html [42]
Filer plugin-et genererer	Ingen
Filer plugin-et er avhengig av	Ingen
Eventuell browserproblematikk	Ingen

Eksempel	`<!-- = Java Sourcecode to HTML automatically converted code = -->`
	`<!-- = Java2Html Converter 5.0 [2006-02-26] = -->`
	`<div align="left" class="java">`
	`<table border="0" cellpadding="3" cellspacing="0" bgcolor="#ffffff">`
	`<tr>`
	`<!-- start source code -->`

```
        <td nowrap="nowrap" valign="top"
            align="left">
           <code>
             <font color="#7f0055">
                <b>interface </b>
             </font>
             <font color="#000000">
                ViktigeKonstanter 
             </font>
             <font color="#000000">
                {
              </font><br />
             ...
   </table>
</div>
```

Skjermbilde

```
interface ViktigeKonstanter {
int AVSLUTT = -1;
int FORTSETT = 1;
}

class ViktigKlasse implements ViktigeKonstanter {
public int svar() {
if (ferdig())
return AVSLUTT; // Kontraktnavn ikke nødvendig pga implements.
else
return FORTSETT;
}
boolean ferdig() { ... }
}
```

Plugin 13: Plugin som genererer HTML for fremvisning av javakode i nettleseren med syntaksmerking.