Støtte for rike klienter i Dynamic Presentation Generator

Peder Lång Skeidsvoll

Institutt for informatikk
Universitetet i Bergen

Lang masteroppgave

Juni 2010

Forord

Dette dokumentet er resultatet av forfatterens mastergrad i Informatikk ved Universitetet i Bergen.

Oppgaven undersøker hvordan støtte for rike klienter kan implementeres i Dynamic Presentation Generator 2.0.

Forfatteren vil gjerne benytte anledningen til å takke dem som har vært med å gjøre oppgaven mulig, og da spesielt veilederene Khalid A. Mughal og Torill Hamre. En stor takk går også til Haakon Nilsen som har sørget for at infrastrukturen har vært i orden.

Videre vil forfatteren takke de andre studentene som har jobbet på JAFU-prosjektet, det vil si Tobias Rusås Olsen, Jostein Bjørge, Ole Henning Vaardal, Øystein Rolland og Morten Høiland.

Sist men ikke minst vil forfatteren takke Julie Elisabeth Risnes for å ha sett over masteroppgaven, og luket bort de verste skrivefeilene.

Peder Lång Skeidsvoll
Bergen, 1. juni 2010

Innhold

Figurer

Tabeller

Kildekode

Forkortelser

AJAX:	Asynchronous JavaScript and XML
API:	Application Programming Interface
CMS:	Content Management System
CSS:	Cascade Style Sheet (Stilark)
DAO:	Data Access Object
DOM:	Document Object Model
DPG:	Dynamic Presentation Generator
FTP:	File Transfer Protocol
HTML:	HyperText Markup Language
HTTP:	HyperText Transfer Protocol
JAFU:	JAva i Fjern Undervisningen
JAR:	Java ARchive
JCR:	Java Content Repository
JPG:	Java Presentation Generator (tidlig versjon av DPG)
JSON:	JavaScript Object Notation
MIME:	Multipurpose Internet Mail Extensions
MVC:	Model View Controller
PCE:	Presentation Content Editor
PICS:	Platform for Internet Content Selection
PM:	Presentation Manager
PV:	Presentation Viewer
RSS:	Really Simple Syndication
SEVU:	Senter for etter- og videreutdanning
SOAP:	Simple Object Access Protocol
StAX:	Streaming API for XML
URL:	Uniform Resource Locator
W3C:	World Wide Web Consortium
WML:	Wireless Markup Language
WYSIWYG:	What You See Is What You Get
XHR:	XMLHttpRequest
XHTML:	Extensible HyperText Markup Language
XML:	Extensible Markup Language
XP:	eXtreme Programming
XSLT:	Extensible Stylesheet Language Transformations

1 Introduksjon

1.1 Innledning

Universitetet i Bergen (UiB) har studenter som studerer via universitetets *Senter for etter- og videreutdanning* (SEVU). Dette er studenter som ikke møter til forelesning og gruppeøvelser, men som tar kursene hjemmefra, såkalt fjernundervisning. I SEVUs studietilbud finner man to begynnerkurs i programmering, *INF100F - Grunnkurs i programmering* og *INF101F - Videregående Programmering*. Fagene administreres av Institutt for informatikk ved UiB.

I fjernundervisningen av disse to fagene ønsket man ulike verktøy for å lette kommunikasjonen mellom fagansvarlige og studenter. Det var blant annet ønskelig at det skulle bli lettere å levere inn studentoppgaver. Det ble derfor opprettet et prosjekt kalt *Java i Fjernundervisningen* (JAFU) ved Institutt for Informatikk.

I 2003 beskrev førsteamanuensis Khalid A. Mughal ved UiB en metode for å skille fremvisning av innhold (*presentasjon*) fra innholdet (*mønsteret*) i artikkelen *Presentation Patterns: Composing Web-based Presentations* [74]. Metoden viste seg å være et godt utgangspunkt for å utvikle verktøy for fjernundervisningen. Masterstudenten Kevin Cruickshanks forsøkte i 2004 å transformere konseptet fra denne artikkelen til et kjørende program, kalt *Java Presentation Generator* (JPG) [19].

Året etter fikk, Yngve Espelid, en annen mastergradsstudent som oppgave å videreutvikle JPG. Selv om JPG fungerte, var det såpass mange problemer med programmet at Espelid valgte å lage et nytt program [21]. Espelids program ble brukt i fjernundervisningen sine

kurs INF100-F og INF101-F hver vår fra 2006 til 2009. Programmet fikk navnet *Dynamic Presentation Generator* (DPG).

DPG var et proof-of-concept, og på grunn av dette ville masterstudentene Karianne Berg [5], Bjørn Ove Ingvaldsen [59] og Bjørn Christian Sebak [80] i 2008 gå grundigere til verks, og lage et mer robust system. De valgte derfor å begynne helt på nytt, og forsøkte å lage et system med fokus på høy kodekvalitet og bruk av kjente teknologier som *Spring* [82] og *JackRabbit* [32]. Spring ble brukt som rammeverk for å oppnå en *MVC*-arkitektur samt *dependency injection*, mens JackRabbit ble brukt til persistens. Den ferdige versjonen ble kalt DPG 2.0.

Denne masteroppgaven bygger på DPG 2.0, og ser på svakheter i det eksisterende systemet, med spesielt fokus på pluginsystemet. Oppgaven vil også undersøke hvordan teknologier som AJAX kan transformere DPG 2.0 til å bli et system med mer dynamisk tilsnitt.

1.2 Mål for oppgaven

1.2.1 Overordnet mål

Det overordnede målet for denne masteroppgaven er å utvide Dynamic Presentation Generator (DPG) 2.0 til å støtte presentasjonsmønstre som benytter rike klienter.

1.2.2 Delmål

Det overordnete målet består av følgende delmål:

1. Refaktorering av arkitekturen til DPG 2.0 for utviklingen av presentasjonsmønstre som støtter rike klienter.
2. Refaktorering av presentasjonsmønsterfikasjonen for å nå hovedmål.
3. Undersøke bruksområder for rike klienter i Kursmønsteret [6].
4. Utvikle klientprogramvare som kan utnytte serverstøtten utviklet i delmål 1.

1.3 Motivasjon

Rike klienter har utviklet seg mye i løpet av de senere årene, og det er ønskelig å la DPGs brukere dra nytte av dette. Ved å utnytte rike klienter er det mulig å utvikle presentasjonsmønstre med en interaktivitet og brukervennlighet som ikke har vært mulig tidligere.

1.4 Utviklingsverkøy

All kode er skrevet i utviklingsmiljøet *Eclipse* [37]. For JavaScript-utvikling er Eclipse-pluginet *JavaScript Development Tools* [38] blitt valgt, mens *Mozilla Firefox* [31] utvidelsen *Firebug* [29] er brukt som *avlusningsverktøy* (eng. *debugging tool*). For å produsere grafiske elementer er *Adobe Photoshop* [50] blitt benyttet. Dette dokumentet er skrevet i LaTeX, mens figurene er tegnet i *Microsoft Visio* [11].

1.5 Utiklingsmetode

Et utvalg av prinsipper fra den smidige (agile) utviklingsmetoden *XP* (*eXtreme Programming*) [68] er brukt i utviklingen av programvaren. Blant annet er *parprogrammering* blitt brukt for å forstå koden til DPG 2.0. Dette prinsippet har også blitt brukt for å implementere de mer kompliserte metodene.

Refaktorering har også vært en svært sentral arbeidsoppgave, da refaktorering er til stor hjelp når man skal sette seg inn i andre programmereres kode. Refaktorering fører også til at koden blir bearbeidet slik at høyere kodekvalitet oppnås.

1.6 Organisering av oppgaven

Oppgaven er delt opp i 9 kapitler, der hvert kapittel tar opp et hovedtema. Det finnes to tillegg (appendikser) bakerst i oppgaven som gir dokumentasjon på hvordan komponentene som er produsert i denne masteroppgaven kan brukes.

Kapittel 2: Bakgrunn og problembeskrivelse

Dette kapittelet presenterer konsepter som er viktige for å forstå oppgaven. Blant annet vil konsepter som presentasjonsmønstre, innholdshåndteringssystemer og DPG bli forklart. I slutten av kapittelet vil dagens situasjon av DPG bli gjennomgått. Det vil videre bli redegjort for hvordan denne oppgaven kan forbedre denne, samt hva som er motivasjonen bak disse inngrepene.

Kapittel 3: Analyse av pluginarkitektur

Plugin er et konsept som Bjørn Ove Ingvaldsen tilførte DPG i sin masteroppgave [59]. I dette kapittelet vil dette konseptet bli analysert. Kapittelet er skrevet i samarbeid med masterstudent Tobias Rusås Olsen [77].

Kapittel 4: Endringer i pluginarkitektur

I dette kapittelet vil endringene som har blitt utført i pluginarkitekturen bli gjennomgått.

Kapittelet er skrevet i samarbeid med masterstudent Tobias Rusås Olsen [77].

Kapittel 5: Ny presentasjonsmønsterspesifikasjon

Presentasjonsmønsterspesifikasjonen som benyttes i DPG har gått gjennom en relativt lang utvikling. Svakheter i spesifikasjonen til DPG 2.0 vil bli gjennomgått, og nye løsninger lansert. Resultatet er presentasjonsmønsterspesifikasjonen til DPG 2.1. Dette kapittelet er skrevet i samarbeid med masterstudent Tobias Rusås Olsen [77].

Kapittel 6: AJAX og AJAX-biblioteker

AJAX er et nytt og spennende konsept. I dette kapittelet vil AJAX bli introdusert og forklart. I tillegg vil det bli foretatt en analyse av tre ulike AJAX-biblioteker.

Kapittel 7: Serverløsning

For at AJAX skal bli interresant er det nødvendig å støtte konsptet på både serveren og klienten. I dette kapittelet vil tre forskjellige forslag for AJAX-støtte i DPG bli gjennomgått. Så vil det bli vist hvordan man kan gi støtte for AJAX ved hjelp av to forskjellige plugins til DPG.

Kapittel 8: Klientsideløsning

Klientsiden er meget interresant fordi den, på en svært konkret måte, vil vise om oppgaven har lyktes i å nå sine mål. I dette kapittelet vil løsninger laget for å omforme klienten fra en passiv til en aktiv klient bli gjennomgått.

Kapittel 9: Evaluering, konklusjon og videre arbeid

I siste kapittel presenteres en konklusjon, samt en liste av idéer for videre arbeid.

2

Bakgrunn og problembeskrivelse

2.1 Bakgrunn

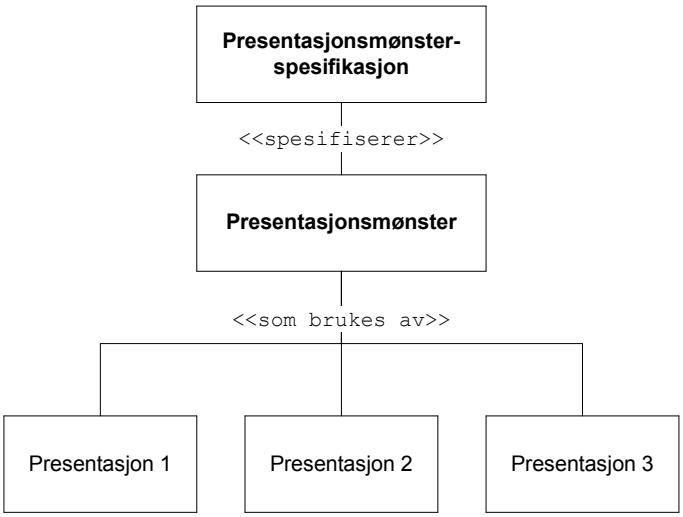
2.1.1 Innholdshåndteringssystem

Et *innholdshåndsteringssystem* (eng. *Content Managment System*), er en samling med prosedyrer som gjør det mulig å lage og organisere innhold. Innhold kan for eksempel være bilder, tekst og video.

Et *internettinnholdssystem* er et dataprogram for å organisere innhold slik at det kan fremvises på internett, og da ofte *World Wide Web*. Et slik system er ofte laget som en internettapplikasjon, det vil si en applikasjon som kjører i nettleseren. En av de viktigste funksjonene til et internettinnholdssystem er at det skal være mulig for mennesker som ikke er datakyndige å publisere innhold. Det vil si at kunnskap om blant annet markeringsspråket HTML ikke skal være påkrevd. For eksempel skal en journalist ha mulighet til å legge inn en nyhetsak i avisens nettavis selv uten kunnskap om hvordan internettsider er bygget opp. Videre vil et slikt system gjøre det enklere å organsiere innholdet, og gjøre det søkbart.

Eksempler på internettinnholdsystem er bloggverktøyet Wordpress [70] og MediaWiki [39], plattformen til nettleksikonet Wikipedia [40].

For mer informasjon om innholdshåndteringssystem, henvises det til [8].



Figur 2.1: Relasjon mellom presentasjonsmønsterspesifikasjon, presentasjonsmønster og presentasjoner.

2.1.2 Presentasjonsmønster

Presentasjonsmønstre og presentasjoner

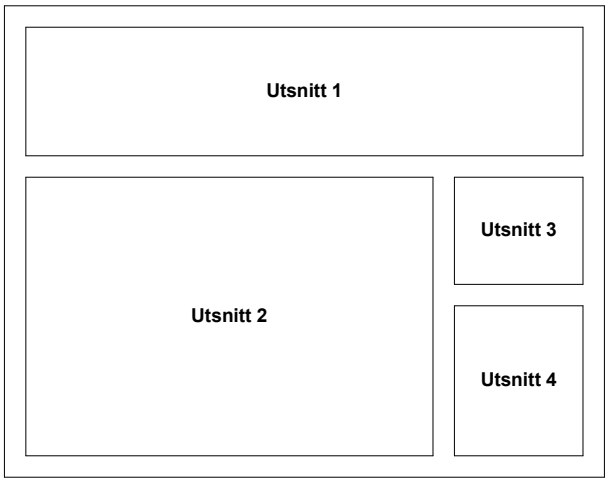
Presentasjonsmønster er, som nevnt i avsnitt 1.1, et konsept som ble lagt frem av Khalid A. Mughal i 2003 [74]. Et presentasjonsmønster spesifiserer hvordan innhold skal *struktureres* og *rendreres*.

Presentasjonsmønstre blir utviklet ut fra *presentasjonsmønsterspesifikasjonen*. Dette er et regelsett som spesifiserer hvordan presentasjonsmønstre syntaktisk blir bygget opp, og hva slags elementer som kan benyttes.

Det kan være flere instanser av et presentasjonsmønster. Disse instansene kalles *presentasjoner*. Dette gjør at det for eksempel er mulig å lage et presentasjonsmønster for fjernundervisning, og ut fra dette lage mange presentasjoner slik at alle fag har sin egen uavhengige instans. Figur 2.1 viser hvordan tre ulike presentasjoner er koblet sammen med et presentasjonsmønster. Dette kan sammenlignes med hvordan en klasse i et objektorientert programmeringsspråk som *Java* og *C++* oppgir struktur og oppførsel for objekter.

Oppbygningen til et presentasjonsmønster

Strukturen til et presentasjonsmønster består av fire hovedelementer: *Entiteter*, *entitetsinstanser*, *utsnitt* og *sider*. Alle disse elementene må defineres i en fil som blir kalt `pattern.xml`. Hvert presentasjonsmønster har sin egen `pattern.xml`-fil.



Figur 2.2: En mal med plass til fire utsnitt

En *entitet* (eng. *entity*) beskriver strukturen til data. En entitet består av et eller flere *felt*. For eksempel kan man definere at et vanlig norsk navn består av et fornavn, mellomnavn og etternavn. Da vil entiteten til et navn bestå av feltene `fornavn`, `mellomnavn` og `etternavn`.

Entitetsinstanser (eng. *entity instance*) er som navnet antyder, instanser av entiteter. Det er mulig å lage flere instanser av samme entitet. Det kan settes en direkte analogi fra dette elementet til `new`-operatoren i objektorienterte språk.

Utsnitt (eng. *views*) er et element som kobler en entitetsinstans sammen med en transformasjon. Dette betyr at entitetsinstansene blir omformet fra den interne strukturen til et format som er mer leselig for andre, for eksempel HTML.

Side (eng. *page*) er det øverste elementet, og brukes til å komponere dokumenter ut fra transformerte utsnitt. Dette gjør den ved hjelp av en *mal* der utsnittene settes inn på definerte plasser. Figur 2.2 viser en mal der det er satt av plass til fire forskjellige utsnitt.

For mer informasjon om presentasjonsmønstre se kapittel 5: *Ny presentasjonsmønsterspesifikasjon*, [6], [59] og [80].

2.1.3 Kursmønsteret

Jostein Bjørge og Peder Lång Skeidsvoll utviklet våren 2009, som et ledd i faget *INF219 - Prosjekt i programmering*, et presentasjonsmønster spesielt vinklet mot fjernundervisningen [6]. Presentasjonsmønsteret, som har fått navnet *Kursmønsteret*, inneholder elementer som ukeplan, pensumliste og meldinger fra fagansvarlig. Dette mønsteret har blitt brukt i fjernundervisningen til Institutt for informatikk våren 2010 for fagene *INF100-F* og *INF101-*

F. I denne oppgaven vil Kursmønsteret bli brukt som *case-study*.

2.1.4 Dynamic Presentation Generator

Dynamic Presentation Generator (DPG) er et internettinnholdssystem utviklet av JAFU-prosjektet ved Institutt for Informatikk ved UiB. Programmet implementerer konseptet om presentasjonsmønstre, og foreligger i skrivende stund i versjon 2.0. For mer informasjon om historien til DPG, se avsnitt 1.1.

Utgaven av DPG som blir utviklet i denne oppgaven har fått versjonsnummer 2.1.

DPG 2.0 er delt opp i fire forskjellige delsystemer. Delsystemene kalles *Presentation Content Editor*, *Presentation Viewer*, *Presentation Manager* og *Lobby*.

I de fire neste avsnittene vil hvert delsystem i DPG bli introdusert.

Presentation Content Editor

Presentation Content Editor (PCE) er delsystemet til DPG som tar seg av redigering av innholdet i en presentasjon. For eksempel er det her brukeren skriver inn en ny melding, eller laster opp et bilde. Figur 2.3 viser hvordan PCE fremstiller en entitet med feltene `header` og `description`.

Presentation Viewer

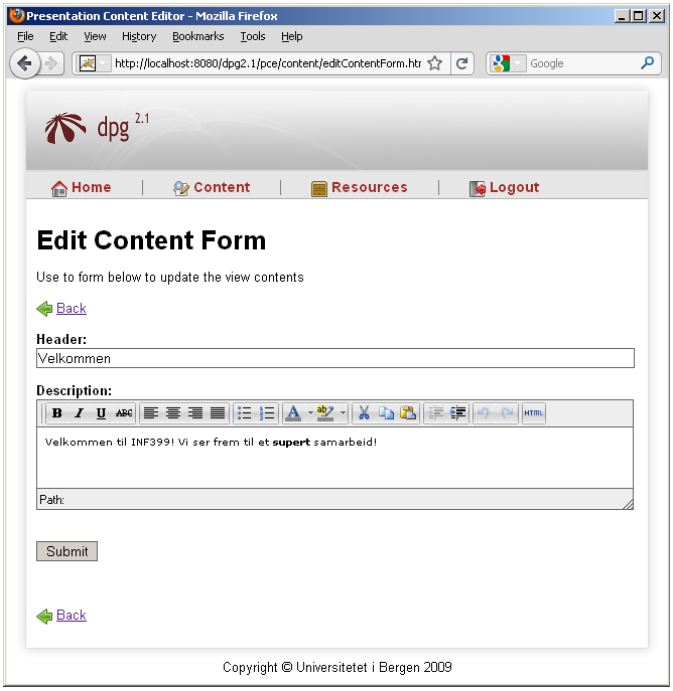
Presentation Viewer (PV) har som oppgave å rendrere presentasjonen for brukeren. PV går gjennom en rekke steg for å komme frem til den ferdige siden. Disse stegene er:

1. Hente ut presentasjonen fra persistenslaget.
2. Transformere utsnittene til for eksempel HTML ved hjelp av transformeringsspråket XSLT [93].
3. Sette sammen de transformerte utsnittene til et ferdig dokument ved hjelp av malmotoren Velocity [33], og sende resultatet tilbake til brukeren.

For mer informasjon om denne prosessen, se [80].

Presentation Manager

Presentation Manager (PM) har som oppgave å konfigurere presentasjoner. For eksempel kan man endre tittelen til en presentasjon i dette delsystemet. PM inneholder også funksjoner for å opprette nye presentasjoner fra eksisterende presentasjonsmønstre, klone andre



Figur 2.3: Et skjermbilde av PCE i bruk.

presentasjoner eller slette presentasjoner. Det er også mulig å utføre andre mer avanserte innstillinger, som å overstyre innstillingene til et presentasjonsmønster.

Lobby

Dette delsystemet brukes til å la brukeren skrive inn brukernavn og passord slik at vedkommende kan logge seg på systemet. I tillegg til dette fungerer det som en inngangsportal for de andre delsystemene, ved for eksempel å liste opp presentasjonene som brukeren har tilgang til.

Brukerroller i DPG

Det finnes tre forskjellige brukerroller i DPG. Rollene er *Reader*, *Publisher* og *Administrator* [66]. Disse brukerrollene har sterk sammenheng med delsystemene til DPG, og forholder seg til hvilke delsystem brukeren har rett til å bruke:

- *Reader* har tilgang til PV. Det vil si at brukere som har denne rollen kun har lov til å se presentasjoner som vedkommende har tilgang til.

- *Publisher* har tilgang til PV og PCE. Denne brukerrollen har i tillegg til å se en presentasjon, også lov å endre innholdet til den gjennom delsystemet PCE.
- *Administrator* har tilgang til PV, PCE og PM. Administrator har altså rettigheter til å gjøre alt en *reader* og en *publisher* har rettigheter til å gjøre. Administrator har i tillegg lov til å opprette nye presentasjoner i delsystemet PM.

Lobby delsystemet har alle brukerrollene tilgang til.

2.1.5 Plugins

Plugins er et relativt vanlig engelsk uttrykk, og vil bli brukt i oppgaven for å være konsekvent med tidligere litteratur om dette temaet. En generell gjennomgang av hva en plugin er vil bli gjennomgått i kapittel 3.

Plugin blir i denne oppgaven brukt i to sammenhenger. I kapittlene 3, 4, 5, 7 og tillegg A vil ordet brukes om eksterne komponenter til DPG, mens det i kapittlene 6, 8 og tillegg B vil brukes om eksterne komponenter til JavaScript biblioteker. Formen til ordet kommer til å være *plugin* for entall, og *plugins* for flertall.

2.1.6 Rike klienter

Rike klienter (eng. *rich clients*), også kalt tykke klienter, betyr i vår sammenheng at klient-programmet tilbyr funksjonalitet uavhengig av serveren. Tradisjonelt sett har all prosessering i en internettapplikasjon foregått på serveren, og nettleseren har fungert som en *tynn klient*, det vil si at den kun har vist resultatet av serverens arbeid. Dette er i ferd med å forandre seg med økende bruk av AJAX, der prosessering også foregår i nettleseren. Les mer om dette i kapittel 6.

2.2 Problembeskrivelse

2.2.1 Rike klienter

Dynamic Presentation Generator (DPG) 2.0 er konstruert for å kommunisere med tynne klienter. Dette vil si at programmet gjør all prossesering selv, før det leverer fra seg ferdig materiale til klienten. Mye har forandret seg siden DPG ble påtenkt, og moderne nettlesere er nå så kraftige at de kan gjøre deler av prosesseringen selv. Ved å bruke dette riktig kan man oppnå nettsider som er mer responsive og brukervennlige. Dette er fordi klienten ikke trenger å kommunisere med serveren ved hver eneste endring, og fordi klienten kan gjøre asynkrone kall til serveren slik at brukeren kan fortsette å bruke nettsiden mens kommunikasjonen pågår.

Det er i denne masteroppgaven ønskelig å utvikle en generell mekanisme for utveksling av data mellom DPG og rike klienter. Dette vil gjøre det enklere å utvikle programvare for klienten som kan visualisere data fra en presentasjon. Med en slik funksjonalitet vil det være mulig å produsere mer dynamiske og smidige presentasjoner enn det som kan gjøres i DPG 2.0.

2.2.2 Pluginarkitektur

DPG 2.0 er utstyrt med en pluginarkitektur [80] som tilbyr svært begrenset funksjonalitet, og som derfor er lite anvendelig. Orginalt er pluginarkitekturen konstruert for å gjøre det lettere å bruke multimedia i presentasjonsmønstre. Det er ønskelig å gjøre plugins til en kraftigere mekanisme.

Med en kraftigere pluginarkitektur er det mulig å utvide funksjonaliteten til DPG uten å øke størrelsen til *kjernen* til DPG. Det vil da for eksempel være mulig å produsere støtte for rike klienter ved hjelp av den nye pluginarkitekturen.

2.2.3 Presentasjonsmønsterspesifikasjon

Presentasjonsmønsterspesifikasjonen til DPG 2.0 har utviklet seg til å bli relativt effektiv. Likevel er det flere punkter der den ikke strekker helt til. Behandlingen av *typer* er ikke optimal, blant annet fordi den skiller mellom innebygde typer og plugins. I tillegg er lister av entiteter definert på en lite elegant måte.

Med en ny og bedre presentasjonsmønsterspesifikasjon vil det bli enklere å produsere nye presentasjonsmønstre. Ved å legge mer til rette for bruk av plugins vil det også føre til at det blir mer aktuelt å både bruke og utvikle nye plugins i DPG.

3

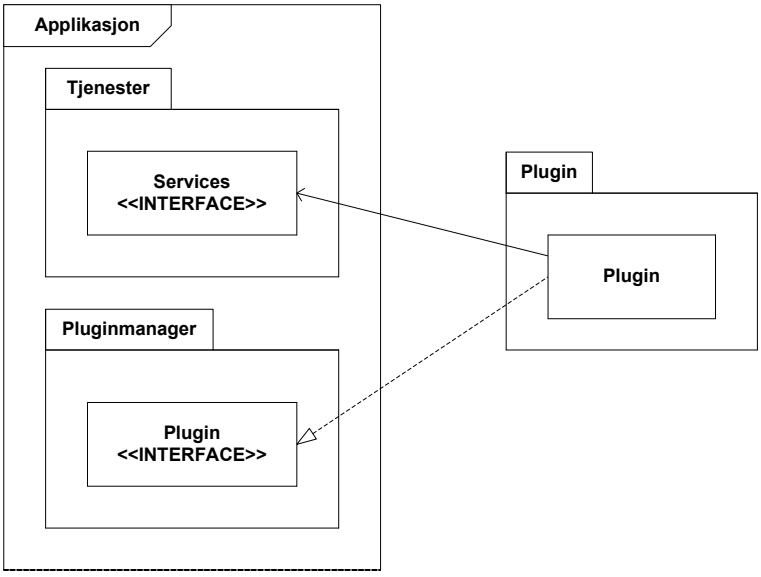
Analyse av pluginarkitektur

Dette kapittelet er skrevet i fellesskap av Tobias Rusås Olsen [77] og Peder Lång Skeidsvoll. Kapittelet inngår i begges masteroppgaver. Årsaken til at dette er et felleskapittel er at begge parter var interessert i å videreutvikle pluginarkitekturen for å kunne gjennomføre målene i sine masteroppgaver som henholdsvis omhandlet bruk av interaktive plugins og AJAX i DPG. Konklusjoner i dette kapittelet er gjort på grunnlag av samtale mellom de to forfatterene. Det er også kommet innspill fra veiledere under ukentlige møter.

3.1 Generelt om pluginarkitektur

Plugins, også kalt *addins*, *add-ons*, *snapins* eller *extensions*, er et stykke programvare som interagerer med en *vertapplikasjon* (eng. *host application*) for å tilby en spesifikk funksjonalitet *på etterspørsel* (eng. *on demand*). Årsaker til at man kan ønske å benytte en pluginarkitektur kan være å:

- Gjøre det mulig for tredjepartsutviklere å utvide applikasjonen.
- Støtte muligheter som ikke er forutsett.
- Redusere størrelsen på vertapplikasjonen.
- Skille kildekoden fra applikasjonen av lisensårsaker.



Figur 3.1: Eksempel på hvordan vertapplikasjon og plugins kommuniserer.

Applikasjonen har ofte en egen pluginmodul som håndterer kommunikasjon mellom plugin og applikasjon.

For at en plugin skal kunne anvendes, må den registreres hos programmet som skal bruke den, den såkalte vertapplikasjon. I tillegg til dette trengs det en protokoll som gjør det mulig at plugin og vertapplikasjonen kan utveksle data. Figur 3.1 viser hvordan de forskjellige komponentene kommuniserer. Vertapplikasjonen skal kunne opptre uavhengig av plugins, og bør helst ikke være avhengig av plugins for å kunne fungere. Dette betyr at applikasjonen ikke bør slutte å fungere dersom en plugin mangler. Ideelt sett skal det være mulig å legge til og endre plugins dynamisk, uten å måtte gjøre endringer i selve vertapplikasjonen.

Et eksempel på bruk av plugins finnes i nettlesere. Nettlesere bruker plugins for å vise for eksempel video og andre presentasjonsformater. Eksempler på dette er Adobe Flash [49], Apple QuickTime [52], Microsoft Silverlight [10].

3.2 Pluginarkitekturen i DPG 2.0

Pluginarkitekturen i DPG 2.0 ble utarbeidet av Bjørn Ove Ingvaldsen [59] i hans masteroppgaveprosjekt ved Institutt for Informatikk, UiB i 2008. Ingvaldsens oppgave var å gi DPG 2.0 mulighet til å behandle multimedia som video og bilder. Til dette formålet konkluderte han med at den mest fornuftige løsningen var å implementere en pluginarkitektur.

Pluginarkitekturen bygger på Martin Fowlers pluginmønster [41]. Dette mønsteret dikterer

at det lages *kontrakter* (eng. *interfaces*) som plugins må implementere. Figur 3.1 viser et diagram over dette mønsteret. I DPG 2.0 finnes det to slike kontrakter:

- `SingleFieldInputPlugin`: Denne kontrakten tillater brukeren å legge inn et felt som inndata til plugin. Dette vises i listing 3.3. Metoden som må implementeres heter `generateTag()` og returnerer et JDOM-element. Eksempler på bruk av denne kontrakten er plugins som håndterer et bilde eller en video.
- `EntityListInputPlugin`: Denne kontrakten tillater brukeren å legge inn en liste med entiteter som inndata. Metoden som må implementeres heter `generateTagWithEntityListInput()`. Den returnerer, som `SingleFieldInputPlugins` metode `generateTag()`, et JDOM-element. Eksempel på bruk er et bildegalleri der brukeren kan navigere mellom flere bilder.

En annen viktig del av pluginarkitekturen er hvordan plugins skal importeres til DPG. Dette er løst ved at utvikleren legger til et nytt element i en XML-filen `plugIn.xml`. Dette vises i listing 3.1. I denne XML-filen er alle plugins som brukes av DPG listet opp med sti til hvor man finner selve implementasjonen til plugins. samt litt annen informasjon. Ved oppstart går DPG gjennom hver plugin som er definert i filen, og importerer dem til systemet slik at de er klar til bruk.

3.3 Legge til plugins i en presentasjon i DPG 2.0

For å legge til en ny plugin må følgende steg gjennomgås:

1. Implementere en plugin.
2. Konfigurere plugin i filen `plugIn.xml`, i presentasjonsmønsteret og i pluginkonfigurasjonsfilen `pluginPattern.xml`.
3. Starte DPG på nytt slik at alle plugins lastes på nytt.

3.3.1 Implementasjon av en ny plugin

Implementasjon av plugins er en enkel prosess. Dette er stegene som må utføres:

- Opprette et nytt Java-prosjekt.
- Legge til JAR-filen `pluginInterfaces.jar` i *byggebanen* (eng. *build path*) til prosjektet. Denne JAR-filen finnes i DPG i `web/WEB-INF`-mappen.
- Opprette en ny klasse som implementerer enten kontrakten `SingleFieldInputPlugin` eller kontrakten `EntityListInputPlugin` - den første for et enkelt felt og den andre for en liste med entiteter. Pluginet må implementere `generateTag()`-metoden.

- I metoden `generateTag()` opprettes det et `Element`-objekt fra JDOM-pakken. Et slikt element representerer en XML-tag, og kan derfor ha tekst og kan tilknyttes attributter eller underelementer. Dette gir stor fleksibilitet i utformingen av innholdet som plugin returnerer.

Når denne prosessen er gjennomført kan pluginet brukes i DPG.

3.3.2 Konfigurering av plugins i DPG 2.0

Gjøre plugins klar for bruk i DPG 2.0

For å bruke plugins i en presentasjon må systemet ha tilgang til pluginet. Dette gjøres ved at pluginutvikleren eksporterer klassen med pluginet som en JAR-fil, og plasserer filen i DPGs `lib`-mappe. Når dette er gjort må utvikleren redigere XML-filen `plugIn.xml`. I denne filen defineres alle plugins som skal benyttes i applikasjonen. For hver plugin defineres *stien* (eng. *path*) til pluginet, navnet på pluginet som brukes i DPG 2.0, hvilken kontrakt som er implementert (`SingleFieldInputPlugin` eller `EntityListInputPlugin`), og hva slags type inndata pluginet krever. Denne inndatatypen kan være `string`, `date`, `entity-list` eller `file`.

Listing 3.1 viser et eksempel på hvordan en plugin er definert. I eksempelet er det definert at navnet på pluginet er `JavaToHTMLPlugin`, den forventer inndata av typen `file`, og at den skal kun ha én inndata ettersom den implementerer kontrakten `SingleFieldInputPlugin`. I feltet `class-name` skrives stien til hvor pluginimplementasjonen finnes.

Listing 3.1: Konfigurasjonsfil for Java-til-HTML-plugin.

```
<plugin>
  <class-name>javaToHtml/JavaToHtmlPlugin.class</class-name>
  <interface>SingleFieldInputPlugin</interface>
  <plugin-name>JavaToHTMLPlugin</plugin-name>
  <formfield-type>file</formfield-type>
</plugin>
```

Legge til plugin i en presentasjon

Det neste steget er å legge til pluginet i en presentasjon. Først går man inn i presentasjonsmønsteret og finner filen `pluginPattern.xml`. Listing 3.2 er et eksempel på hvordan en plugin defineres i denne filen.

Listing 3.2: Konfigurasjon av plugin i pluginPattern.xml.

```
<!-- Her definerer det at pluginet MinPlugin skal kalles minPlugin i -->
<!-- pattern.xml                                                      -->
 <plugin-config id="minPlugin" plugin-name="MinPlugin">
  <!-- Den neste linjen definerer et parameter -->
  <param name="mittParameter">verdi</param>
</plugin-config>
```

I denne filen kan det legges til inndata, kalt *parameterne*, til pluginet. Dersom det arbeides med en bildeplugin kan eksempler på slike parametre være *høyde* og *bredde*. Disse parametrene skiller seg fra inndata til plugins ved at disse parameterene ikke kan endres i delsystemet PCE. Fordelen med å ha parameterkonfigurering her er at en kan ha unike verdier for hver presentasjon. Dette betyr at en kan bruke samme bildeplugin i flere presentasjoner, men likevel ha unik høyde og bredde per bilde per presentasjon.

Deretter må utvikleren inn i filen `pattern.xml` for å knytte plugin opp mot et felt i en entitet. Dette gjøres ved å legge til et nytt felt i en entitet, definere at feltet skal håndteres av en plugin, og deretter si hvilken plugin som skal håndtere feltet. I listing 3.3 vises det et eksempel på en entitet med et felt som håndteres av en plugin og et felt som håndteres av en streng.

Listing 3.3: Definering av et felt som plugin

```
<entity id="testEntity">
  <fields>
    <field type="string">testInnhold</field>
    <field type="plugin" pluginConfig="minPlugin">minPlugin</field>
  </fields>
</entity>
```

Verdien til attributtet `pluginConfig` må være lik navnet som er gitt på en plugin i filen `pluginPattern.xml`.

Det siste steget er å legge til et XSLT-uttrykk i transformasjonsfilen som tilhører utsnittet. I dette tilfellet kan det være:

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

Til slutt må man inn i selve presentasjonen og legge til et nytt element som stemmer overens med det som er definert i XSLT-dokumentet. Elementet må ha en attributt av type `type` som er satt til verdien `plugin` for at man skal kunne legge til inndata til feltet i PCE. Dette feltet burde kunne automatisk genereres hvis det mangler, men i DPG 2.0 må det legges til manuelt. Dette vises under. `pluginConfig` attributtet forteller DPG hva slags plugin som skal være koblet mot feltet.

```
<mittFelt type="plugin" pluginConfig="minPlugin">innhold</mittFelt>
```

3.3.3 Initialisering av plugins

Innlastingen av plugins i DPG 2.0 gjøres ved at systemet går gjennom hver plugin i filen `plugIn.xml`. Ut fra hva som står definert i denne filen, kobler den hvert plugin-navn opp mot rikig plugin-klasse. Klassen må ligge i systemets *byggebane*. Etter dette er gjort kobler DPG seg sammen med pluginet ved hjelp av *refleksjon* (eng. *reflection*) [76].

Denne operasjonen kjøres automatisk ved oppstart av DPG.

3.4 Svakheter og løsninger i nåværende pluginarkitektur

Etter å ha gått gjennom pluginarkitekturene, ble det avdekket en rekke mangler og svakheter slik den ble ferdigstilt i DPG 2.0. Nå vil svakhetene analyseres og forslag til forbedring bli beskrevet. De faktiske implementasjonsendringene er diskutert i kapittel 4.

3.4.1 Bruk av plugins i presentasjonsmønstre

I DPG 2.0 er det to måter å definere datatypen til felter i entiteter:

- Definere at feltet er av en innebygget datatype som `string`, `xhtml` eller `file`.
- Definere at feltet skal bruke `plugin` som datatype.

At det er to forskjellige måter å definere typen til et felt på blir vurdert som en svakhet, siden det kan løses ved kun å bruke én måte. Det er ønskelig at de innebygde datatypene også skal kunne håndteres av en plugin. Da kan definisjonen av datatypen fjernes og en trenger bare å definere hvilken plugin som skal håndtere feltet. Dette vil føre til at man får en generell mekanisme for håndtering av felt, som konseptuelt er en bedre løsning. Dette vil gjøre systemet enklere å forstå og lettere å vedlikeholde. For å implementere dette trengs det plugins som håndterer de innebygde datatypene som `string` og `file`.

3.4.2 Forenkling av plugininstallasjon

I delkapittel 3.3.2 ble prosessen med å lage og integrere en plugin i DPG forklart. Dette er en unødvendig komplisert prosess med flere fallgruver. For eksempel inneholder filen `plugIn.xml` stort sett informasjon som kan automatiseres ved hjelp av diverse teknikker, som refleksjon eller ved å endre på pluginarkitekturen.

Det er derfor konkludert med at XML-filen `plugIn.xml` er overflødig. Dens arbeidsoppgaver kan automatiseres ved hjelp av refleksjon og ved at plugins selv sørger for å holde rede på hva slags inndata de krever. Det er plugin selv som vet best hva slags type inndata og hvilke parametere den trenger for å fungere.

Figur 3.2: Eksempel på merkelappskyplugin til last.fm.

En fordel med å beholde filen `plugIn.xml` er at man da vet hvor man skal se for å finne ut hvilke plugins som er tilgjengelig i DPG. Likevel er ikke dette en god nok grunn for å komplisere prosessen med å inkludere nye plugins. Dersom en lurer på hvilke plugins som er tilgjengelig for bruk, kan man se på stien der JAR-filene som innholder implementasjonen av forskjellige plugins er lokalisert.

3.4.3 Ressurstilgang

Plugins har ikke tilgang til ressurser fra andre entiteter. Dette ville være en nyttig mulighet dersom plugins skal brukes til å behandle informasjon fra presentasjonen den befinner seg i.

Et eksempel på når slik tilgang vil være nyttig, er en plugin som henter ut informasjon fra en entitetsliste og presenterer den på en ny måte. Noe tilsvarende er gjort på nettstedet *fxdteam.com* [43], som benytter seg av det åpne API-laget til musikknettstedet *last.fm* [64]. De henter informasjon om hvilke sanger en bruker har hørt på for å genere en *merkelappsky* (eng. *tagcloud*) [60] som på en lettfattelig måte gir oversikt over musikksjangrene vedkommende lytter på. De sjangrene som er lyttet på flest ganger er vist i større skriftstørrelse enn de som er lyttet på færre ganger. I figur 3.2 kan man se at sjangeren *Minimal* har størst skriftstørrelse, noe som betyr at den er den mest populære sjangeren hos denne brukeren. De små navnene under sjangeren er eksempler på populære artister eller grupper som opererer i sjangeren.

Etter å ha sett nærmere på om ressurstilgang virkelig ville være nyttig, ble det konkludert med at det var best å løse slike situasjoner uten å benytte plugins. Det å presentere informasjonen fra entiteter i forskjellige utsnitt på forskjellige måter er en del av nøkkelfunksjonaliteten til DPG, og det er utbredt støtte for forskjellige måter å sortere eller filtrere informasjonen i XSLT-transformasjonen.

En fremtidig løsning kan være å tilby et API-lag som programmerere kan benytte for å hente ut informasjon fra systemet, men nytteverdien for denne løsningen er liten. Slike API-lag krever at formen av dataene er faste. DPG har en dynamisk struktur, og det er derfor vanskelig å hente ut data slik det ofte gjøres i slike API-lag.

3.4.4 Filsystemplassering av plugins

I dagens system legges plugins i mappen `lib` under DPGs rotmappe. Denne mappen brukes også til alle andre bibliotek, som for eksempel loggsystemet log4j [27] og XML-biblioteket JDOM. Dersom man skal gjøre endringer i `lib`-mappen er det ikke lett å få oversikt når bibliotek og pluginfiler er i samme mappe. En bedre løsning er å plassere plugins i en egen undermappe under `lib`-mappen.

3.4.5 Hendelseshåndtering

Plugins kan i skrivende stund ikke reagere på hendelser. Det hadde vært nyttig å ha denne funksjonaliteten dersom en plugin skal kunne oppdatere innholdet sitt basert på for eksempel innholdsendring i systemet, eller at et visst klokkeslett inntreffer. Dette er en såpass sentral funksjon at det er ønskelig med en generell mekanisme som håndterer dette.

3.4.6 Skrive- og lesetilgang

Plugins har ikke mulighet til å lese eller skrive til filer. Dette må være mulig for brukere av presentasjonen dersom man har en plugin som håndterer lagring av kommentarer. For å lese av kommentarer trengs det en plugin med lesetilgang, slik at man kan hente ut lagrede kommentarer. Et annet eksempel der lagring vil være fornuftig er dersom man har et pluginspill og ønsker å lagre de beste resultatene.

Den valgte løsningen benytter den eksisterende klassen `ResourceDao` for lagring av data. Denne klassen bruker Java Content Repository (JCR). Flere av parameterne som kreves i klassen `ResourceDao` kan settes automatisk, og det er derfor fornuftig å benytte et *fasademønster* (eng. *facade-pattern*) [44] for å få en bedre kontrakt å jobbe mot.

3.4.7 Inndataløse plugins

Et av feltene som må defineres i filen `plugIn.xml` for hver plugin er feltet `field`. Dette feltet beskriver hva slags data som forventes av rollen *publisher* når innholdet i presentasjonen blir redigert i delsystemet PCE. Dette feltet kan være `string`, `date`, `xhtml` eller `file`. I flere scenarioer vil det derimot oppstå situasjoner der inndata ikke er ønskelig.

Et eksempel på en slik situasjon er sporing av brukeraktivitet ved hjelp av verktøyet Google Analytics [54]. Dette verktøyet krever at sidene inkluderer et fire linjer langt Javascript som håndterer koblingen til Google. Det eneste som må forandres i denne kodesnutten er at man må definere en *bruker-ID*. Denne bruker-IDen vil være lik for alle sidene i en presentasjon. Den beste løsningen vil i denne situasjonen være å definere bruker-ID som en parameter i konfigurasjonsfilen `pluginPattern.xml`, innholdet i denne filen kan ikke forandres av en publisher. Bruker-ID er det eneste konfigurerbare med en slik plugin, og derfor trenger ikke plugin et eget inndatafelt.

3.4.8 Redgjøring av inndata og parametre

En plugin kan hente inndata fra to forskjellige kilder. Den første kilden er gjennom å forandre presentasjonen, det vil si at en bruker som er med i brukergruppen *publisher* gjør forandringer i skjemaene til delsystemet PCE. Den andre kilden er parametre som settes i filen `pluginPattern.xml` (vist i 5. linje i listing 3.2).

Omverdenen har likevel ikke noen mulighet for å vite hva pluginen forventer av hverken parametre eller inndata i presentasjonen. Dette gjør at for eksempel et fremtidig redigeringsverktøy som skal hjelpe utvikleren med å produsere korrekte presentasjonsmønstre vil få problemer. Dersom et redigeringsverktøy eller en utvikler vil vite hvilken type inndata eller parametere en plugin trenger, må den inn i implementasjonen av pluginet og finne ut hvor inndata fra parameteret `parameterMap` eller `idOfResource` er brukt.

For å gjøre det enklere å finne ut hvilke parametere og inndata en plugin trenger, bør det derfor opprettes metoder som returnerer dette.

3.4.9 Tillate inndata til plugins

Plugins i DPG 2.0 tillater kun inndata fra PCE og parametere i filen `pluginPattern.xml`. Det er ønskelig å utvikle støtte for at også brukere med brukerrollen *reader* skal kunne sende data til systemet.

3.4.10 Pluginkontrakter

Pluginkontraktene er definert på en svært uoversiktlig måte. Dette skyldes først og fremst at kontraktene har for mange parametere, `SingleFieldInputPlugin` og `EntityListInputPlugin` har henholdsvis elleve og tretten parametere som vist i listing 3.4 og 3.5. Med så mange parametere er det svært vanskelig å få kontroll over parameterrekkefølgen. Det hjelper heller ikke at parameterrekkefølgen på kontraktene er inkonsekvent, de inneholder flere like parametere, men de dukker opp i forskjellige rekkefølge i de to kontraktene. I følge boken *Clean Code* bør man unngå å ha flere enn to parametere i en metode [69].

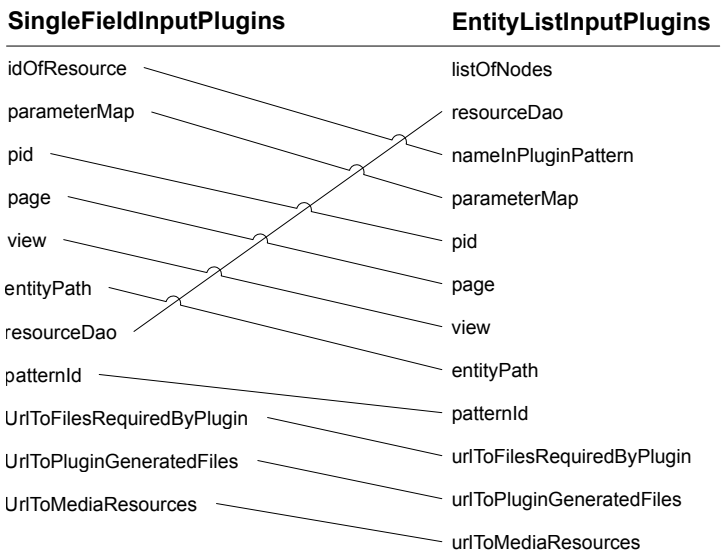
Listing 3.4: SingleFieldInputPlugins sin metode for generering av pluginelement

```
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);
```

Listing 3.5: EntityListInputPlugins metode for generering av pluginelement

```
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);
```

Et eksempel på den nevnte inkonsekventheten er parametret `resourceDao`, som forekommer som nummer syv i `SingleFieldInputPlugin` og som nummer to i `EntityListInputPlugin`. For en oversikt over inkonsistensen, se figur 3.3. Strekene forbinder de ekvivalente parametrene i de to pluginkontraktene `SingleFieldInputPlugins` og `EntityListInputPlugins`. Det er viktig å påpeke at feltene `idOfResource` og `nameInPluginPattern` inneholder samme informasjon, og at de tre siste parameterene



Figur 3.3: Forskjeller mellom de to kontraktene.

er skrevet med forskjellig bokstavtype. I følge Javas kodekonvensjoner [72] skal variabler begynne med liten forbokstav, og de tre siste parameterne i listing 3.4 bør endres til liten forbokstav.

En bedre og mer oversiktlig løsning for å håndtere disse parameterene er å benytte seg av en pluginbønne som har kontroll på de forskjellige parameterene. Dette vil bryte bakoverkompabilitet, men pluginkontraktene bør uansett endres på grunn av inkonsekvens i parameterrekkefølgen.

4

Endringer i pluginarkitektur

Dette kapittelet er skrevet i fellesskap av Tobias Rusås Olsen [77] og Peder Lång Skeidsvoll. Kapittelet inngår i begges masteroppgaver. Årsaken til at dette er et felleskapittel er at begge parter var interessert i å videreutvikle pluginarkitekturen for å kunne gjennomføre målene i sine masteroppgaver, som henholdsvis omhandler rike klienter i DPG og bruk av interaktive plugins. Konklusjoner i dette kapittelet er gjort på grunnlag av samtale mellom de to forfatterene. Det er også kommet innspill fra veiledere under ukentlige møter.

4.1 Utbedringer

I kapittel 3 ble pluginarkitekturen til DPG 2.0 gjennomgått, og en rekke svakheter og mangler ble avdekket. I dette kapittelet vil utbedringene som er utført på systemet bli gjennomgått.

4.1.1 Bruk av plugins i presentasjonsmønstre

Det er blitt bestemt å fjerne innebygde typer, og la alle felt være prosessert av plugins. Les mer om dette i kapittel 5: *Ny presentasjonsmønsterspesifikasjon.*

4.1.2 Endring av navn på filen `pluginPattern.xml`

I den orginale versjonen av pluginarkitekturen fantes det en konfigurasjonsfil med navn `pluginPattern.xml`. Denne filen ble brukt til å konfigurere plugins. Med konfigurering menes det å gi hver enkelt plugin et navn som brukes i `pattern.xml`, og parametere som inndata til pluginet. Filnavnet `pluginPattern.xml` gir ingen informasjon om dette. Derfor er det besluttet å gi filen det nye navnet `pluginConfig.xml`, som reflekterer filens innhold på en bedre måte.

4.1.3 Forenkling av plugininstallasjon

Ved å fjerne konfigurasjonsfilen `plugIn.xml` er plugininstallasjonen gjort betraktelig enklere. Denne filen ble brukt til å laste inn plugins til DPG 2.0, slik at de kunne benyttes i presentasjonsmønstre.

For å kunne fjerne denne filen måtte det finnes erstatninger for dens bruksområder. Dette er arbeidsoppgavene til konfigurasjonsfilen `plugIn.xml`:

1. Definere sti til hvor plugins befinner seg i filsystemet.
2. Finne ut hvilken type inndata pluginet forventer, for eksempel `string`, `xhtml` eller `file`.
3. Gi pluginet et navn til bruk i pluginkonfigurasjonsfilen `pluginPattern.xml`.
4. Definere hvilken kontrakt pluginet implementerter, `SingleFieldInputPlugin` eller `EntityListInputPlugin`.

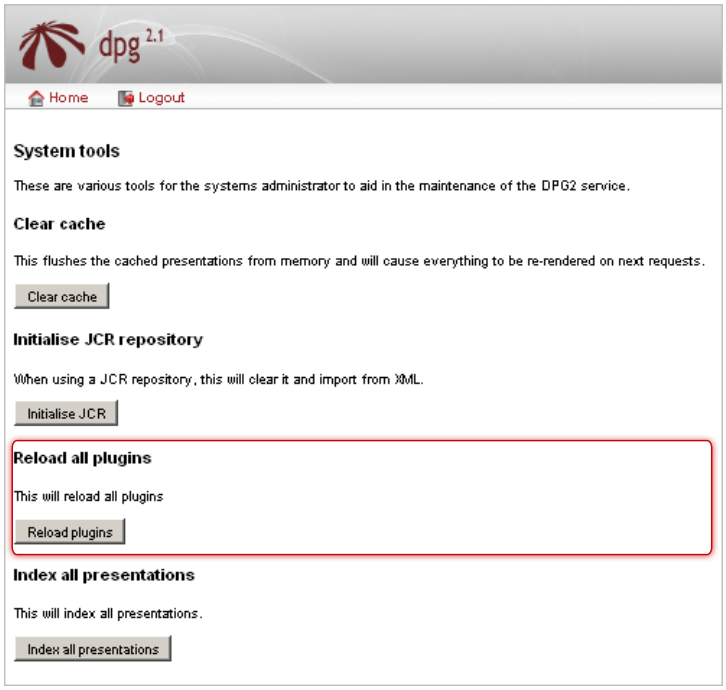
For å unngå å måtte definere hvor hver enkelt plugin er plassert, er vurdert at den beste løsningen er å spesifisere en sti til én mappe som inneholder plugins. Stien til mappen med plugins blir da definert i en *innstillingsfil* (eng. *property file*). Det ble bestemt å legge stien i filen `repository.properties`, siden dette er filen der alle andre stier som omhandler DPGs *oppbevaringssted* (eng. *repository*) er definert. Dette er et eksempel der pluginstien settes til `lib/plugins`:

```
repository.path.plugins = lib/plugins
```

Når det er laget en plugin, pakkes den inn i en JAR-fil og legges i denne mappen.

DPG 2.1 vil ved oppstart, eller ved å trykke knappen merket *Reload Plugins* i systemets *Verktøy*-side (siden vises i figur 4.1), automatisk sjekke alle JAR-filer etter klasser som implementerer grensesnittet `SingleFieldInputPlugin` og laste disse inn i systemet ved hjelp av refleksjon. På denne måten blir punkt 1 og 4 løst.

For å løse punkt to, ble det lagt til en ny metode i kontrakten `SingleFieldInputPlugin`. Metoden har følgende signatur:



Figur 4.1: Verktøyvinduet til DPG 2.1. Feltet for å laste inn plugins er rammet inn.

```
public String getFieldType();
```

Metoden returnerer hvilken type inndata pluginet forventer, for eksempel `file` eller `string`.

I forhold til punkt 3 er en mulig løsning at DPG kan bruke klassenavnet til pluginet som navn. Det vil si at hvis klassenavnet til en plugin er `MapPlugin`, vil navnet til pluginen i `pluginConfig.xml` bli det samme. Likevel bør man ha muligheten til å gi pluginet et annet navn dersom det skulle være behov for det. Et eksempel er hvis man ønsker en plugin med navn `String`. Uten alternativ løsning ville dette navnet ha havnet i konflikt med `String`-klassen i Java-biblioteket.

Dette kan løses ved å legge til en ekstra metode i kontrakten, for eksempel `getName()`. Dette ville vært trivielt å implementere, men ville forkludret kontrakten uten god grunn. DPG må dessuten ha sjekke returverdien til `getName()`-metoden for verdien `null` hvis man skal ha klassenavn som standardverdi.

En annen løsning er å ha en abstrakt klasse som et mellommledd mellom kontrakten og pluginet. Denne klassen kan da ha en implementasjon av metoden `getName()` som returnerer klassenavnet, men som kan overskrives i pluginimplementasjonen dersom man ønsker å ha noe annet enn standardverdien. Ulempen med denne løsningen er at plugins da ville utvidet en klasse framfor å implementere et grensesnitt, noe som ikke er anbefalt [44]. Likevel vil denne implementasjonen løse problemet på en tilfredsstillende måte.

Løsningen som er valgt er å bruke *annoteringer* (eng. *annotations*) [75]. Listing 4.1 viser et eksempel på hvordan man bruker en slik annotering.

Listing 4.1: Navnsetting

```
@PluginName(name="String")
public class MyAnnotatedStringPlugin implements SingleFieldInputPlugin {
  ...
}
```

I denne løsningen sjekker DPG 2.1 om klassen er annotert med `@PluginName`. Dersom den er det, vil den få det annoterte navnet i systemet. Hvis den ikke har noen annotasjon, vil pluginet bli navngitt etter navnet på klassen.

4.1.4 Ressurstilgang

Som nevnt i delkapittel 3.4.3 ble det enighet i at det ikke er en god idé å gi plugins tilgang til ressurser fra andre utsnitt, og dette er derfor ikke implementert.

4.1.5 Hendelseshåndtering

Hendelseshåndtering vil uten tvil være en veldig nyttig funksjon. Det er likevel konkludert med at det ikke blir prioritert å bruke tid på å implementere denne løsningen, da dette vil være en relativt stor og tidkrevende oppgave som vil gå på bekostning av oppfyllelsen av oppgavens hovedmål.

Senere masterstudenter oppfordres til å utvikle en løsning for dette.

4.1.6 Skrive- og lesetilgang

Skrive- og lesetilgang er en essensiel funksjonalitet. Dette er fordi avanserte plugins må ha tilgang til å lagre og hente ut data. Det er utviklet en løsning som bruker DPGs eksisterende arkitektur for å lagre ressurser. Denne arkitekturen har en kontrakt med navn `ResourceDao` som inngangsport.

Listing 4.2: Eksempel på metode fra `ResourceDao`

```
public void importPresentationResource(
        String presentationId,     // unik id for en presentasjon
        String resourceName,       // Hvor innholdet skal lagres og
                                   //     unikt navn  for innholdet
        String parentFolderPath,   // mappe der innhold skal lagres
        String mimetype,           // definisjon av innholdstype
        ResourceType type,         // type ressurs
        InputStream resourceStream // innholdet som skal lagres
);
```

`ResourceDao` tilbyr en rekke metoder for lagre eller hente ut data fra persistenslaget. I listing 4.2 vises et eksempel på en typisk metode fra denne kontrakten. Metoden viser hvilke parametre som kreves for å importere en ressurs.

Det vil nå bli gjennomgått hvilke parametre som er nødvendige, og hvilke som det ikke er behov for.

Når en plugin vil lagre en ressurs vil den alltid vite hvilken presentasjon den befinner seg i. Det er derfor ikke nødvendig å eksplisitt oppgi hvilken presentasjon den vil lagre ressursen i. Det er heller ikke ønskelig at en plugin skal kunne lagre data i en annen presentasjon. Parameteret `presentationId` er derfor ikke nødvendig.

Parameteret `parentFolderPath` er ikke tatt i bruk i dagens system og er derfor ikke nødvendig å oppgi.

Parameteret `ResourceType` er en *oppramstype* (eng. *enumeration*) der en plugin alltid vil ha verdien `ResourceType.PLUGIN_RESOURCE`, det er derfor ikke nødvendig å oppgi dette parameteret eksplisitt.

På grunn av disse observasjonene, er det bestemt å lage en ny kontrakt for lagring av pluginressurser. Denne kontrakten har fått navnet `PluginDao` og bruker et *fasademønster* (eng. *facade pattern*) [44]. Et fasademønster brukes til å tilby et enkelt grensesnitt til et komplekst undersystem. Metoden for å lagre en ressurs til persistenslaget, som er vist i listing 4.2, er i `PluginDao` blitt forenklet til å bli som i listing 4.3. Metoden `saveResource()` trenger kun tre parametre, en klar forenkling i forhold til metoden `importPresentationResource()` som krever seks.

Listing 4.3: Eksempel på metode fra `PluginDao`

```
public String saveResource(
  String filePath,
  String mimeType,
  InputStream resource
);
```

Klassen `ResourceDao` lagrer data som en nøkkelverditabell ved at det blir defineret en nøkkel som passer til en verdi. Utvikleren må bruke nøkkelen for å hente verdien. I dette tilfellet er parameteret `filePath` nøkkelen og parameteret `resource` verdien. I tillegg knyttes det

til en `mimeType` som gjenspeiler hvilken type data som lagres. *MIME* (*Multipurpose Internet Mail Extensions*) [9] er en tekststreng som forteller mottakeren av ressursen hva slags type ressurs det er. Dette for at mottakeren skal kunne vite hva den skal gjøre med den. For eksempel vet en nettleser at en MIME-type som er av typen `application/xhtml+xml` skal behandles som HTML, mens en MIME-type av typen `image/gif` skal behandles som et bilde.

Denne løsningen gir stor fleksibilitet i forhold til hva man velger å lagre. Det som lagres er av klassen `InputStream` og man kan derfor lagre stort sett hva man vil, enten det er en bildefil, en XML-fil eller et lydspor. Under vises stien som benyttes for å kunne rette en forespørsel til serveren som returnerer en ressurs.

```
getPatternResource.html?patternId=patternId&type=
                                        PLUGIN_RESOURCE&fileId=filePath
```

Selv om dette er en bedre løsning en den som var der fra før, er den ikke optimal. Et av de største problemene er at plugins kan skrive over hverandres ressurser. Dette er fordi det ikke finnes en egen lagringsplass for hver enkelt plugin, eller noen form for eierskap på ressursene. Dette bør håndteres i en fremtidig versjon av DPG.

4.1.7 Inndataløse plugins

Plugins som ikke trenger inndata fra brukerrollen *publisher*, kan få slippe det. Dette gjøres ved å definere at pluginets metode `getFieldType()` returnerer strengverdien `none`. Dersom en *publisher* vil endre innholdet på en side der en slik plugin finnes, vil ikke feltet dukke opp over listen med felt med inndata.

4.1.8 Redegjøring av inndata og parametre

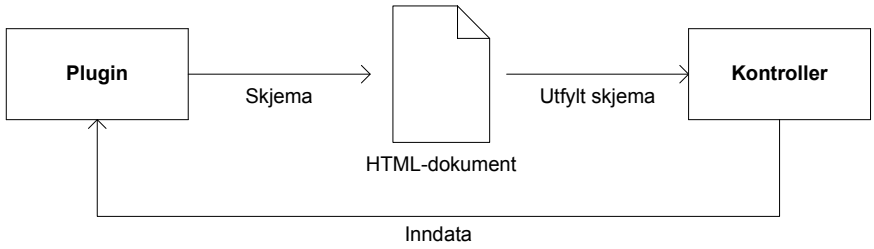
For å gjøre det lettere for fremtidige utviklingsverktøy for presentasjonsmønstre å forstå hvilke parametre en plugin støtter, er alle plugins utrustet med metoden `getParameters()`. Slik ser signaturen til metoden ut:

```
public List<String> getParameters();
```

Metoden returnerer en liste med navnene til alle parametrene som plugin-programmereren trenger å ta hensyn til.

4.1.9 Tillate at plugins håndterer inndata

Det er utviklet støtte for plugins som kan få inndata fra brukere med brukerrollen *reader*. For å gjøre dette mulig er det utviklet en kontrollerklasse med navn `PluginFormController`.

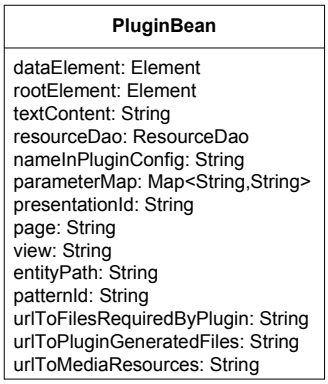


Figur 4.2: Interaksjon fra bruker.

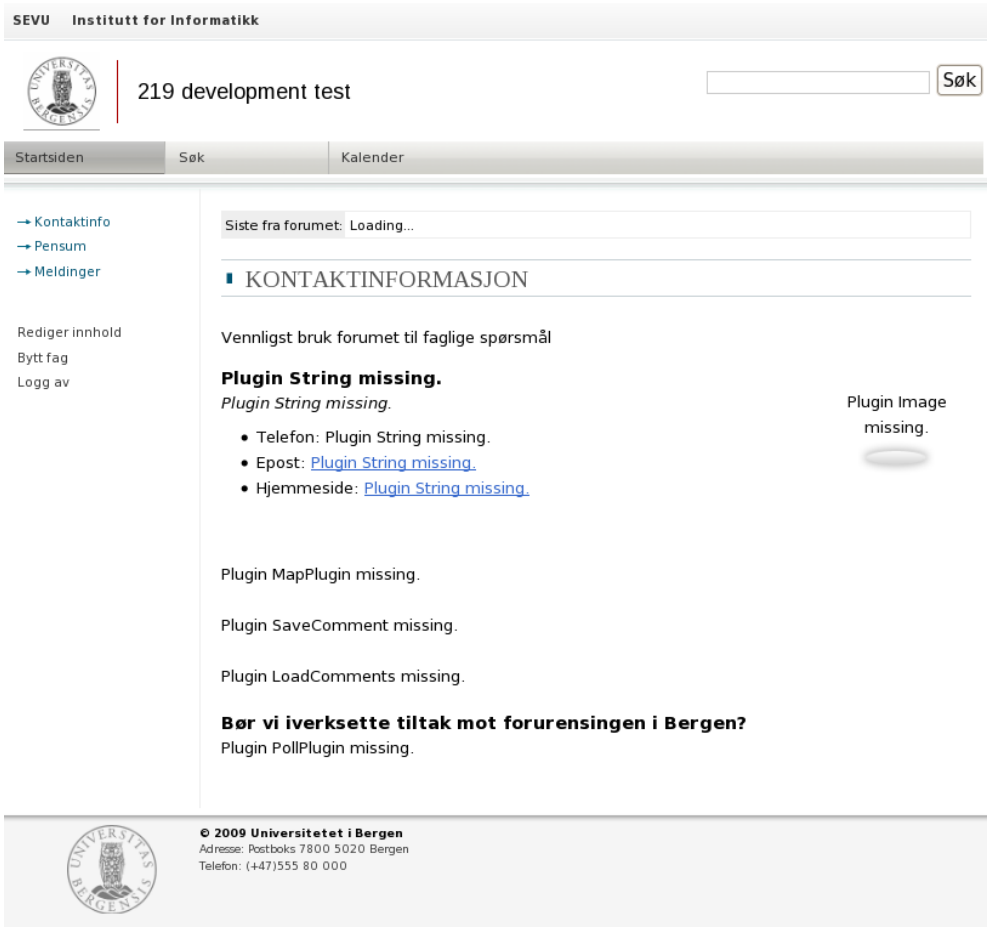
Kontrollerens oppgave er å håndtere inndata som er sendt fra brukeren. Inndataen blir sendt gjennom et HTML-skjema generert av pluginet og fylt ut av brukeren. `PluginFormController` har et forespørselsobjekt som inneholder inndata. Inndata blir omgjort til en nøkkelverditabell i kontrolleren. Nøkkelverditabellen sendes så tilbake til pluginet i metoden `processInputs()`. Dette er en ny metode som alle plugins må implementere. Plugins kan så behandle denne nøkkelverditabellen etter eget ønske. Figur 4.2 viser hvordan denne prosessen foregår.

4.1.10 Pluginskontrakter

Det er valgt å lage en *bønne* (eng. *bean*) med navn `PluginBean`. Denne bønnen inneholder alle parametrene som tidligere ble sendt som metodeparametre (se figur 3.3). I tillegg til dette er det valgt å fjerne hele kontrakten `EntityListInputPlugins` fordi den har mye overlappende funksjonalitet med kontrakten `SingleFieldInputPlugins`. Den ekstra funksjonaliteten kan like gjerne kan legges til i denne kontrakten. Metodene til `PluginBean` vises i figur 4.3.



Figur 4.3: Den nye PluginBean klassen.



Figur 4.4: Skjermbilde av side der alle plugins mangler.

4.1.11 Feilhåndtering av plugins

Det er et krav til applikasjoner med pluginarkitektur at feilende plugins ikke skal få vertsapplikasjonen til å feile. Det er derfor implementert en endring som gir beskjed på nettsiden hvis em plugin mangler. Man vil også få beskjed om hvilken plugin det er som mangler, slik at man skal kunne håndtere problemet uten å lese i systemets logger.

Feilhåndteringen er løst ved å undersøke om plugin-navnet definert i filen `pluginConfig.xml` er lastet inn i systemet. Dersom pluginet ikke finnes, returneres det istedenfor et JDOM-element med innhold som forklarer at pluginet mangler. Figur 4.4 viser hvordan siden ser ut dersom alle plugins mangler.

4.2 Konklusjon

Det har nå blitt gjennomgått en rekke forandringer som er utført i DPGs pluginarkitektur. Forandringene har ført til at det nå er lettere å:

- Installere plugins.
- Bruke plugins i presentasjonsmønstre.
- Utvikle kraftige plugins.
- Videreutvikle DPGs pluginarkitektur.

I kapittel 5 undersøkes det hvordan presentasjonsmønsterspesifikasjonen kan forbedres for å støtte den nye plugin-arkitekturen.

5

Ny presentasjonsmønsterspesifikasjon

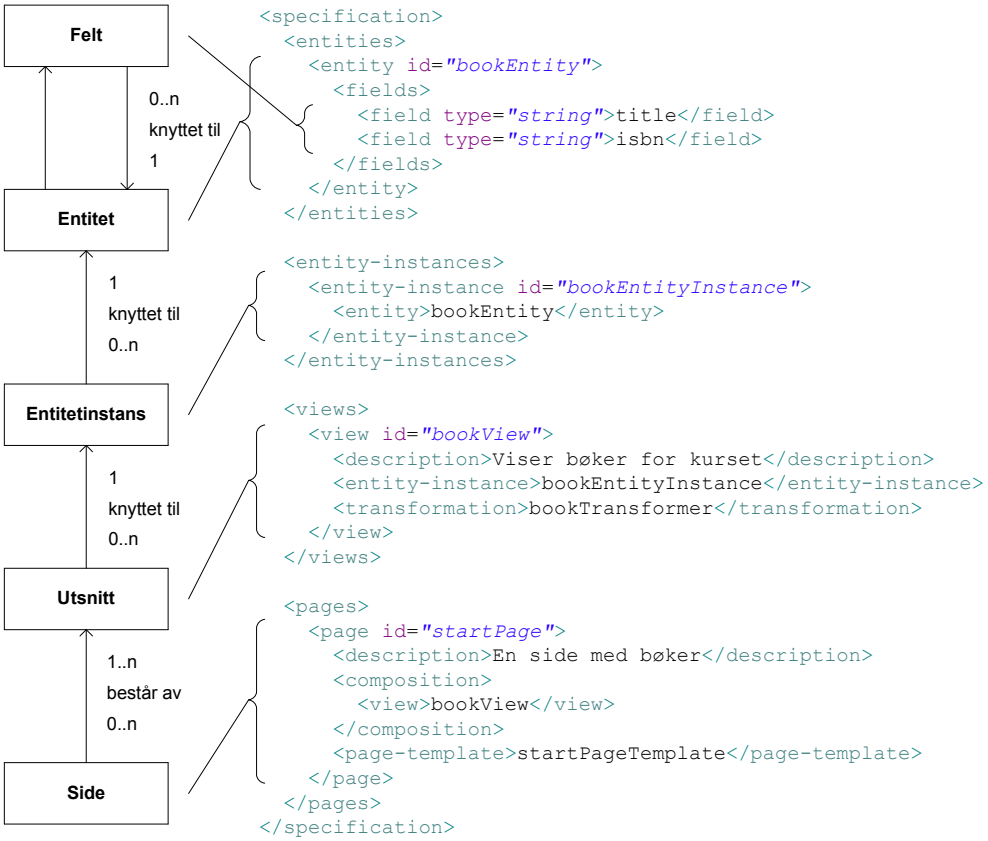
Dette kapittelet er skrevet i fellesskap av Tobias Rusås Olsen [77] og Peder Lång Skeidsvoll. Kapittelet inngår i begges masteroppgaver. Konklusjoner i dette kapittelet er gjort på grunnlag av samtale mellom de to forfatterene. Det er også kommet innspill fra veiledere under ukentlige møter.

Dette kapittelet beskriver endringer og forbedringer gjort i presentasjonsmønsterspesifikasjonen, som definerer elementene i et presentasjonsmønster.

5.1 Bakgrunn

Før arbeidet med DPG 2.0 startet, gikk masterstudentene Rossini og Liberati [89] gjennom presentasjonsmønsterspesifikasjonen til DPG 1.0 og foreslo endringer av denne. Dette arbeidet ble så igjen revidert, i all hovedsak av masterstudenten Sebak [80], noe som resulterte i presentasjonsmønsterspesifikasjonen som benyttes i DPG 2.0.

Figur 5.1 viser hvordan de forskjellige komponentene i presentasjonsmønsterspesifikasjonen til DPG 2.0 og sammenhengen mellom dem. For eksempel viser den at en *side* (eng. *page*) består av en eller flere *utsnitt* (eng. *view*), og et et *utsnitt* er å finne på null eller flere *sider*. Det er ikke laget noen tekniske begrensninger på at et utsnitt må være tilknyttet en side, men et utsnitt uten en slik kobling vil ikke ha noen nytteverdi.



Figur 5.1: Presentasjonsmønsterspesifikasjonen til DPG 2.0.

5.2 Bruk av plugins i presentasjonsmønstre

DPG 2.0 har to forskjellige måter å definere datatypen til et felt: Innebygde typer eller plugins. Det finnes i DPG 2.0 fem forskjellige innebygde typer: `string`, `date`, `entity-list`, `file` og `date`. Listing 5.1 viser hvordan en innebygd type defineres i DPG 2.0. I dette eksempelet finnes det et felt som heter `boktittel` som er av typen `string`.

Listing 5.1: Eksempel på et felt av typen `string`.

```
<field type="string">
  boktittel
</field>
```

Dersom et felt skal håndteres av en plugin i stedet for en innebygd type, må feltet defineres som i listing 5.2. I dette eksempelet er typen til feltet `boktittel` definert som `plugin` gjennom `type`-attributtet.

Dette forteller DPG 2.0 at systemet nå skal bruke en plugin, og gjør samtidig at DPG 2.0 forventer et attributt med navn `pluginConfig`. Dette attributtet forteller DPG at feltet skal bruke pluginet `bokPlugin`.

Listing 5.2: Eksempel på et felt som bruker en plugin.

```
<field type="plugin" pluginConfig="bokPlugin">
  boktittel
</field>
```

Dette er en uheldig løsning fordi plugins ikke bør være et spesialtilfelle. En bedre løsning er å la plugins være den eneste måten å definere typen til et felt. Ved å implementere denne løsningen vil DPG få en generell mekanisme for håndtering av felttyper. Fordelen med denne løsningen er:

- Bruken av plugins fremmes slik at det blir mer aktuelt for presentasjonsmønsterutviklere å benytte seg av plugins.
- Filen `pattern.xml` vil bli enklere å lese og skrive.
- Kodebasen til DPG kortes ned siden den ikke trenger å behandle spesialtilfeller.

I DPG 2.1 er de innebygde typene forsvunnet, og alt er overlatt til plugins. I den nye spesifikasjon er det semantiske innholdet til attributtet `type` endret. Attributtet `type` angir nå hvilken plugin feltet skal håndteres av. Slik ser det ut hvis det lages et felt med den nye spesifikasjonen, der navnet på feltet er `feltnavn`, og `pluginNavn` er navnet på pluginet som håndterer det:

```
<field type="pluginNavn">feltnavn</field>
```

Denne endringen gjør det enklere å bruke plugins i et presentasjonsmønster ved at det ikke lenger er et spesialtilfelle, men den vanlige fremgangsmåten for å definere typen til et felt. For å gjøre overgangen lettere er det utviklet plugins som erstatter de tidligere innebygde typene. Disse er:

- `date`
- `file`
- `string`
- `xhtml`

Den eneste innebygde typen som ikke er implementert er lister, og det vil i det neste delkapittelet diskuteres løsninger for dette.

5.3 Lister

I DPG 2.0 er lister definert ved å sette attributten `type` til `entity-list` i et felt. Listing 5.3 viser et eksempel på en slik listedeklarasjon. Attributten `ref-id` gir DPG beskjed om hvilken entitet det skal lages liste av.

Listing 5.3: Et felt defineres som en entitetsliste.

```
<entity id="softwareEntity">
  <fields>
    <field type="entity-list" ref-id="url">URLs</field>
  </fields>
</entity>
```

I avsnitt 5.2 ble de innebygde typene til DPG fjernet, og det trengs derfor en erstatning for lister. Lister er en spesiell felttype fordi den inneholder andre entiteter. Målet er at også lister skal omfavnes av den generelle mekanismen for felttyper.

Det er utarbeidet tre forslag for hvordan lister skal håndteres i DPG 2.1, og disse vil nå bli presentert.

5.3.1 Lister behandles som i DPG 2.0

Dette forslaget innebærer å la lister være uforandret. Fordelen er at mønstre som allerede er utviklet kan brukes uten modifikasjon, noe som gir en viss bakoverkompabilitet. Løsningen i DPG 2.0 er i tillegg en velprøvd metode som har vist seg å fungere i praksis. Ulempen til forslaget er at målet om uniform håndtering av felt ikke tilfredsstilles, og forslaget er derfor uaktuelt dersom det finnes andre løsninger som tilfredsstiller dette målet.

5.3.2 Initialisering av lister i entititetsinstansen

Det andre forslaget er å definere entitetslister på entitetsinstansnivå. Dette betyr at lister definerers på nivået over entitet (se figur 5.1).

Listing 5.4 viser et eksempel på hvordan en slik løsning kan se ut. Her defineres en todimensjonal liste av entiteten `softwareEntity`.

Listing 5.4: Lister definiert i entitetsinstanser.

```
<entity-instances>
  <!-- Her defineres en todimensjonal liste -->
  <entity-instance id="urlEntityInstance" type="[][]">
    <entity>softwareEntity</entity>
  </entity-instance>
</entity-instances>

<entity id="softwareEntity">
  <fields>
    <field type="string">URLs</field>
  </fields>
</entity>
```

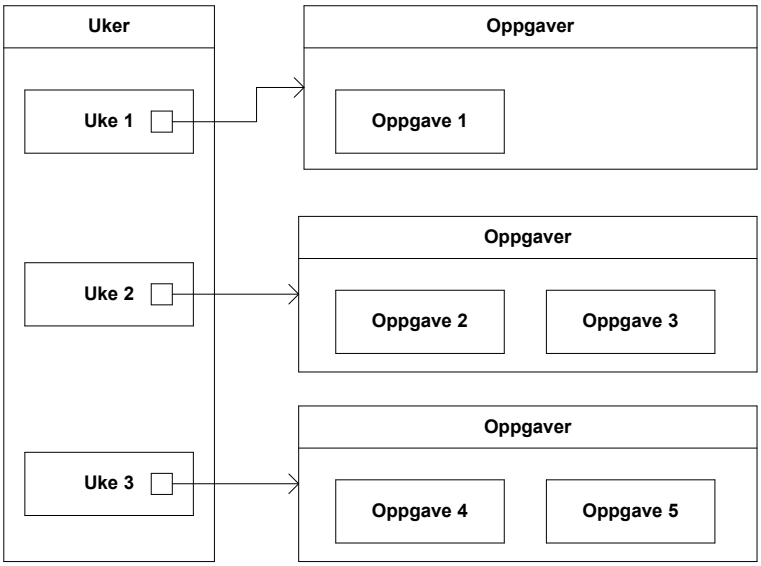
Listen er definert ved hjelp av []-paranteser. Dette er samme syntaks som blir brukt til å definere *tabeller* (eng. *arrays*) i programmeringsspråkene Java og C. I dette tilfellet er listen todimensjonal.

Svakheten til denne løsningen er at det er umulig å definere lister innenfor andre lister. Dette er fordi entiteter ikke kan inneholde entitetsinstanser.

For eksempel er det i Kursmønsteret viktig å kunne ha flere oppgaver på en uke. Dette må løses ved at det er en liste med uke-entiteter, der hver uke-entitet inneholder en liste med oppgave-entiteter. Figur 5.2 viser et diagram over eksempelet. Diagrammet har en liste med uker, der hver uke inneholder en liste med oppgaver. Dette kan ikke gjennomføres med dette forslaget. En uke-entitet måtte i så tilfelle hatt en referanse til en entitetsinstans, noe som ikke er mulig fordi lister defineres, i dette forslaget, på entitetsinstansnivå.

5.3.3 Lister som plugins

Det siste forslaget som er utarbeidet er å la lister bli behandlet av plugins. Det vil da være få forskjeller i presentasjonsmønsteret fra hvordan det ble skrevet i DPG 2.0. Forslaget vil kreve en del endringer i koden. Listing 5.5 viser hvordan en entitet som inneholder en liste i ett av feltene sine vil se ut hvis dette forslaget tas i bruk. Her defineres en liste av entiteten `url`.



Figur 5.2: Liste med uker som inneholder lister med oppgaver.

Listing 5.5: Lister definert med plugins.

```
<entity id="softwareEntity">
  <fields>
    <field type="stringPlugin">title</field>
    <!-- Her blir listen med url-er definert -->
    <field type="list" entity="url">urls</field>
  </fields>
</entity>

<entity id="url">
  <fields>
    <field type="string">title</field>
    <field type="string">address</field>
  </fields>
</entity>
```

Denne fremgangsmåten har flere fordeler. Blant annet er det mulig å utvikle flere listeplugins som for eksempel kan generere RSS-filer basert på innholdet i listen.

Konklusjon

Det er ikke nødvendig å ta hensyn til bakoverkompatiblitet. Det er avdekket at presentasjonsmønsterspesifikasjonen i DPG 2.0 har flere alvorlige svakheter, og derfor bør disse tas hånd om før det utvikles flere presentasjonsmønstre. Denne vurderingen gjør at det forslaget om å beholde listehåndteringen fra DPG 2.0 ikke er aktuelt.

Figur 5.3: Et utsnitt som inneholder en entitet

Forslaget som omhandler instansieringen av lister på instansnivå, ser ved første øyekast ut som et godt forslag. Ved nærmere undersøkelse viser det seg at det vil redusere funksjonaliteten til systemet. Dette er fordi lister i dette forslaget ikke kan inneholde andre lister, som påpekt i 5.3.2. Derfor er heller ikke dette et tilfredsstillende alternativ.

Det siste forslaget, der liste ble behandlet av plugins, utvider funksjonaliteten til lister ved å gjøre dem mer fleksible. Det vil i tillegg føre til at målet om en felles mekanisme for behandling av typer blir overholdt. På grunnlag av dette er det bestemt å utvikle en pluginløsning for bruk av lister.

5.4 Fjerning av `fields`-elementet

I presentasjonsmønsterspesifikasjonen til DPG 2.0 er elementet `fields` påkrevd. Dette elementet innholder `field`-elementene til en entitet. Listing 5.6 viser et eksempel på hvordan `fields` elementet blir brukt i DPG 2.0. Siden det ikke er mulig å legge inn andre elementer enn `field` i en entitet, er det besluttet å fjerne dette elementet. Elementet hadde ingen funksjon, og gjorde filen `pattern.xml` mer komplisert. Endringen gjør det lettere å lese og skrive presentasjonsmønstre.

5.5 Resultat

Figur 5.3 et et skjermbilde som viser hvordan en entitet kan fremstå når den er ferdig rendret. Navnet på entiteten er `Programvare`. Entiteten innholder en tittel, en beskrivelse, et bilde, og en liste med lenker. Lenkene består av en adresse og et lenkenavn.

Listing 5.6 viser hvordan disse entitetene blir definert i filen `pattern.xml` i DPG 2.0. Hvert felt må definere en type, og dersom det er valgt type `plugin` må det i tillegg defineres en attributt som forteller hvilken type plugin dette feltet skal knyttes til.

Listing 5.6: Definisjon av entiteter i DPG 2.0.

```
<entities>
  <entity id="softwareEntity">
    <fields>
      <field type="string">title</field>
      <field type="xhtml">description</field>
      <field type="plugin" pluginConfig="imagePlugin">image</field>
      <field type="entity-list" ref-id="url">URLs</field>
    </fields>
  </entity>

  <entity id="url">
    <fields>
      <field type="string">address</field>
      <field type="string">name</field>
    </fields>
  </entity>
</entities>
```

Listing 5.7 viser hvordan de samme entitetene som i listing 5.6 defineres med den nye presentasjonsmønsterspesifikasjonen til DPG 2.1. Alle feltene blir definert som plugins, inkludert lister. Koden er mer kompakt, og er både lettere å skrive og lese. Ved å indikere forskjellen mellom presentasjonsmønsterspesifikasjonene i DPG 2.0 og DPG 2.1 har typen til feltene i sistnevnte spesifikasjon fått postfikset `Plugin` selv om alle felter uansett er håndtert av plugins.

Listing 5.7: Definisjon av entiteter i DPG 2.1.

```
<entities>

 <entity id="softwareEntity">
   <field type="stringPlugin">title</field>
   <field type="xhtmlPlugin">description</field>
   <field type="imagePlugin">image</field>
   <field entity="url" type="listePlugin">urls</field>
 </entity>

 <entity id="url">
   <field type="stringPlugin">address</field>
   <field type="stringPlugin">name</field>
 </entity>
</entities>
```

6

AJAX og AJAX-biblioteker

6.1 Innledning

AJAX står for *Asynchronous JavaScript and XML*, og er en betegnelse på hvordan *flere* ulike teknologier og teknikker kan brukes til å skape interaktive nettsider. AJAX gjør det mulig å sende og motta data fra en server uten å sende forespørsler om hele sider, men istedenfor i form av *XMLHttpRequest (XHR)*-objekt [92].

XHR er et objekt som brukes til å sende forespørsler til en server om data. Selv om navnet knytter seg til XML, kan hvilken som helst type data bli overført ved hjelp av XHR. Objektet ble funnet opp av Microsoft, og ble første gang støttet i selskapets *Internet Explorer 5.0*. Microsoft implementerte XHR som et ActiveX-objekt [14], noe ingen andre nettlesere har støtte for. På grunn av dette er det laget en alternativ implementasjon som har blitt standardisert av organisasjonen *World Wide Web Consortium* (W3C). Denne implementasjonen blir ikke støttet av Internet Explorer versjoner tidligere enn versjon 7.0.

AJAX er et forholdsvis nytt konsept, og ble første gang beskrevet av Jesse James Garrett i 2005 [45]. Konseptet gjør det mulig å utvikle internettapplikasjoner med en interaktivitet som tidligere ikke var mulig uten bruk av tilleggsmoduler som *Adobe Flash* [49] og Java-applets [15], og har derfor blitt svært populært.

Det som i første omgang blir endret i det en nettside tar i bruk AJAX, er *arbeidsflyten* (eng. *workflow*) til brukeren. Det vil i de to neste avsnittene bli gjennomgått hvordan denne arbeidsflyten endres i det AJAX blir tatt i bruk.

6.1.1 Nettsider uten AJAX

Nettsider som *ikke* blir drevet av AJAX har en arbeidsflyt blir kalt *arbeid-vent* (eng. *work-wait*) [18]. Denne arbeidsflyten har to stadier. I det første stadiet venter klienten på data fra serveren, mens den i det andre presenterer data som den har fått i respons. Klienten er ikke tilgjengelig for bruk mens den er i det første stadiet, og brukeren av klienten må derfor vente til den kommer til det andre stadiet. Konkret innebærer denne modellen at klienten sender en forespørsel til en server om en ressurs. Denne ressursen kan for eksempel være et HTML-dokument eller et bilde. Mens dette pågår er det ikke mulig å utføre noen operasjoner på klienten. Serveren vil så svare ved å sende den ønskede ressursen tilbake til klienten, som deretter presenterer den for brukeren. Hvis noe av siden skal oppdateres, må prosessen gjentas. Figur 6.1 viser et sekvensdiagram over en typisk økt uten AJAX. Denne arbeidsflyten passer dårlig med brukerens tankegang, og er derfor ikke bra for brukervennligheten til nettsiden.

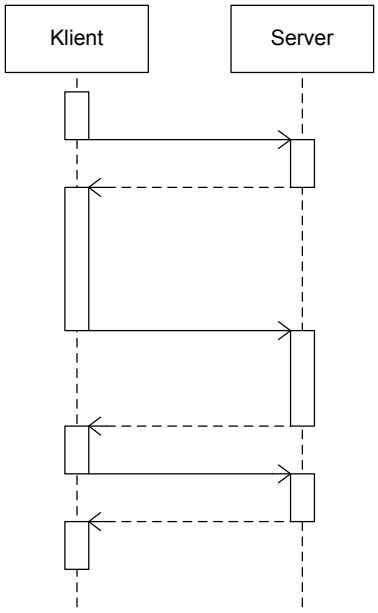
6.1.2 Nettsider med AJAX

Nettsider drevet med AJAX-konseptet vil ha en arbeidsflyt som kalles *arbeid-arbeid* (eng. *work-work*) [18]. Figur 6.2 viser en økt mellom server og klient der denne modellen benyttes. Kommunikasjonen med serveren vil foregå like ofte som på sider der AJAX ikke blir benyttet. Forskjellen er at klienten vil være tilgjengelig for bruk mens kommunikasjon pågår. Dette er mulig fordi kommunikasjonen foregår asynkront.

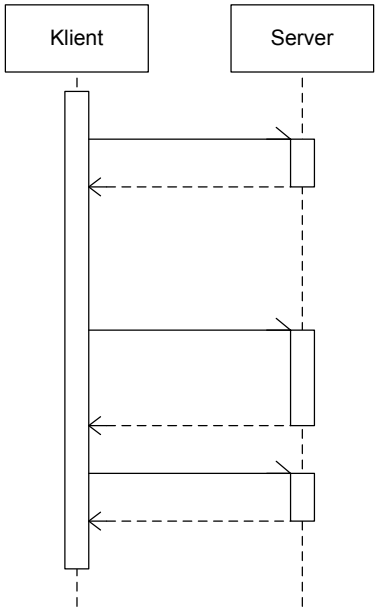
Denne modellen er ved første forespørsel lik den som ble beskrevet i avsnitt 6.1.1. Endringen er at responsen som blir sendt tilbake fra serveren inneholder eksekverbar kode som kan sende nye forespørsler til serveren. Forespørslene blir da, som tidligere nevnt, sendt i XHR-objekter. For mer informasjon om XHR og AJAX generelt, se [18].

6.2 JavaScript

For å benytte AJAX er det nødvendig med et *skriptprogrammeringsspråk* på klientsiden. Et skriptprogrammeringsspråk, ofte kalt *skriptsspråk*, er en type programmeringsspråk som blir eksekvert av et annet program, i dette tilfelle nettlesere, og ikke av systemets prosessor. Det finnes i dag flere forskjellige skriptspråk som støttes i nettlesere, slik som for eksempel *VBScript* [13] og *JavaScript* [30]. Det er likevel kun JavaScript som er støttet av samtlige moderne nettlesere. Som alle andre *kompilatorer* og *tolkere* (eng. *interpreters*) er de forskjellige leverandørenes programvare implementert på forskjellig vis, og har dermed noe ulike måter å tolke kode på. For å klare å programmere opp mot forskjellige nettlesere er det derfor meget tidsbesparende å benytte seg av et AJAX-bibliotek, som kan produsere kode som fungerer stort sett likt på alle de største JavaScript-implementasjonene. Tre slike bibliotek vil bli gjennomgått i neste avsnitt.



Figur 6.1: Kommunikasjon med server etter tradisjonell modell.



Figur 6.2: Kommunikasjon med server ved bruk av AJAX.

6.3 AJAX-biblioteker

Det har det i de de senere år oppstått flere forskjellige biblioteker for JavaScipt- og AJAX-utvikling. Disse bibliotekene har alle forskjellige utgangspunkt, og derfor forskjellig funksjonssett og bruksmåte. Det de alle har til felles er at de forsøker å gjøre det lettere for utviklere å produsere kode som fungerer på tross av de forskjellige JavaScript-implimentasjonene. De har også som oppgave å forenkle vanlige operasjoner som for eksempel å traversere DOM-treet til en internettside, og å sende XHR-forespørsler.

6.3.1 Indirekte- og direkte-biblioteker

Det finnes i all hovedsak to kategorier med AJAX-biblioteker. Den første kategorien kalles *indirekte AJAX-biblioteker*. Her skriver utvikleren kode i et annet språk, som for eksempel *Java*, *Python* eller *C*, som så blir kompilert om til JavaScript og tilhørende HTML og CSS [63]. Slike biblioteker er i all hovedsak konstruert for å utvikle internettapplikasjoner som kan erstatte tradisjonelle skrivebordsapplikasjoner. Eksempler på applikasjoner fra denne gruppen er *Google Documents* [55] og *GMail* [53]. Eksempler på biblioteker av denne typen er *Google Web Toolkit (GWT)* [56] som kompiler kode skrevet i Java, og *Wt* [20] som kompilerer kode skrevet i *C++*.

Den andre kategorien kalles *direkte AJAX-biblioteker*. I slike bibliotek må utvikleren kode rett i JavaScript. Bibliotekene er ofte små og kompakte, og inneholder stort sett bare basisfunksjonalitet for å gjøre det lettere å utvikle kode som kjører i forskjellige nettlesere. Slik basisfunksjonalitet kan være å hente et DOM-element fra en HTML-side eller å bruke XHR-objektet. Eksempler på slike biblioteker er Prototype [84], Dojo [35] og jQuery [61]. På grunn av at DPG i hovedsak brukes til nettsider som *presenterer* informasjon, vil det bli fokusert på denne typen biblioteker i resten av oppgaven.

6.3.2 Widgets

En viktig oppgave bibliotekene har, er å støtte noe som blir kalt *widgets*. Widgets kan sammenlignes med brukergrensesnittkomponenter i for eksempel Java Swing biblioteket [71] eller GTK+ [28]. De mindre bibliotekene som det blir fokusert på i denne oppgaven, inneholder stort sett ikke widgets, men de kan derimot legges til gjennom plugins. Det finnes ingen standard for hva en widget er eller hva slags type widget et bibliotek skal støtte, men vanligvis finnes det widgets som en *dialogboks*, *trekkspill* (eng. *accordion*), *fremdriftsindikator* (eng. *progressbar*) og *faner* (eng. *tabs*) i et typisk AJAX-bibliotek. Eksempel på to forskjellige widgets fra biblioteket *jQueryUI* (se avsnitt 6.4.3) vises i figur figur 6.3. Til venstre er det en dialogboks, og til høyre er det et trekkspill. Begge widgetene finnes i en alternativ implementasjon for *Dojo* (se avsnitt 6.4.2).

Figur 6.3: Eksempler på widgets.

6.4 Evaluering av AJAX-biblioteker

Tre forskjellige direkte AJAX-biblioteker vil nå bli gjennomgått. Oppgaven vil se på *Prototype*, *Dojo* og *jQuery*, og undersøke hvilket som skal brukes i resten av denne masteroppgaven. De tre er valgt med bakgrunn i deres popularitet, store fellesskap, og fordi de har hatt flere år med utvikling bak seg. Flere andre biblioteker kunne også vært aktuelle, slik som *MooTools* [78], *The Yahoo! User Interface Library (YUI)* [58] og *qooxdoo* [3].

6.4.1 Prototype

Prototype [84] er et av de mest utbredte AJAX-bibliotekene som finnes. Dette på grunn av sin enkle syntaks, og sin nære forbindelse med det svært populære Ruby-rammeverket *Ruby on Rails* [47]. Prototype sin basispakke inkluderer metoder for å:

- Gjøre XHR enklere.
- Velge elementer fra en DOM-treet til en internettside.
- Forenkle objektorientert programmering i JavaScript.

I tillegg er Prototype nært knyttet opp mot det grafiske biblioteket script.aculo.us [42] som tar seg av *dra-og-slipp* (eng. *drag-and-drop*) og animasjoner. Dra-og-slipp er en funksjonalitet som gjør det mulig å bruke musen til å klikke og holde nede museknappen på et objekt, for så å dra objektet til en annen plassering, og der slippe museknappen. Funksjonen brukes mye i moderne operativsystemer. For eksempel brukes den til å flytte filer mellom mapper i blant annet *Microsoft Windows* og *Apple MacOS*. Biblioteket *script.aculo.us* har et ganske begrenset widget-bibliotek, der den eneste widgeten i skrivende stund er en autofullførtekstboks.

Prototype er ikke særlig omfattende, og forsøker i første omgang kun å gjøre vanlig JavaScript-utvikling enklere. Dette biblioteket var blant de første mer populære bibliotekene, men mangler likevel industristøtte. Prototype brukes derimot aktivt i lærebøker om AJAX, siden man da lærer bort språket JavaScript, og ikke spesialiserer seg på et bestemt bibliotek.

Blant brukerene av Prototype finnes blant andre datamaskinfabrikanten *Apple* og det sosiale nettstedet *last.fm*.

6.4.2 Dojo

Selv om Prototype er utvidbart, er det ikke laget med tanke på modularitet på samme måte som Dojo [35]. Kjernedelen av Dojo er på kun 28 kilobyte, og inneholder stort sett de samme funksjonene som Prototype, slik som elementvelger og XHR-innkapsler. Kraften til Dojo kommer derimot først når man utvider det med widgetsystemet *Dijit* [34]. Dette er et intrikat system som inkluderer en mengde widgets, som blant annet:

- Trekkspill
- Datovelger
- Dialogboks
- Faner
- Slider
- Avkryssningsfelt
- Meny (både popup og vanlig)
- Verktøylinje
- Riktekst-editor

Dojo er godt dokumentert, og har bred industristøtte fra blant andre IBM og Oracle.

Eksempler på sider som bruker Dojo-biblioteket er karttjenesten MapQuest [57] og epost-tjenesten til AOL [51].

6.4.3 jQuery

jQuery er et av de mest populære bibliotekene i dag, og får støtte av blant andre Mozilla Foundation og Microsoft. Dette biblioteket ligger tett opp mot Prototype funksjonsmessig, og har høy fokus på elementutvelging fra et DOM-tre. Noe av det som gjør jQuery unikt, er bruken av et spesielt jQuery JavaScript-objekt. Dette objektet blir returnerert fra alle jQuerys metoder. På grunn av dette kan alle jQuery-metodene brukes direkte på returverdien til andre jQuery-metoder. Dette kalles kjeding, og listing 6.1 viser et eksempel på dette. jQuery har et eksternt, men likevel fullt integrert, widget-bibliotek kalt *jQueryUI*. Dette biblioteket inkluderer høykvalitetsimplementasjoner av følgende widgets:

- Trekkspill
- Datovelger

Bibliotek	Filstørrelse
jQuery 1.3 + jQuery UI	244
Prototype 1.6 + script.aculo.us	267
Dojo 1.3.1 + Dijit	3 316

Tabell 6.1: Størrelse på de tre aktuelle bibliotekene. Alle tall er oppgitt i kB.

- Dialogboks
- Faner
- Slider
- Prosessindikator

Selv om jQueryUIs widget-bibliotek ikke kan sammenlignes med Dojo sitt i omfang, har biblioteket veldig god støtte for plugins. Dette har ført til at det eksisterer et stort utviklermiljø rundt biblioteket som legger til ny funksjonalitet. På grunn av dette miljøet er også dokumentasjonen god. jQuery har i mindre grad også blitt brukt i DPG 2.0 tidligere. Eksempler på sider som bruker jQuery er Digg.com, Mozilla.com og Google.com.

6.4.4 Valg av bibliotek

Krav til bibliotek

Siden det er meningen at AJAX-biblioteket skal brukes i mange forskjellige presentasjoner og presentasjonsmønstre, var det viktig å bruke tid på å evaluere de forskjellige bibliotekene grundig. Kravene som ble satt til bibliotekene var de følgende:

- Åpen og fri kode.
- Utvidbarhet. Det er viktig å ha et bibliotek som kan utvides ved hjelp av plugins.
- Fornuftig størrelse. Siden mange av presentasjonene vil bruke en mindre del av bibliotekene, er det ikke ønskelig at klienten er avhengig av å laste ned biblioteker med stor filstørrelse.
- Tilfredsstillende respons, slik at brukeropplevelsen føles raskere.
- Funskjonsmessig må biblioteket kunne oppnå målene i avsnitt 1.2 enten ved hjelp av kjernefunksjonalitet eller gjennom plugins.

Størrelse

Bibliotekene har forskjellig mengde funksjoner og størrelsen til bibliotekene er derfor noe vanskelig å sammenligne. I de tre bibliotekene som er vurdert, har Prototype minst funksjonalitet, mens Dojo har mest. Likevel er det viktig å tenke på filstørrelsen, da ikke alle

	Test 1	Test 2	Test 3	Test 4	Test 5	Gjennomsnitt
jQuery 1.3	1689	1677	2288	1686	2070	1882,00
Prototype 1.6	2498	2501	2500	2892	2842	2646,60
Dojo 1.3.1	603	593	610	611	611	605,60

Tabell 6.2: Resultater fra fem runder med den modifiserte utgaven av SlickSpeed. Alle tider er oppgitt i millisekunder.

brukere har bredbåndstilgang. Det er for eksempel en økende trend å bruke mobiltelefonen til å få tilgang på internett. Alle bibliotekene som synes nødvendig er tatt med i sammenligningen, og ikke kun kjernebibliotekene. Resultatene er vist i tabell 6.1. Mens Prototype og jQuery har all koden i to relativt små filer, har Dojo spredd koden sin ut over over 400 forskjellige filer. Det betyr at nettleseren ikke laster ned alle megabytene, men bare det den trenger. Samtidig vil det også si at det blir svært mange HTTP-forespørsler, noe som vil senke hastigheten når man bruker Dojo [81].

Hastighet

Fordi manipulering av elementer i DOM-treet er en av de mest brukte operasjonene på en AJAX-drevet side, er det viktig at bibliotekene utfører disse operasjonene så raskt som mulig. Det er ønskelig å øke interaktiviteten og responstiden til en nettside, og det er derfor viktig at nettsiden er tilgjengelig så fort som mulig.

For å teste hurtigheten til bibliotekene, er det blitt brukt en modifisert versjon av verktøyet SlickSpeed [79]. Dette verktøyet kjører den mest brukte DOM-manipuleringsoperasjonen, *element-velging*, i fem forskjellige bibliotek, og rapporeter tilbake hvor lang tid operasjonen tar. SlickSpeed er ikke oppdatert til å bruke de siste versjonene av bibliotekene, og verktøyet er derfor modifisert til å ta i bruk de i skrivende stund nyeste. Ved å kjøre igjennom alle testene fem ganger, er gjennomsnittstiden til de tre bibliotekene funnet. Resultatene vises i tabell 6.2. De to siste bibliotekene som er med i SlickSpeed-testen er *MooTools* og *YUI*. Disse bibliotekene er ikke med i evalueringen, og det er ikke funnet grunnlag for å lage støtte for disse bibliotekene i den modifiserte versjonen av SlickSpeed. Resultatene fra dem vil heller ikke bli gjennomgått her.

Selv om hastigheten til alle bibliotekene er respektable, er det liten tvil om at Dojo kommer ut langt bedre en konkurrentene.

Funksjoner

Funksjonsmessig har de tre bibliotekene svært forskjellig forankring. Som nevnt forsøker Prototype å nå serverside-utviklere ved å gjøre JavaScript mer egnet til blant annet objektorientering. Mange velger Prototype på grunn av dette alene. I forhold til widgets og effekter ligger Prototype et stykke bak, og har ikke helt den samme utviklingsgruppen rundt seg som

	jQuery+jQueryUI	Dojo+Dijit	Prototype+s.a.u*
Elementvelger	☒	☒	☒
XMLHttpRequest	☒	☒	☒
Dra og slipp	☒	☒	☒
Unittest rammeverk	☒	☐	☐
Animasjoner/Effekter	☒	☒	☒
Layoutfunksjonalitet	☐	☒	☐
Støtter plugins	☒	☒	☒

Tabell 6.3: Et utvalg av funksjoner i de forskjellige bibliotekene. *script.aculo.us

de to andre.

Dojo er veldig omfattende, og byr på det meste en utvikler trenger av funksjoner. Dojo inkluderer mange spesialiserte funksjoner, slik som funksjonen som automatisk genererer URL-er ut fra HTML-skjemaer. Dette er nyttige funksjoner som gjør mange vanlige oppgaver mindre utsatt for feil.

jQuery har sin styrke i at det bruker et spesielt jQuery objekt. Dette objektet returneres av alle funksjoner, og derfor kan funksjoner bli satt i en kjede. For eksempel vil listing 6.1 først velge alle `div` elementer på siden, så vil den fjerne alle elementer som bruker CSS klassen `blue` fra mengden. Etter dette vil den så legge til alle elementer som er av typen `h1` før den så itererer over alle de innsamlede elementene, og skriver ut innholdet i dem via en dialogboks.

Listing 6.1: Eksempel på kjeding av funksjoner i jQuery

```
jQuery('div').not('.blue').add('h1').each(
  function(){
    alert(this.innerHTML)
  }
);
```

For å sammenligne funksjonalitet er det satt opp to matriser. Den første (tabell 6.3), viser basisfunksjonalitet. Tabellen viser at bibliotekene støtter mange av de samme funksjonene. Den andre tabellen (tabell 6.4) viser hvilke widgets som finnes. Det er her langt større forskjell. Likevel er det viktig å se hva slags widgets som er mest essensielle, og mange av de ekstra widgetsene som Dojo har i forhold til de andre, passer bedre til internettsider som forsøker å erstatte vanlige skrivebords applikasjoner enn til tradisjonelle internettsider som har som formål å vise frem informasjon (jf. verktøylinje). For mer informasjon om widgetene, henvises det til API dokumentasjonen til de aktuelle bibliotekene [36] [62] [83].

	jQuery+jQueryUI	Dojo+Dijit	Prototype+s.a.u*
Faner	☒	☒	☒
Slider	☒	☒	☒
Autofullfør	☒	☐	☒
Trekkspill	☒	☒	☐
Datovelger	☒	☒	☐
Dialogboks	☒	☒	☐
Prosessindikator	☒	☒	☐
WYSIWYG-teksteditor	☐	☒	☐
Menyer	☐	☒	☐
Fargevelger	☐	☒	☐
Tremenyer	☐	☒	☐

Tabell 6.4: Widgets som eksisterer i de forskjellige bibliotekene. *script.aculo.us

6.4.5 Konklusjon

I dette kapittelet har tre forskjellige AJAX-biblioteker blitt gjennomgått. Det har blitt lagt vekt på forskjellene mellom bibliotekene, siden de tre har mye overlappende funksjonalitet.

Prototype er med sin objektorientering og lille filstørrelse veldig attraktiv, men siden DPG i første omgang ikke skal bli en ren AJAX-drevet applikasjon, er ikke objektorientering utslagsgivende i valget. Prototype er heller ikke veldig rask, og har et mindre utviklermiljø rundt seg en de to andre.

Dojo frister med sitt omfattende bibliotek og sin raske AJAX-motor, og ville absolutt vært interresant for bruk i DPG. Likevel blir det mindre hensiktsmessig å ha et så stort bibliotek når man sannsynligvis bare vil bruke en liten brøkdel av funksjonaliteten. Måten DPG i skrivende stund håndterer ressurser på, gjør også dette biblioteket noe tungvindt å bruke.

jQuery gjør det svært lett å legge til AJAX-funksjonalitet til eksisterende sider. Den er gjennomsnittlig når det gjelder hastighet, men har til gjengjeld en godt gjennomtenkt og utvidbar arkitektur. jQuery er også lite i filstørrelse, og alt er inkludert i tilsammen to filer. jQuery har videre god industristøtte. På bakgrunn av dette vil jQuery bli brukt videre i denne masteroppgaven for utvikling av rike klienter i DPG 2.1.

7 Serverløsning

7.1 Innledning

AJAX trenger grovt sett to komponenter for å fungere: Klientstøtte og serverstøtte. I dette kapittelet vil den sistnevnte komponenten bli gjennomgått.

Selv om en AJAX-drevet side kan benytte seg av statiske ressurser, vil ikke brukeropplevelsen bli så mye forandret før også ressursene blir dynamiske. Et eksempel på dette kan man hente fra fotballnettsider: En statisk ressurs med resultater fra fotballkamper kan være nyttig, men siden blir ikke interaktiv og spennende før denne ressursen blir oppdatert dynamisk etter hvert som et lag scorer mål.

I dette kapittelet skal det undersøkes hvordan serverstøtte for AJAX kan utvikles, spesielt med tanke på DPG. Det er viktig å bemerke at DPG i skrivende stund ikke har støtte for interaksjon fra bruker. Det vil derfor kun bli fokusert på hvordan serveren kan gi fra seg informasjon til klienten, og ikke omvendt. Resultatet av denne undersøkelsen kan derimot også ha betydning for hvordan en fremtidig løsning, der det kommuniseres begge veier, kan bli designet.

7.2 Forslag til AJAX støtte i DPG

Det er i denne oppgaven utarbeidet tre forskjellige forslag for hvordan man kan få serverstøtte for AJAX i DPG. Alle forslagene vil nå bli gjennomgått, og det vil deretter bli foretatt en evaluering av dem. Dette slik at det er mulig å velge hvilken løsning som skal bli brukt i resten av oppgaven.

7.2.1 Forslag 1: AJAX-utsnitt

AJAX-utsnitt forslaget bygger på hvordan *utsnitt* (eng. *views*) blir rendrert i DPG.

For å forstå dette forslaget, må man først forstå prosessen til hvordan en side blir komponert i DPG. Denne prosessen vil derfor bli gjennomgått i det følgende:

En side i DPG består av en mengde forskjellige utsnitt. For eksempel består forsiden til Kursmønsteret av utsnittene `main-menu`, `menu`, `messages` og `weekplan`. Hvert av disse utsnittene er koblet mot innhold (*entitetsinstanser*) og en transformasjon. Transformasjonen er i DPGs tilfelle et *XSLT* (*Extensible Stylesheet Language Transformations*)-dokument [93].

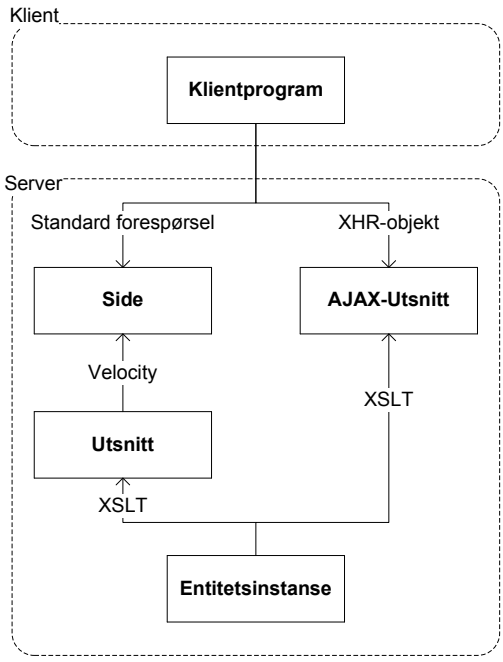
XSLT er et språk som brukes til å transformere et XML-dokument slik at det blir et annet XML-dokument. For eksempel kan det transformere et XHTML-dokument til å bli et WML-dokument [65], slik at det kan leses av mobiltelefoner. I DPG brukes XSLT-transformasjonene til å transformere en entitetsinstanse til å bli XHTML. Listing 7.1 viser et eksempel på hvordan XML-en til en entitetsinstanse kan se ut.

Siden det er mulig å transformere et utsnitt til XHTML, er det også mulig å transformere det til et annet XML-dokument som et JavaScript-skript lettere kan tolke, som for eksempel *Atom*-formatet som blir vist i listing 7.9.

Etter at utsnittene til en side er transformert til XHTML, blir alle sammen satt sammen til et felles dokument ved hjelp av en mal. Denne malen kalles *hovedmalen* (eng. *master template*), og er lik for alle sidene til en presentasjon. Verktøyet som brukes til å sette utsnittene inn i malen kalles *Apache Velocity* [33].

For å få dette forslaget til å fungere må det utvikles en mekanisme for å avslutte rendreringsprosessen etter at utsnittet er transformert. Dette må gjøres fordi rendreringsprosessen ellers vil sette utsnittet inn i en mal. Figur 7.1 viser hvordan klienten enten kommuniserer den tradisjonelle veien, og da går via *side*, mens den også kan gå via *AJAX-utsnitt* hvis den er interresert i å hente data i et format som JavaScript lettere kan tolke.

Dette forslaget har den fordelen at innholdet i det transformerte dokumentet til en hver tid vil være synkronisert med det faktiske innholdet, uten behov for hendelseshåndtering (se avsnitt 7.4.5). Det vil også bli en meget fleksibel løsning siden man har muligheten til å generere utdata i hvilket som helst format. Løsningen vil også fungere fint med fremtidige formater som enda ikke er påtenkt. Den vil også være godt integrert med fremtidige utviklingsverktøy, siden utviklingsmetoden vil være relativt lik måten utvikleren forholder seg til resten av



Figur 7.1: Bruk av views for å generere innhold som kan brukes av klientens AJAX (jf. 5.1). Pil angir dataflyt.

presentasjonsmønsterutviklingen.

Ulempene til forslaget er at det ikke finnes noen mulighet til å flette sammen data fra forskjellige utsnitt. Dette er fordi alle utsnittene på en side er uavhengige av hverandre (se avsnitt 3.4.3). Det er heller ingen mulighet for å inkludere data fra eksterne datakilder. Forslaget gjør også at det blir vanskelig for mønsterforfatteren å utvikle presentasjoner, siden vedkommende må kunne spesifikasjonen til utvekslingsformatet i tillegg til HTML, CSS og XSLT. På grunn av dette er det også muligheter for feil i utvekslingsformatet.

Det største problemet med AJAX-utsnitt-forslaget er likevel at det vil bli en inkonsekvens i rendreringsprosessen. PV-en må vite om et utsnitt skal sendes tilbake for å brukes med AJAX-komponenter, eller om det skal settes sammen til en komplett side ved hjelp av templater. Det vil være mulig å unngå dette ved å gjøre løsningen om fra *AJAX-utsnitt* til *AJAX-side*. Dette innebærer at en bruker spesielle maler for sidene som skal produsere data til bruk av AJAX-klienter, men dette vil føre til enda mer inkonsekvens i rendreringsprosessen ved at hovedmalen også må overkjøres.

7.2.2 Forslag 2: Direkte tilgang til presentasjon

DPG er en applikasjon som benytter XML i svært stor grad. Både strukturen, transformasjonen, og innholdet blir lagret i XML. Siden XML også er et av hovedformatene for

å utveksle data mellom server og klient på en AJAX-drevet internettside, faller det neste forslaget naturlig.

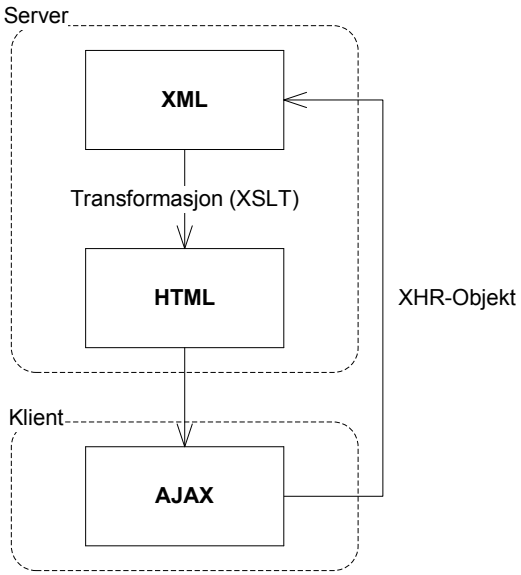
Programmet på klienten vil i dette forslaget aksessere XML-data som finnes på serveren i den formen den foreligger før den blir transformert til HTML. Listing 7.1 viser et eksempel på hvordan data som blir utvekslet mellom serveren og AJAX-klienten kan se ut. Figur 7.2 viser hvordan XML-en på serveren blir transformert til HTML ved hjelp av XSLT. Denne HTML-en, som inkluderer JavaScript kode, blir så sendt over til klienten. På klientsiden vil så nettleseren kjøre JavaScriptet, som igjen aksesserer XML-en på serveren ved hjelp av XHR-objekter.

Listing 7.1: Eksempel pa ren XML fra server (forenklet)

```
<?xml version="1.0" encoding="UTF-8"?>
<bookListEntity type="rootEntity"
         createdBy="jafutest" updatedBy="jafutest"
         createdOn="18/06/2008 15:35">
  <books type="entity-list">

    <bookEntity type="subEntity"
             createdBy="jafutest" updatedBy="jafutest"
             createdOn="18/06/2008 15:35">
      <title>test tittel</title>
      <chapters>1-7, 8-13</chapters>
    </bookEntity>

    <bookEntity type="subEntity"
             createdBy="jafutest" updatedBy="jafutest"
             createdOn="18/06/2008 16:35">
      <title>Java som forste programmeringssprak</title>
      <chapters>hele boken</chapters>
    </bookEntity>

  </books>
</bookListEntity>
```

Direkte tilgang til presentasjon har mange fordeler. For det første er det her mulig å lage meget avanserte AJAX-sider siden klienten får tilgang til all data. I tillegg til dette er det lite som må gjøres med DPG for å få den ønskede funksjonaliteten. Likevel er det også noen store ulemper. Blant de største er problemet med sikkerhet: Siden alt som kan leses av klientens program, også kan leses av som ren tekst, vil det være lett for en datasnok å finne et presentasjonsmønsters interne struktur, noe som ikke nødvendigvis er ønskelig. Det er for eksempel ikke alltid ønskelig at hvem som helst skal kunne se brukernavnet på den brukeren som har opprettet en melding.

Et annet problem er at det blir vanskelig å gjenbruke AJAX-komponenter i forskjellige mønstre siden XML-en kan være forskjellig avhengig av hvordan mønsteret er definert. For eksempel kan det tenkes at navnet til en person er definert som `navn` i et mønster, og `name` i et annet.



Figur 7.2: Mulig løsning for tilgang til data fra server. Pil angir dataflyt.

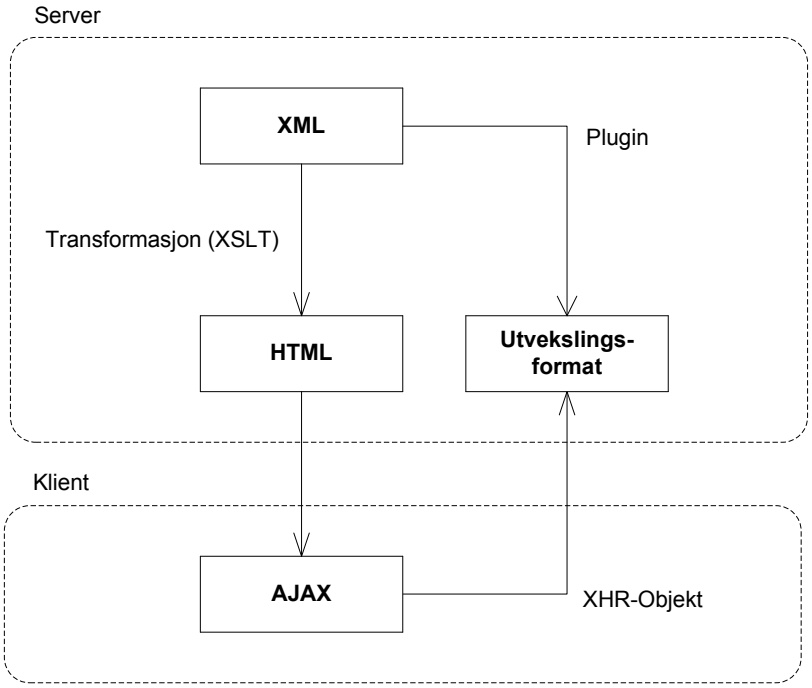
7.2.3 Forslag 3: Indirekte tilgang til presentasjon

Det tredje forslaget har en litt annen innfallsvinkel: I stedet for at AJAX-programmet på klienten kan kommunisere direkte med presentasjonsmønsterets XML, aksesserer programmet en generert ressurs som avbilder innholdet til presentasjonen. Denne ressursen vil i DPG sitt tilfelle kunne bli generert som en fil ved hjelp av en plugin. Mønsterforfatteren vil da kunne velge hvilken informasjon som skal bli sendt til klienten. For eksempel vil det være mulig å legge ved brukernavnet, eller ta det bort.

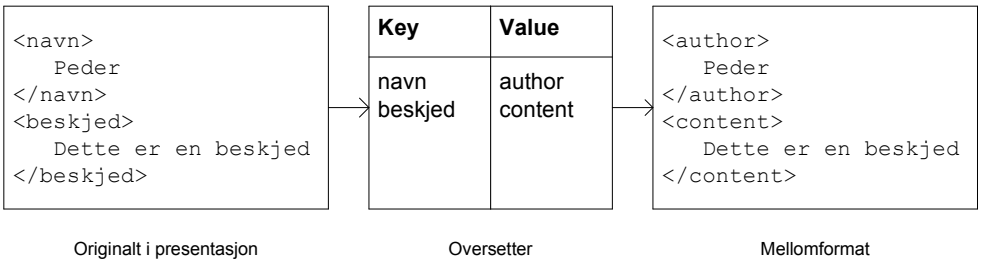
Figur 7.3 viser hvordan dette forslaget vil fungere: XML-en på serveren blir transformert til HTML ved hjelp av XSLT. HTML filen inkluderer JavaScript. Etter at dette er gjort, vil en plugin i DPG transformere XML-en slik at den foreligger i et *utvekslingsformat*. Når dette er utført vil serveren sende den genererte HTML-filen over til klienten, der JavaScriptet blir eksekvert. JavaScriptet lager så XHR-objekter som det bruker til å aksessere ressursen (*utvekslingsformat* i figur 7.3).

Fordelene til denne metoden er at sikkerheten blir ivaretatt, siden man kan skjule informasjon som vanlige brukere ikke skal ha tilgang til. I tillegg til dette kan den genererte ressursen foreligge i et forutsigbart format som gjør at AJAX-komponentene lett kan gjenbrukes i forskjellige situasjoner. Dette kan gjøres ved at en plugin kan *avbilde* (eng. *map*) mellom et *felt* i presentasjonen, og et fastsatt ord i ressursen. For eksempel er det mulig for pluginen å si at *navn* i presentasjonen er det samme som *author* i ressursen (se figur 7.4).

Den store ulempen er at det vil ta ekstra tid å generere denne ressursen. Både det å skrive den til disk, og det å utføre denne mappingen krever noe mer tid en de to andre forslage-



Figur 7.3: AJAX-skriptet har indirekte tilgang til presentasjonsmønsteret. Pil angir dataflyt.



Figur 7.4: Avbilding mellom presentasjon og mellomformat

ne. Dette motargumentet vil bli svekket ved bruk av effektive cache-løsninger. Det andre motargumentet er at dette forslaget vil kreve mer kode på serversiden en de andre forslagene. Det er derimot, som tidligere nevnt, mulig å gjøre dette ved hjelp av den oppdaterte pluginarkitekturen. På grunn av dette vil ikke forslaget gjøre kjernen til DPG større.

7.2.4 Evaluering

For å finne ut hvilket forslag det skal jobbes videre med, vil de tre forlagene bli evaluert.

Forslag 2, *Direkte tilgang til presentasjon*, kan ikke brukes. Grunnen til dette er at det er for stor risiko å vise frem hele presentasjonensmønsteret med blant annet brukernavnet til forfatteren og tidpunkt. Det hadde vært mulig å sensurere disse dataene, men det ville samtidig føre til mindre fleksibilitet, siden disse dataene kan være til stor nytte i noen tilfeller. Enkelte deler av en presentasjon kan også være skjult, for eksempel fremtidige oppgaver, og det er da ikke fornuftig å vise frem disse delene heller.

Forslag 1, *AJAX-utsnitt*, er et forslag som godt kunne vært implementert. Forslaget har mange gode egenskaper, spesielt hvordan det bruker den eksisterende konseptuelle modellen, samt de gode verktøyene og teknologiene. De største problemene med dette forslaget henger sammen med hvordan hele DPG er implementert:

- Alle utsnitt er helt uavhengige av hverandre. Det er dermed ikke mulig å flette sammen innhold fra flere forskjellige utsnitt. Det er mulig å gjøre denne operasjonen på klientsiden, men det bør være serveren sin oppgave. Det er for eksempel ikke alle klienter som har støtte for dette (for eksempel strømleseren til Mozilla Firefox).
- XSLT er et verktøy for å transformere XML dokumenter, og ikke et programmeringsspråk (selv om det er turing-komplett). Det har ikke kraften til å for eksempel hente inn eksterne kilder.

I tillegg til dette vil denne løsningen heller ikke gjøre det enklere å produsere AJAX-drevne presentasjoner. Dette veier også tungt siden denne masteroppgaven er interessert i å fremme bruk av AJAX.

Forslag 3, *Indirekte tilgang til presentasjon*, kommer til å bli brukt videre i masteroppgaven. Det er det ingen større ulemper med dette forslaget. Det kreves en del mer programmering en i de andre forslagene, men samtidig får man den mest fleksible, og mest gjenbrukbare løsningen. Den vil ikke kreve mye av presentasjonsmønsterforfatteren og vil, ved hjelp av en god implementasjon, gjøre det mulig å flette sammen flere datakilder. På grunn av at den må lagre ekstra filer på serveren, vil løsningen dessverre kreve litt mer av maskinvaren. Dette er likevel mindre viktig, sammenlignet med de store fordelene knyttet til bruk av indirekte tilgang til presentasjon.

7.3 Dataoverføring

7.3.1 Nytt eller eksisterende format

For å sende data fra serveren til klientapplikasjonen bør dataformatet mellom de to systemene være klarert på forhånd. Se avsnitt 7.2.2 for grunnene til dette. Det står i utgangspunktet mellom to valg: Systemet kan bruke et eksisterende format, eller det kan bruke et format som er spesielt laget for dette systemet.

Fordelene ved å lage et nytt format er at det kan tilpasses til å omfatte alle spesialtilfellene som er ønskelig.

Det er derimot vanskelig å se for seg alle bruksområdene et format skal håndtere i all fremtid, og det er i tillegg krevende å designe og vedlikeholde et nytt format. SOAP [95] sitt utvekslingsformat, som er et filformat som er laget med tanke på å kommunisere mellom internettjenester, er også utelukket fordi det ikke har har ferdig klarerte XML-elementer.

Det andre valget er å bruke et allerede eksisterende format. Ved å bruke et slikt format vil man miste fleksibiliteten som man kunne fått ved å designe et eget. Samtidig trenger man ikke bruke tid på å designe og vedlikeholde formatet. Det er også større muligheter for at man finner kompatible tredjepartsmoduler, noe som vil føre til at det er lettere å bygge ny funksjonalitet inn i systemet.

Det vil i det følgende bli benyttet et eksisterende format. Dette er fordi det ikke synes mulig å produsere et perfekt format for alle scenarier. Kompatibilitet med andre tjenester, som for eksempel nyhetsformidlere og værmeldinger, vil heller ikke oppnås ved å lage et nytt format, noe som vil være en stor ulempe.

7.3.2 Valg av utvekslingsspråk

I teorien er det mulig å utveksle data mellom server og klient på en AJAX-drevet nettside ved hjelp av hvilket som helst tekstdokument. Det er for eksempel mulig å sende vanlig HTML eller ren tekst. Likevel er det i praksis stort sett to forskjellige språk som brukes: XML og JSON [1].

XML er, som tidligere nevnt, et språk som brukes mye i DPG. Listing 7.2 viser hvordan et navn kan bli representert ved hjelp av XML.

Listing 7.2: Eksempel på XML

```
<person>
  <fornavn>Peder</fornavn>
  <etternavn>Skeidsvoll</etternavn>
</person>
```

JSON står for *JavaScript Object Notation*. Dette språket bruker JavaScript sine *nøkkelverditabeller* (eng. *hashmap*) til å representere data. Figur 7.3 viser hvordan samme data som ble representert i XML i listing 7.2 kunne blitt representert i JSON.

Listing 7.3: Eksempel på JSON

```
{
  "fornavn": "Peder",
  "etternavn": "Skeidsvoll"
}
```

Siden JSON i grunnen bare er en smart måte å utnytte JavaScript sin nøkkelverditabell-notasjon, er denne notasjonen meget rask på klientsiden. XML må derimot håndteres ved å traversere XML-ens DOM-tre, noe som både er mer tidkrevende, og vanskeligere for å programmere mot.

På serversiden er det derimot andre veien. DPG har god innebygget støtte for XML gjennom biblioteket JDOM, men ingen støtte for JSON.

Selv om det finnes flere gode verktøy for å bruke JSON med Java (for eksempel [2]), synes det best å holde seg til XML. Dette både fordi det finnes langt flere XML-baserte utvekslingsformater, som for eksempel de populære formatene Atom og RSS (se neste delkapittel), og fordi det er mer populært blant klienter å støtte XML baserte språk. For eksempel har nettleserene Apple Safari og Mozilla Firefox innebygget støtte for å bruke Atom og RSS i sine brukergrensesnitt.

XML vil derfor bli valgt som språk for utvekslingsformatet.

7.3.3 Valg av format

For å velge format er følgende krav satt opp i prioritert rekkefølge:

1. Støtte de fleste scenarier i Kursmønsteret i DPG.
2. Åpent og veldefinert format.
3. Bred industriell støtte.

For å vurdere om et format følger krav 1, er det nødvendig med scenarier fra kursmønsteret. Følgende scenarioer er tenkt ut:

- Visning av meldinger. Felt: Overskrift, meldingskropp og tidspunkt.
- Billedgalleri. Felt: Adresse (URL) til bildefilene, billedtekst.
- Visning av siste forumposter. Felt: Tittel på posten, addresse til posten og tidspunkt.
- Kobling av frister (for eksempel innleveringer) mot kalender. Felt: Tittel, tidspunktet for fristen og ekstrainformasjon.

To formater synes å oppfylle kravene kravene nevnt over. Disse formatene er *Really Simple Syndication (RSS) 2.0* [7] og *Atom 1.0* [4]. Dette er formater som orginalt er tiltenkt enkel syndikering av nyhetsartikler til eksterne nyhetslesere. Bortimot alle nyhetssider, som nasjonale Dagbladet og VG, og internasjonale medier som BBC og CNN, tilbyr nyhetene sine i et slikt format. Formatene er relativt like, og har stort sett bare overlappende funksjonalitet. En tabell med de mest relevante elementene i Atom og RSS finnes i tabell 7.1. For mer informasjon om formatene refereres det til de respektives dokumentasjon.

RSS har også en rekke med elementer som egentlig ikke er nødvendige, for eksempel `cloud` *(nettsky)* og `rating` *(PICS [91] rangering av strømmen)*, og andre som standard XML hadde løst: For eksemepel er elementet `language` unødvendig siden man har attributtet `xml:lang`.

Begge formatene vil håndtere scenarioene nevnt ovenfor, og siden de er såpass like, er det vanskelig å konkludere på spørsmålet om hvilket format som er best på dette området.

På krav 2 skiller formatene seg likevel litt mer. RSS er ikke standardisert, og har ikke noe XML-skjema for validering. Et slikt skjema har derimot Atom. Atom er også standardisert av Internet Engineering Task Force som RFC 4287 [24].

RSS er uten sammenligning det mest brukte formatet på internett, og brukes av stort sett alle nyhetstilbydere. Krav 3 er det derfor RSS som oppfyller mest tilfredsstillende.

Likevel synes det viktigere at formatet er gjennomtenkt og standardisert, enn at det er mest brukt. Dessuten er det relativt enkelt å konvertere en fil fra RSS til Atom, og vica versa.

Valget har derfor falt på Atom.

7.4 Utvikling

7.4.1 Rammeverk for strømsyndikering

Som forklart i avsnitt 7.2.4, er en løsning bygget på DPG sin pluginarkitektur valgt. På grunn av dette valget kan denne løsningen også skilles fra DPGs kjernearkitektur.

For å utvikle plugins som kan oppfylle de oppsatte kravene, er det nødvendig med et bibliotek for behandling av strømmer. Kravene som er satt til dette biblioteket, er at det skal:

- Tilby et programmeringsgrensesnitt for å interaktere med strømmer.
- Kunne lese inn Atom-strømmer.
- Kunne produsere Atom-strømmer.

To biblioteker synes å oppfylle disse kravene: Apache sitt Abdera [26] bibliotek, og ROME [85].

RSS 2	Atom 1.0	Forklaring
author	author	Epostadressen til forfatteren
category	category	Kategorisere strømmen
channel	feed	Informasjon om strømmen
cloud	-	
comments	-	
copyright	rights	
description	subtitle	description er delt opp i subtitle...
description	summary/content	...og summary eller content
docs	-	
enclosure	-	Mediafil. Atom bruker rel="enclosure"
generator	generator	
guid	id	
image	logo	Bilde som identifiserer strømmen
item	entry	Definerer en artikkel i strømmen
language	-	
lastBuildDate	updated	
link	link	URL til artikkelen.
managingEditor	author/contributor	Forfatter av artikkel.
pubDate	published	Når artikkelen ble opprettet.
rating	-	
rss	-	Rotelementet til en RSS-strøm.
skipDays	-	
skipHours	-	
source	-	
textInput	-	
title	title	Tittel på artikkel eller strøm.
ttl	-	
webMaster	-	
-	contributor	
-	icon	Ikonet til strømmen
-	source	

Tabell 7.1: Sammenligning av utvalgt funksjonalitet i RSS 2.0 og Atom 1.0.

Abdera er et bibliotek som kun fokuserer på Atom-strømmer, og som derfor ikke fungerer med andre formater som for eksempel de forskjellige RSS variantene. Abdera har derimot meget god og komplett Atom støtte, noe som selvsagt er viktig siden Atom-formatet er valgt for datautveksling. Teknisk sett holder Abdera en høy standard. Biblioteket kommer opprinnelig fra IBM, og har en relativt stor skare med frivillige utviklere. Hastigheten til Abdera er også upåklagelig på grunn av at *StAX (Streaming API for XML)* [17] benyttes til å tolke strømmer. Det finnes prosjekter for å få Abdera til å støtte RSS også, men disse er i skrivende stund ikke offisielt støttet. I listing 7.4 vises et eksempel på hvordan en Atom-strøm kan opprettes i Abdera.

Listing 7.4: Eksempel på bruk av Abdera

```
// Oppretter strommen
Abdera abdera = new Abdera();
Factory factory = abdera.getFactory();

Feed feed = factory.newFeed();

// Setter stromegenskaper
feed.setUpdated(new Date());
feed.setTitle("Tittel paa strommen");
feed.addLink("http://www.ii.uib.no");

// Legger til post til strommen
for(int i=1; i < 5; i++) {
  Entry entry = factory.newEntry();

  // Setter post egenskaper
  entry.setUpdated(new Date());
  entry.setTitle("Post " + i);
  entry.setContentAsXhtml("<em>Dette er innholdet til post "+i+"</em>");
  entry.addLink("http://xkcd.com/" + i);

  feed.addEntry(entry);
}
```

ROME sitt utgangspunkt er, i motsetning til Abdera sitt, å støtte RSS-formatet. Utviklingen har likevel ført til at biblioteket også støtter de to mest brukte Atom-standardene (Atom 0.3 og Atom 1.0) i tillegg til åtte forskjellige RSS formater (RSS 0.90, RSS 0.91 Netscape, RSS 0.91 Userland, RSS 0.92, RSS 0.93, RSS 0.94, RSS 1.0 og RSS 2.0). Rammeverket har god støtte for både Atom og RSS, men støtten for Atom er ikke like komplett som Adberas. ROME er opprinnelig et Sun prosjekt, men vedlikeholdes i skrivende stund av frivillige. ROME bruker JDOM til å tolke strømmer istedenfor STAX, og blir på grunn av dette betydelig tregere en Abdera. Likevel fører dette til at samme bibliotek som resten av DPG kan brukes. JDOM er også den eneste avhengigheten til ROME, noe som fører til mindre avhengigheter i DPG. Listing 7.5 viser et eksempel på hvordan en Atom strøm opprettes i ROME. Eksempelkoden gjør nøyaktig det samme som ble vist i Abdera i listing 7.4, men den bruker ROME i stedet for Abdera.

Listing 7.5: Eksempel på bruk av ROME

```
// Oppretter strommen
SyndFeed feed = new SyndFeedImpl();

// Setter stromegenskaper
feed.setTitle("Tittel paa strommen");
feed.setPublishedDate(new Date());
feed.setLink("http://www.ii.uib.no");

// Legger til post til strommen
List<SyndEntry> entries = new ArrayList<SyndEntry>();
for(int i=1; i < 5; i++) {
    SyndEntry entry = new SyndEntryImpl();

    // Setter post egenskaper
  entry.setPublishedDate(new Date());
  entry.setTitle("Post " + i);
  entry.setLink("http://xkcd.com/" + i);

  SyndContent content = new SyndContentImpl();
    content.setType("text/html");
    content.setValue("<em>Dette er innholdet til post " + i + "</em>");
    entry.setDescription(content);

  entries.add(feedEntry);
}
feed.setEntries(entries);
```

Begge bibliotekene inneholder funksjoner for å tolke og skrive ut strømmer, samt metoder for å manipulere strømmene abstrakt og formatuavhengig. I listing 7.5 og 7.4 vises det at strømmer defineres på en svært lik måte, selv om ROME benytter seg av flere abstraksjonsnivåer en Abdera (se spesielt på defineringen av innhold i en post).

Ut fra en samlet vurdering, er biblioteket ROME valgt for videre bruk i masteroppgaven. Spesielt er det lagt vekt på at ROME har god støtte for RSS. Dette er viktig da det er ønskelig at pluginene som blir utviklet skal kunne lese inn kilder fra eksterne servere, som gjerne kun tilbyr sitt innhold i form av RSS. I tillegg til dette er det positivt at biblioteket kun behøver JDOM for å fungere.

Strømmer og poster

Det vil nå bli gitt noen definisjoner på uttrykk som vil bli brukt videre i masteroppgaven.

Det engelske ordet *feed* er oversatt med det norske ordet *strøm*. En strøm er et generisk uttrykk for dataformater som tilbyr klienter lister over *poster*. Eksempler på strømmer er formatene Atom og RSS.

Poster blir på engelsk kalt *entries*. Strømmer inneholder som nevnt en liste med poster. For eksempel tilbyr dagbladet.no en strøm med sine 15 nyeste poster.

7.4.2 AJAX DPG-plugins

For å vurdere hva slags plugins som skal utvikles for systemet vil listen i avsnitt 7.3.3 fremdeles være relevant. Listen inkluderer nå krav til pluginen istedenfor til dataformatet:

1. Visning av meldinger. Krav til plugin: Hente ut XML-elementene fra presentasjonsmønsteret, og konvertere dem til Atom format for så å lagre denne filen på serveren.
2. Billedgalleri. Krav til plugin: Samme som i forrige punkt.
3. Visning av siste forumposter. Krav til plugin: Laste inn strøm fra ekstern side, eventuelt konvertere denne fra RSS til Atom, for så å lagre på server.
4. Kobling av frister (for eksempel innleveringer) mot kalender. Krav til plugin: Flette sammen forskjellige strømmer til én, og lagre denne på server.

Ut fra disse punktene er det konkludert med at to forskjellige plugins er nødvendige.

- En plugin som transformerer intern struktur i DPG til Atom, for så å lagre denne på serveren. Denne pluginen vil gjøre punkt 1 og 2 mulig.
- En plugin som fletter sammen forskjellige strømmer til en, der en strøm både kan være lokalt på serveren, eller fra en ekstern server. Da vil det å hente og lagre én ekstern side på lokal server være et spesialtilfelle, og vil derfor gjøre punkt 3 mulig. I tillegg vil denne pluginen i sammarbeid med den første gjøre punkt 4 mulig siden den da kan flette sammen strømmene fra for eksempel meldinger og innleveringer.

7.4.3 Pluginen List2Atom

`List2Atom` er det første pluginet som ble utviklet. `List2Atom` brukes på lister av entiteter, og vil for en presentasjonsmønsterutvikler se ut som en hvilken som helst annen liste. Forskjellen er at den også skriver en Atom-strøm til systemets persistenslag.

Det vil nå bli gjennomgått hvordan et `List2Atom`-plugin kan bli brukt i et presentasjonsmønster. Ved å bruke case-studyet *Kursmønsteret* kan man se på hvordan en Atom-strøm med meldinger kan lages.

Definering av entiteter

XML-strukturen til entiteten som skal omgjøres til en post i Atom strømmen må defineres. Dette gjøres i filen `pattern.xml`. Meldingen inneholder en overskrift samt en tekstkropp som rommer selve meldingen. Listing 7.6 viser hvordan deklarasjonen av entiteten kan se ut. Feltnavnene til meldingsoverskriften og meldingen er her satt til henholdsvis `header` og `messageBody`. Det er derfor valgt at overskriften skal være en uformatert streng, mens tekstkroppen skal kunne være formatert tekst i form av XHTML.

Listing 7.6: Eksempel på definisjon av en entitet som skal omgjøres til en Atom-post.

```
<entity id="messages">
  <field type="stringPlugin">header</field>
  <field type="xhtmlPlugin">messageBody</field>
</entity>
```

Det som så må gjøres, er å koble `List2Atom` plugin til denne entiteten. Det må også bestemmes hva *denne* instansen av `List2Atom` skal hete. Instansen vil her bli kalt `messagePlugin`. Listing 7.7 viser hvordan dette kan gjøres.

Listing 7.7: Eksempel på kobling av `List2Atom` pluginet mot entitet.

```
<entity id="meldingerEntity">
  <field type="messagePlugin" entity-id="message">messageList</field>
</entity>
```

For informasjon om hvordan resten av filen `pattern.xml` defineres henvises det til kapittel 5: *Ny presentasjonsmønsterspesifikasjon* og [80].

Spesifisering av plugin

Spesifisering av plugins blir, som skrevet om i kapittel 4: *Endringer i pluginarkitektur*, gjort i filen `pluginConfig.xml`. Her blir plugins gitt et navn og et sett med parametre. `List2Atom` benytter parametrene for å vite hva slags informasjon som skal plasseres hvor i Atom-strømmen. `List2Atom` kan bli definert som i listing 7.8.

Listing 7.8: Definering av plugin, og spesifisering av parametre.

```
<plugin-config id="messagePlugin" plugin-name="List2Atom">

  <!-- Setter tittel, beskrivelse og kategori pa strommen -->
  <param name="feedTitle">Meldinger</param>
  <param name="feedDescription">Meldinger i INF399</param>
  <param name="feedCategory">Meldinger</param>

  <!-- Forteller List2Atom hva strommen skal lagres som -->
  <param name="feedFileName">meldinger.xml</param>

  <!-- Avbilder tittelen til posten som feltet header og -->
  <!-- beskrivelsen av posten som feltet messageBody.    -->
  <param name="feedEntryTitle">/header</param>
  <param name="feedEntryDescription">/messageBody</param>

</plugin-config>
```

Den første linjen definerer her at `messagePlugin` er av plugintypen `List2Atom`. Etter dette defineres en rekke parametre. Det første parametret (`feedTitle`) forteller hva navnet

på Atom-strømmen skal være, mens det andre (`feedDescription`) gir en kort beskrivelse av strømmen. Det tredje (`feedCategory`) forteller hva strømmen skal kategoriseres som. I det fjerde parameteret (`feedFileName`) skal man fortelle `List2Atom` hva filnavnet til Atom-strømmen skal være på serveren. De to neste parametrene (`feedEntryTitle` og `feedEntryDescription`) er viktige fordi de definerer hva som skal stå i `title` og `description` feltene til hver post i strømmen. Legg merke til at det skrives en skråstrek foran navnene på feltene. Dette gjøres for å fortelle `List2Atom` at det skal gi feltene verdien til elementene som har dette navnet. Hvis skråstreken hadde blitt utelatt ville alle postene derimot fått tittelen *header* i dette tilfellet. Figur 7.5 viser et forenklet sekvensdiagram over hvordan pluginet `List2Atom` bruker parametere som er satt i nøkkelverditabellen `ParameterMap` til å avbilde mellom elementer i filen `pattern.xml`, og hva slags elementer det skal være i en Atom-strøm. For en full oversikt over lovlige parametre og hvordan de brukes henvises det til tillegg A.

Resultat

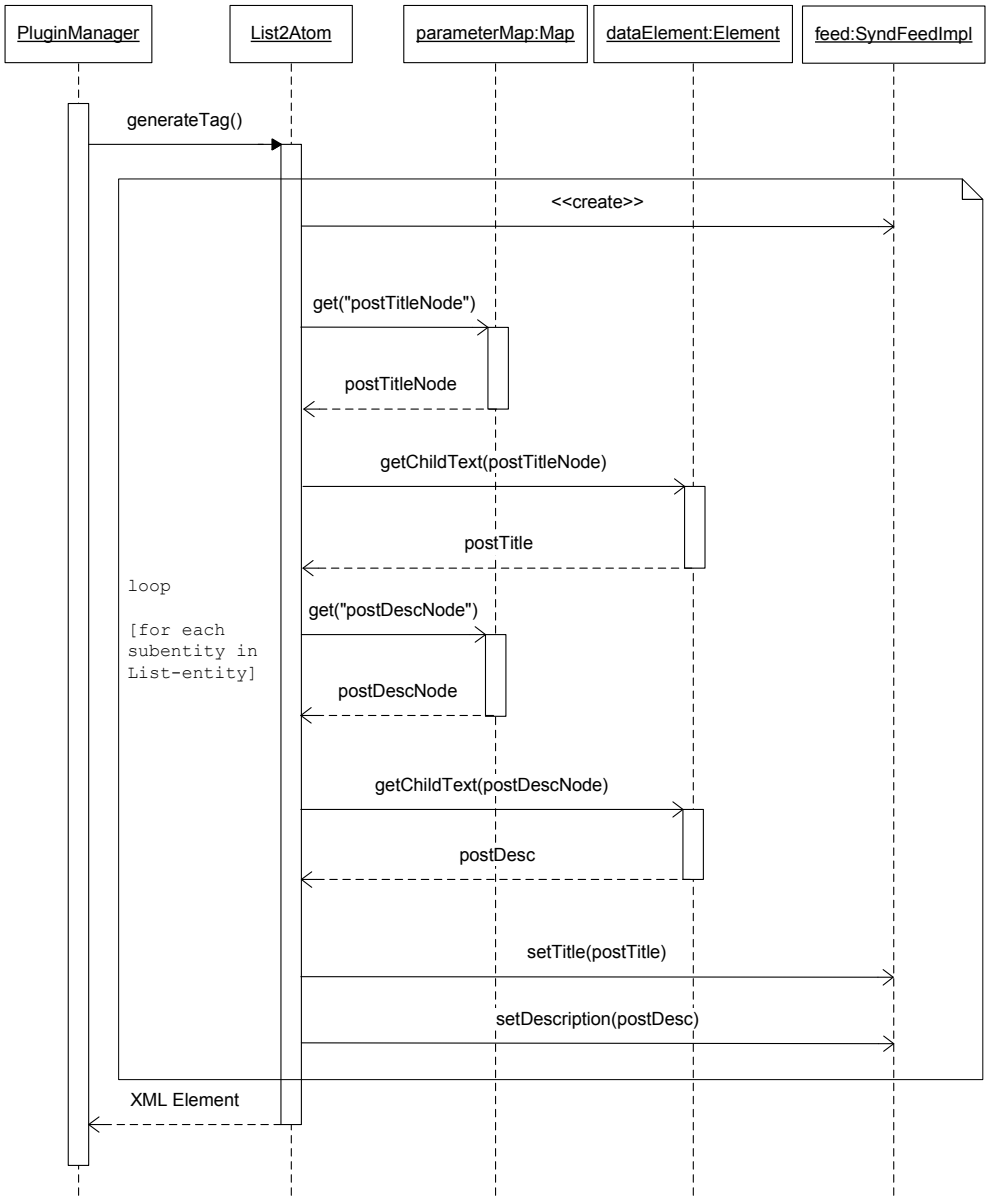
Listing 7.8 definerte at den ferdige strømmen skulle lagres som `meldinger.xml` på serveren. Dette gjøres via plugins sitt grensesnitt mot persistenslaget. På grunn av dette vil filen kunne bli aksessert gjennom adressen:

```
getPresentationResource.html?pid=meldingerTest&type=
                                        PLUGIN_RESOURCE&fileId=meldinger.xml
```

Filen vil da være beskyttet sikkerhetsmessig på lik linje som andre ressurser, noe som innebærer både fordeler og ulemper. For eksempel kan ikke en student abonnere på strømmen gjennom en strømleser, siden studenten kun kan aksessere filen når vedkommende er pålogget. Samtidig beskytter det for at uvedkommende får tilgang til dataene, og AJAX-script som er en del av presentasjonen vil alltid kunne hente ut data. Et eksempel på hvordan meldingseksempelet (`meldinger.xml`) kan se ut med to poster vises i listing 7.9. Atomstømmen er delt opp i to deler: Den første delen inneholder metainformasjon om strømmen, slik som en tittel, og tidspunkt for når den sist ble oppdatert. Den andre delen inneholder en liste med `entry`-elementer, og det er disse elementene som definerer en post. Disse elementene inneholder i sin tur andre elementer som definerer for eksempel tittel og publiseringsdato for den enkelte posten.

Listing 7.9: Eksempel på en ferdig generert Atom-strom av meldinger.

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:dc="http://purl.org/dc/elements
    /1.1/">
  <!-- Del 1 -->
  <title>Meldinger</title>
  <link rel="alternate" href="http://localhost:8080/dpg2.1/
      getPresentationResource.html?pid=inf219&amp;type=PLUGIN_RESOURCE&amp;
      fileId=meldinger.xml" />
  <subtitle>Meldinger i INF399</subtitle>
```



Figur 7.5: Sekvensdiagram for `List2Atom`.

```
    <updated>2010-03-02T13:05:35Z</updated>
    <dc:date>2010-03-02T13:05:35Z</dc:date>

    <!-- Del 2 -->
    <entry>
      <title>Velkommen</title>
      <link rel="alternate" href="http://localhost:8080/dpg2.1/pv/presentation.
          html?pid=inf219&amp;page=startPage&amp;view=meldingerView" />
      <category term="Meldinger" />
      <updated>2010-02-01T23:00:00Z</updated>
      <published>2010-02-01T23:00:00Z</published>
      <summary type="html">&lt;p&gt;Velkommen til INF399! Vi ser frem til et &lt;
          strong&gt;supert &lt;/strong&gt;samarbeid!&lt;/p&gt;</summary>
      <dc:date>2010-02-01T23:00:00Z</dc:date>
    </entry>
    <entry>
      <title>Husk</title>
      <link rel="alternate" href="http://localhost:8080/dpg2.1/pv/presentation.
          html?pid=inf219&amp;page=startPage&amp;view=meldingerView" />
      <category term="Meldinger" />
      <updated>2010-02-13T23:00:00Z</updated>
      <published>2010-02-13T23:00:00Z</published>
      <summary type="html">&lt;p&gt;Husk gjesteforelesningen p&amp;aring mandag!&
          lt;/p&gt;</summary>
      <dc:date>2010-02-13T23:00:00Z</dc:date>
    </entry>
  </feed>
```

7.4.4 FeedMerger-plugin

Den neste plugin som trengs for å oppfylle kravene som er satt, er en plugin som tar inn en eller flere forskjellige strømmer, og setter dem sammen til én. Denne pluginen vil da også ha mulighet til å laste ned en strøm, og lagre den lokalt slik at man omgår sikkerhetsprinsippet til nettleserene om at kun lokale ressurser er tilgjengelige for JavaScript. Det utviklede pluginet blir kalt `feedmerger`. Det vil nå bli gjennomgått hvordan `FeedMerger` kan brukes i et presentasjonsmønster. Som tidligere vil Kursmønsteret bli brukt som case-study. `FeedMerger` skal her *flette* (eng. *merge*) sammen meldingsstrømmen som ble laget i forrige avsnitt med de siste nyhetene som JavaZone har skrevet på det sosiale nettverket Twitter [87].

Definering av entitet

Defineringen av `FeedMerger` i filen `pattern.xml` er svært enkel. Dette pluginet skal ikke ha inndata gjennom PCE, og har derfor felttype `none`. `FeedMerger` (initialisert som `messagesAndJavaZone` her) blir definert som vist i listing 7.10.

Listing 7.10: Definering av `FeedMerger` i pattern.xml.

```
<!-- Vi kaller entiteten som FeedMerger skal legges i for calendarEntity -->
<!-- fordi vi vil bruke den i en kalender                                 -->
<entity id="calendarEntity">
  <!-- Denne instansen av FeedMerger kaller vi messagesAndJavaZone       -->
  <field type="messagesAndJavaZone">feeds</field>
</entity>
```

Spesifisering av plugin

Tilbake i filen `pluginConfig.xml` må man sette opp sammenhengen mellom `messagesAndJavaZone` som ble skrevet i filen `pattern.xml`, og `FeedMerger`. I tillegg til dette må også parametre spesifiseres. Listing 7.11 viser hvordan dette er gjort.

Listing 7.11: `FeedMerger` definert i pluginConfig.xml.

```
<plugin-config id="messagesAndJavaZone" plugin-name="FeedMerger">
  <param name="localFileName">nyhetsfeeds.xml</param>
  <param name="feedURL-1">meldinger.xml</param>
  <param name="feedURL-2">http://twitter.com/15148494.rss</param>
</plugin-config>
```

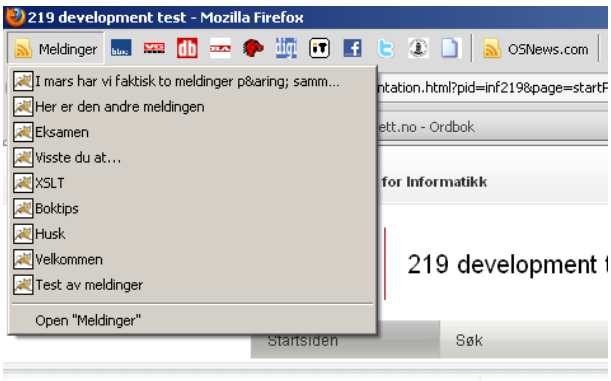
Første parameter (`localFileName`) beskriver her hva den ferdige strømmen skal lagres som i DPGs persistenslag. Dette er likt som i `List2Atom` pluginet. De to neste (`feedURL-1` og `feedURL-2`) er derimot noe mer spesielle:

Det første som skjer er at `FeedMerger` henter `meldinger.xml` lokalt gjennom DPG sitt persistenslag. Hvis `FeedMerger` oppdager at parameteret ikke er en lokal adresse, men en URL, vil pluginet hente innholdet rett fra internettet. Siden det er mye raskere å hente ressurser gjennom det lokale persistenslaget enn å gå gjennom internettet, er førstnevnte den foretrukne metoden. Det er også mulig å definere enda flere strømmer ved å legge til flere parametere med navn `feedURL-n`, der n erstattes med et tall.

`FeedMerger` vil bevare den orginale strømmen sine egenskaper, slik som publiseringsdato og kategori. Det er likevel mulig å overstyre kategorien ved å definere et nytt parameter per strøm, og dermed sette kategorien for hele strømmen. Et eksempel på dette vises i listing 7.12. I dette eksempelet blir strømmen fra Twitter kategorisert som *JavaZone*.

Listing 7.12: Overkjøring av kategori

```
<plugin-config id="messagesAndJavaZone" plugin-name="FeedMerger">
  <param name="localFileName">nyhetsfeeds.xml</param>
  <param name="feedURL-1">meldinger.xml</param>
  <param name="feedURL-2">http://twitter.com/statuses/user_timeline/15148494.
      rss</param>
  <param name="feedCategory-2">JavaZone</param>
</plugin-config>
```



Figur 7.6: Mozilla Firefox viser en strøm generert av `List2Atom` og `FeedMerger`.

Etter at innholdet fra de forskjellige strømmene er lest inn, vil så `FeedMerger` hente ut postene fra alle strømmene, og sette dem sammen til en stor strøm som blir sortert etter publiseringsdato på postene. Etter dette lagres strømmen på serveren.

Resultat

Resultatet som `FeedMerger` produserer er også en standard Atom-strøm. Det blir derfor referert til listing 7.9.

En sideeffekt av valgene i dette kapittelet er at strømmene som blir laget også kan brukes i vanlige strømlesere. Brukere kan derfor abonnere på for eksempel meldinger og oppgaver. Et eksempel på hvordan dette kan se ut i Mozilla Firefox, vises i figur 7.6.

7.4.5 Inkonsekvens i genererte ressurser

Et problem med implimentasjonen av serverløsningen er at det kan bli inkonsekvens i de genererte ressursene (for eksempel en meldingsstrøm) i forhold til innholdet i presentasjonen. Pluginmodellen til DPG har den egenskapen at plugins kun blir kjørt når en presentasjon blir rendrert, det vil si når en bruker besøker en side.

Denne modellen fungerer utmerket så lenge de genererte ressursene brukes på samme side som de ligger i presentasjonsmønsteret. For eksempel vil en generert meldingsressurs alltid være oppdatert hvis den brukes på samme *page* som meldingene er definert i filen `pattern.xml`.

Problemet vil dukke opp når en ressurs ikke blir brukt i denne sammenhengen. For eksempel vil ikke den genererte ressursen være oppdatert hvis en bruker aksesserer den direkte via en nettleser.

For å løse dette problemet bør DPG få hendelseshåndtering for plugins. Pluginene kan da for eksempel bli kjørt hvert femte minutt for å oppdatere de genererte ressursene slik at de blir synkroniserte med presentasjonen.

For å minske problemet er det implementert en løsning der plugins, i tillegg til å bli kjørt når en bruker besøker den aktuelle siden, også blir kjørt i det noen legger inn ny informasjon i PCE-en. Dette fungerer fint så lenge det kun er ressurser fra presentasjonen som blir brukt, men løsningen vil ikke være tilstrekkelig for å holde eksterne ressurser synkronisert. Senere masterstudenter anbefales å se videre på denne løsningen.

8 Klientsideløsning

8.1 Innledning

Det har i det foregående kapittelet blitt forklart hvordan serveren kan behandle presentasjoner slik at man kan få generert ressurser i forutsigbare formater som er enkle å jobbe mot fra klientsiden. Dette kapittelet vil se på hvordan klientsiden kan behandle de samme ressursene ved hjelp av AJAX. Målet med dette kapittelet er å vise noen bruksområder som åpner seg opp når man har denne støtten i DPG.

Også her vil det kun bli fokusert på hvordan informasjon kommer fra server til klient, siden interaksjon den andre veien ikke er implementert.

8.2 Bruksområder

Først skal konsepter fra andre medier undersøkes, og det skal så vurderes om konseptene kan brukes i Kursmønsteret.

8.2.1 Kalender

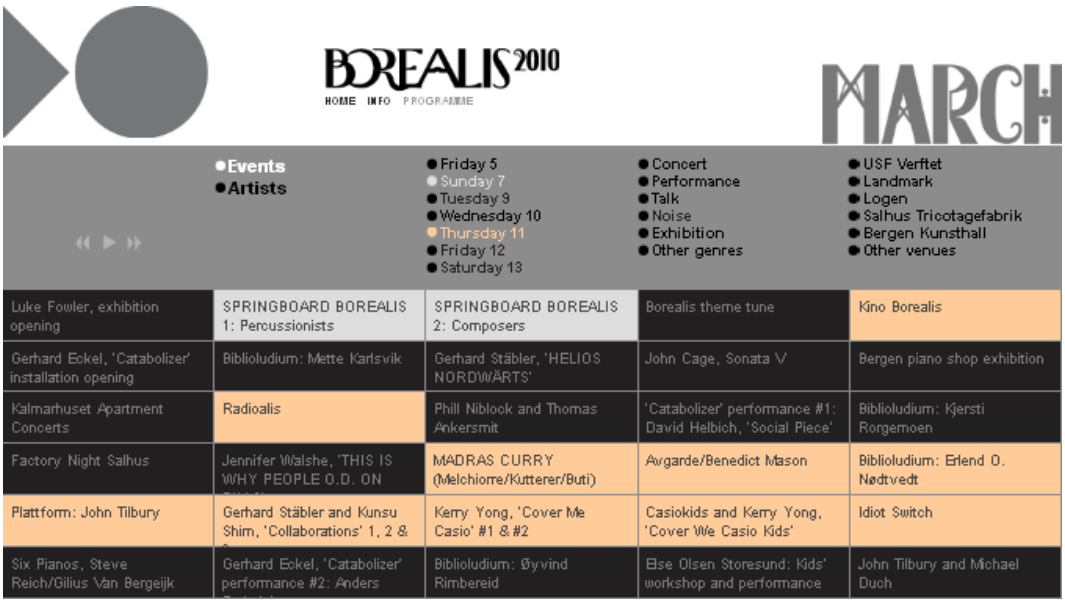
Kalendere er mye brukt på internettsider. De brukes i stor utstrekning for å vise når en hendelse skal foregå, eller har foregått. For eksempel bruker avisen Bergens Tidende en

Figur 8.1: BT bruker en kalender til å vise hva som skjer i Bergen i løpet av de neste månedene [86].

kalender til å vise når det skjer kulturbegivenheter i Bergen, se figur 8.1. Denne kalenderen er tradisjonelt utformet noe som gjør den svært enkel å bruke. Hvis brukeren vil finne ut hva som skjer på en bestemt dato, trykker vedkommende på datoen, hvor da begivenhetene vil bli listet opp under kalenderen.

Et noe mer sofistikert eksempel på en kalender vises på internettsidene til Borealis-festivalen, se figur 8.2. Her har utviklerene valgt å gå bort fra den tradisjonelle visualiseringen av en kalender, og i stedet for valgt å sortere alle begivenhetene alfabetisk. Brukeren kan så endre hvordan kalenderen vises ved å velge et eller flere filtre øverst på siden. Ut fra hvilke filtre som er valgt, vil begivenhetene som passer med de valgte filtrene (for eksempel *Thursday 11* og *Noise*) bli markert. Resultatet blir snittet av alle valgene. Dette er en utradisjonell tilnærming til kalendere, noe som kan føre til at mange kan få problemer med å forstå hvordan den virker.

I vårt case-study, Kursmønsteret, er det stor bruk for en kalender. Brukeren av systemet kan da for eksempel se når obligatoriske oppgaver skal inn, eller når et forelesningsnotat ble lagt ut. En kryssning mellom de to eksemplene synes å være hensiktsmessig for prosjektets formål. Ved å bruke utformingen til en tradisjonell kalender, men med filtreringsmulighetene til Borealis-eksempelet, kan man forhåpentligvis få brukervennligheten fra den tradisjonelle kalenderen, men fleksibiliteten til det andre eksempelet.



Figur 8.2: Borealisfestivalens kalender [22].

8.2.2 Nyhetsticker

TV-kanalen CNN ble i sin tid kjent for sin røde linje på bunnen av fjernsynsskjermen som viste de siste *Headline News* (se figur 8.3). Denne linjen kalte de en *ticker*, eller på norsk *lystavle*. Tickeren ble brukt av CNN for å vise de aller siste nyhetssakene i det de kom inn til redaksjonen, slik at programmet som holdt på i hoveddelen av skjermen kunne gå uforstyrret. Slike *nyhetstickere* har blitt svært populære blant nyhetskanaler, og nå har blant andre britiske *BBC News* og norske *TV2 - Nyhetskanalen* slike i bunnen av TV-bildene sine.

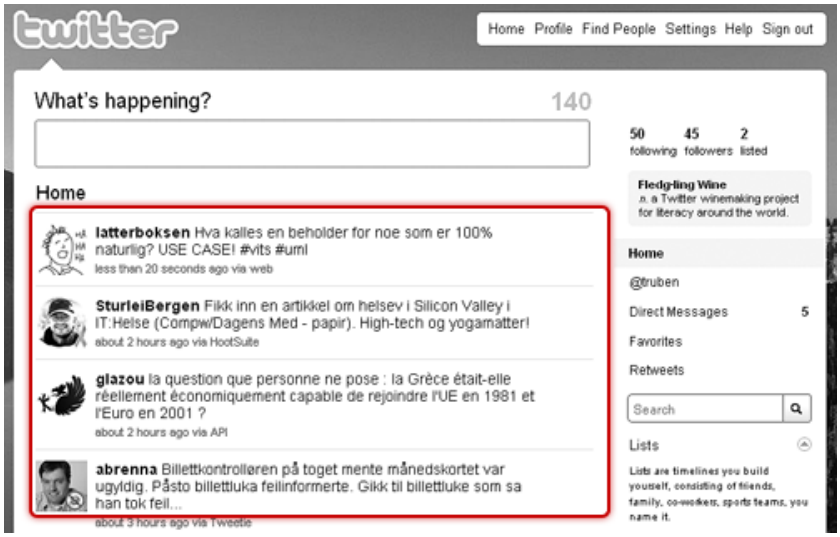
I Kursmønsteret er det nødvendigvis ikke så stort behov for å vise nyhetssaker, men det er derimot lurt å kunne vise de siste innleggene fra systemets hjelpeforum for å stimulere til økt bruk. De fleste forumer tilbyr en eller flere strømmer som inneholder de nyeste innleggene, noe som kan utnyttes i Kursmønsteret.

8.2.3 Kronologisk strømleser

Med en kronologisk strømleser menes det en komponent som viser data fra en strøm i kronologisk rekkefølge, det vil si nyeste først. På det sosiale nettverket *Twitter* [88] blir man for eksempel presentert for en liste med de siste utspillene (kalt *tweets*) fra personer som man har valgt å følge med på (se figur 8.4). Denne listen blir automatisk oppdatert når noen kommer med et nytt utspill.

I Kursmønsteret kan en slik komponent også dras nytte av. Spesielt vil det være nyttig for

Figur 8.3: En skjermdump fra CNN. Nyhetstickeren er innrammet.



Figur 8.4: En skjermdump fra det sosiale nettverket Twitter. Meldingene er innrammet.

å vise meldinger som er lagt til av kursansvarlig.

8.3 Utviklede komponenter

Som vist i avsnitt 6.4.5, er jQuery valgt som rammeverk for å utvikle AJAX-komponenter til Kursmønsteret. Selv om jQuerys bibliotek er omfattende, er bruk av enkelte tredjepartsbiblioteker nødvendig for å oppnå ønsket funksjonalitet. Det er derfor forsøkt å finne komponenter som passer med ønskene for Kursmønsteret, og hvis ikke dette har vært mulig, har egne blitt utviklet.

I første omgang skal komponenter som utvikles fokusere på å støtte nettleserene Mozilla Firefox, Apple Safari, Google Chrome og Opera. Microsoft Internet Explorer, som er den mest utbredte nettleseren, har såpass dårlig støtte for standard JavaScript og CSS at utvikling for denne ville vært både tidskrevende, og ville ødelagt det rene designet som er forsøkt utviklet. Mange av funksjonene som er produsert vil likevel fungere også i denne nettleseren.

8.3.1 Strømtolker

Et bibliotek som som er spesielt viktig for alle komponentene, er en som kan lese inn strømmer for deretter å gjøre dem tilgjengelige for JavaScript som JavaScript-objekter. jQuery-pluginen `jFeed` [48] er valgt for dette formålet. Dette er et lettvektsbibliotek for innlesing av RSS- og Atom-strømmer. Biblioteket `jFeed` er veldig lite i både filstørrelse (4.5 kB) og funksjonalitet, og begrenser seg til å lese inn informasjon fra en strøm, og legge postene inn som en tabell av `jFeedItem`-objekter. Biblioteket er også svært begrenset i forhold til hva slags strømelementer det støtter. For eksempel er det ikke støtte for elementet `category` (kategori). Innlesing av tidspunkt er heller ikke tilfredsstillende: RSS og Atom bruker forskjellige datoformat, RSS bruker RFC822 [25] (for eksempel *Mon, 15 Aug 2010 15:52:01 +0000*), mens Atom bruker RFC3339 [23] (for eksempel *2010-08-15T15:52:01+00:00*). Dette reflekterer ikke `jFeed`, som gir tilbake tidspunkt på samme måte som det står skrevet i strømmen.

På grunn av disse manglene er `jFeed` videreutviklet til å støtte flere strøm-elementer. For eksempel er støtte for elementet `category` lagt til for å kategorisere postene, og `published` for når posten ble opprettet første gang. I tillegg til dette er koden for hvordan `jFeed` lagrer datoer modifisert til å bruke JavaScripts innebygde `Date`-klasse. Dette fører til at datoformatet blir likt uansett hvilket format som blir brukt, i tillegg gjør det det enkelt å sammenligne postenes datoer, siden det da både er mulig å bruke den innebygde `sort()`-funksjonen, samt å bruke vanlige sammenligningsoperatorer som =, < og >.

Figur 8.5: Flere poster på samme dato.

8.3.2 Kalender

Det finnes en rekke kalendere fra før av, både i JavaScript og som jQuery-plugins. Blant annet har *jQueryUI*-biblioteket en egen datovelger. Ut fra et ønske om et funksjonalitetsnivå som ikke finnes i andre komponenter, er det valgt å utvikle en egen kalender.

I kalenderen er følgende funksjonalitet ønsket:

- Direkte integrering med RSS- og Atom-strømmer.
- Mulighet for å filtrere hva som blir vist i kalenderen ut fra kategorier.
- Mulighet for å trykke på datoer i kalenderen for deretter å få opp en boks som viser aktiviteten på denne datoen.
- Enkel og intuitiv navigering mellom måneder.

Den ferdige kalenderen vises i figur 8.6. Den oppfyller alle kravene som er satt, og integreres meget enkelt med serverløsningen. For å legge en kalender til en presentasjon, skrives et enkelt JavaScript (vist i listing 8.2) som starter opp kalenderpluginen. Det er også mulig å gi kalenderen ekstra parametere som for eksempel forteller den hva slags måned eller år den skal begynne i. For komplett dokumentasjon av kalenderen, se B.1.

Kalenderen baserer seg på standard konvensjoner i forhold til hvordan kalendere ser ut: Helt øverst finnes navnet på måneden og årstallet, og på hver side av denne, piler som fører frem og tilbake i året. Under dette finnes en matrise med datoer, der første rad er mandager, andre rad tirsdager og så videre. Denne delen av kalenderen er markert med et 1-tall på figur 8.6

Når kalenderen blir opprettet vil pluginen gå gjennom strømmen som den er koblet til (ved hjelp av `atomFeed` parameteret), og markere datoer i kalenderen hvis den finner at en post i strømmen har en `updated` verdi som samsvarer med datoen. Denne markeringen vil være forskjellig ut fra hva slags kategori posten tilhører. Markeringen er fullstendig konfigurerbar ved hjelp av *stilark* (eng. *stylesheet/CSS*). Hvis det er flere poster som er koblet til samme dag, vil en stjerne bli synlig før datoen. Hvis brukeren lar musepekeren hvile over datoen vil også en liten tekst komme opp som forteller hvor mange poster som skjuler seg bak datoen. Dette vises i figur 8.5.

Hvis en bruker trykker på en dato som har en post koblet mot seg, vil en modal dialogboks vise mer informasjon fra posten (for et eksempel, se figur 8.9). Som en visuell indikasjon på at denne dialogboksen viser informasjon fra nettopp en dato, vil det vises en kort animasjon der datoen i kalenderen vil transformeres til en dialogboks. Det er utviklet en egen dialogboks for å vise disse beskjedene, men det er mulig å overstyre dette ved hjelp av parameteret `dialogFunction`. Man kan for eksempel bruke jQueryUIs dialogboks, eller å bruke nettleserens egen funksjon for å vise beskjeden. Et eksempel på et parameter som får kalenderen til å vise innholdet til en dato ved hjelp av nettleserens innebygde metode, vises i listing 8.1.

Listing 8.1: Parameter for å overstyre visning av eventer.

```
// alert() er JavaScripts innebygde dialogboksfunksjon
dialogFunction: function($element){alert($element.text());}
```

Helt nederst i kalenderkomponenten vises en liste med alle kategoriene som finnes i strømmen (markert med et 2-tall i figur 8.6). Bakgrunnen til hver av kategoriene er fargelagt i samme farge som bakgrunnen til datoene som er koblet mot poster i samme kategori. Disse kategoriene fungerer som *av-på-knapper* for å fargelegge kalenderen. Det vil si at hvis en kategori er skrudd av, vil ikke poster som er koblet mot denne kategorien vises i kalenderen.

Listing 8.2: Initialisering av kalenderkomponenten.

```
// Skriptet startes nar DOM-treet er ferdig konstruert
$(document).ready(function() {
  // Denne kommandoen starter kalenderen
  $('.coolCalendarPlace').coolCalCreate({
    // Her definerer vi parameterene til kalenderen
    atomFeed: 'nyhetsfeeds.xml'
  });
});
```

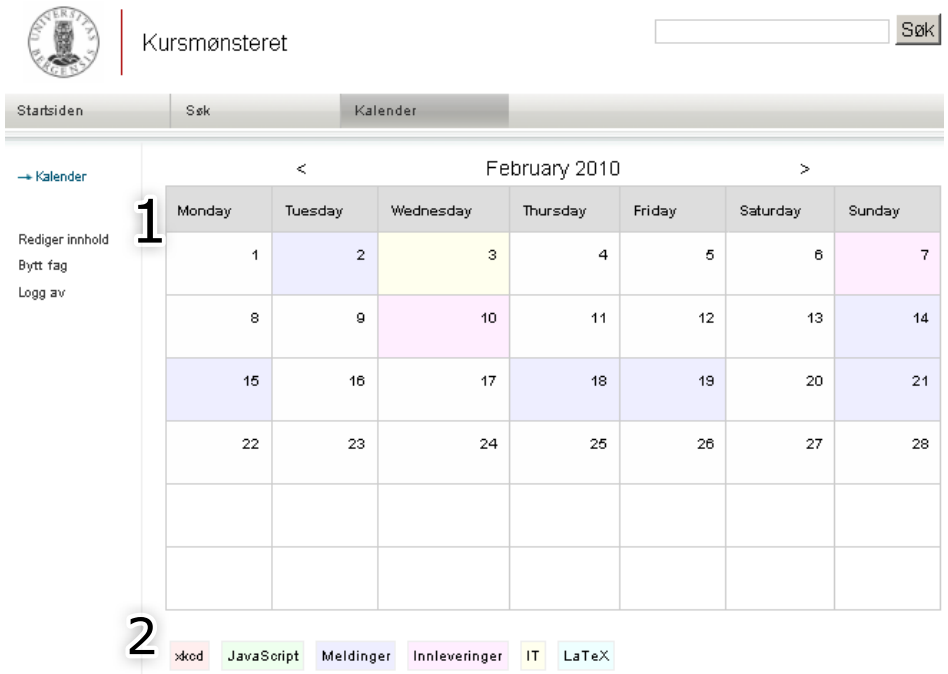
8.3.3 Nyhetsticker

Nyhetstickeren er en forholdsvis enkel plugin, som det finnes flere implementasjoner av allerede. For eksempel finnes *BBC style news ticker* [46] og *liScroll* [73]. Problemet med dem er at de ikke er produsert med tanke på AJAX, samt at de ikke kan kobles mot strømmer. På grunn av dette er det utviklet en egen nyhetsticker som kan kobles direkte mot strømmer.

Følgende funksjonalitet var ønskelig:

- Direkte integrering med RSS- og Atom-strømmer.
- Enkelt, men konfigurerbart design.

Den ferdige nyhetstickeren sees i figur 8.7, og koden for å initialisere den finnes i listing 8.3. En relativt lik grafisk fremvisningsmåte som CNN (se figur 8.3) er valgt, med en tickeroverskrift

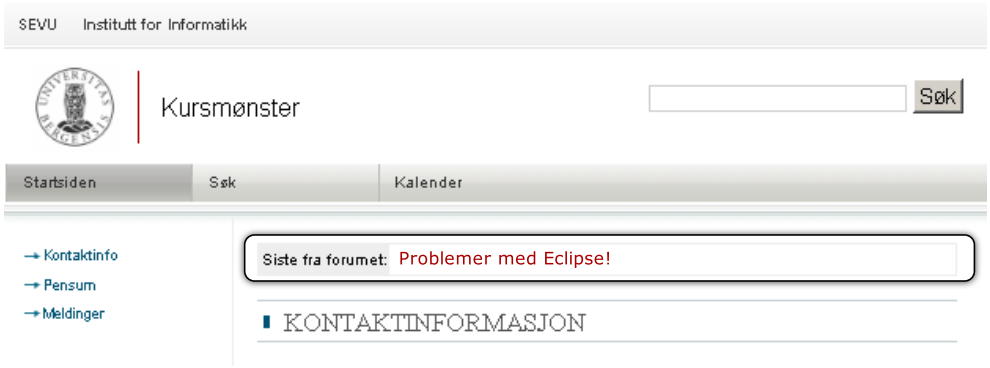


Figur 8.6: Kalenderkomponenten.

som forteller hva nyhetstickeren handler om (på figur 8.7 er den *Siste fra forumet*), og et område for overskriften til selve nyhetssaken. Det er da mulig å trykke på denne overskriften for å bli sendt videre til internettsiden der nyhetssaken vises (i dette tilfellet forumet). Med `timer` konfigurasjonen kan utvikleren velge hvor lenge hver nyhetssak skal stå før den skifter til neste i rekken. Som et visuelt hint om at den skifter nyhetssak vil teksten rulle nedover, og den nye komme inn fra toppen. For komplett dokumentasjon av nyhetstickeren se B.2.

Listing 8.3: Initialisering av nyhetstickerkomponenten.

```
// Skriptet startes nar DOM-treet er ferdig konstruert
$(document).ready(function() {
  // Kommando for a starte tickeren
  $('#ticker').coolTickerCreate({
      // Parametere til komponenten
      atomFeed: 'forum.xml',
      tickerTitle: 'Siste fra forumet',
      timer: '5000'
     });
});
```



Figur 8.7: Nyhetstickerkomponenten i Kursmønsteret. Komponenten er rammet inn.

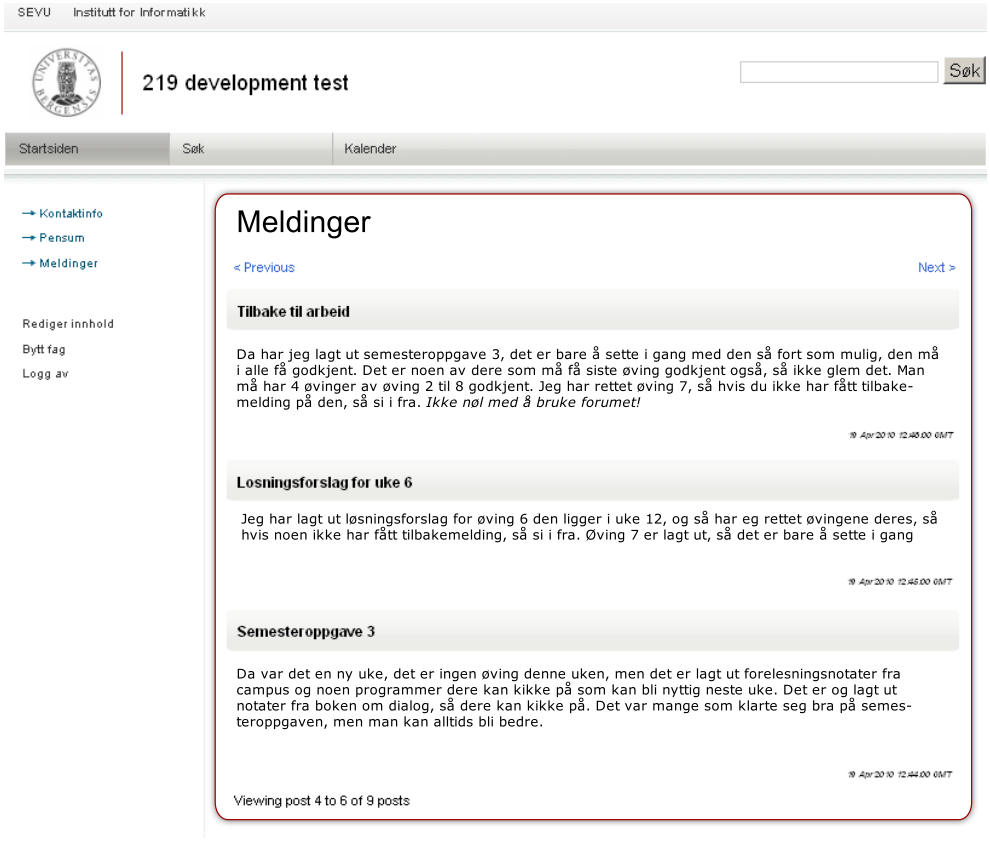
8.3.4 Kronologisk strømleser

Den kronologiske strømleseren er en komponent for å vise frem kortere beskjeder i kronologisk rekkefølge. I kursmønsteret er det, som tidligere nevnt, meningen at kursansvarlig skal kunne gi kortere beskjeder til studentene. Den ferdig utviklede komponenten vises i figur 8.8, og koden for å starte den opp finnes i listing 8.4.

Listing 8.4: Initialisering av den kronologiske stromleser

```
// Skriptet startes nar DOM-treet er ferdig konstruert
$(document).ready(function() {
  // Kommando for a starte leseren
  $('#meldinger').coolMessageCreate({
    // Parametere til komponenten
    atomFeed: 'meldinger.xml',
    numberOfPosts: 3
   });
});
```

Denne komponenten er den eneste av de tre komponentene som kan bli produsert med relativt lik funksjonalitet bare ved hjelp av XSLT. Det som derimot ikke er mulig i dagens DPG, er å dele opp innhold og presentere det på flere separate sider (eng. *pagination*). Dette er mulig i den kronologiske strømleseren som er utviklet i denne masteroppgaven. I eksempelet er det valgt å kun vise tre meldinger per side. Det er da mulighet for å trykke på lenkene *next* og *previous* for å bla seg frem og tilbake i meldingene. Som visuell indikasjon på hva som skjer når en bruker trykker på en av knappene, vil meldingene *skli* til den naturlige siden for å gi plass til nye. For full dokumentasjon av den kronologiske strømleseren, se tillegg B.4.



Figur 8.8: Kursmønster med kronologisk-strømleser (rammet inn).

8.3.5 Dialogboks

Det var i utgangspunktet ikke ønskelig å lage en ny dialogboks widget fordi det finnes så mange varianter av denne komponenten. For eksempel har jQueryUI en egen versjon (se 6.3). Denne syntes likevel litt for avansert for bruk i Kursmønsteret, og en enklere komponent var derfor ønskelig. To forskjellige tredjepartskomponenterer ble derfor testet ut i stedet.

Komponentene som ble testet ut var *SimpleModal* [67] og *Facebox* [94]. Begge to er svært enkle komponenter som viser hvilken som helst type HTML. Problemet med dem er at ingen av dem er laget for AJAX bruk. Dette vil si at de ikke tilpasser seg dynamisk innhold. Derfor ville for eksempel ikke dialogboksen få riktig størrelse hvis innholdet for eksempel var et bilde.

På grunn av dette problemet ble det valgt å utvikle en ny dialogboks. Denne dialogboksen venter til alt innholdet er lastet inn før den blir vist. Dette gjør at den får riktig størrelse på boksen. Den har videre fått et mer moderne utseende enn de alternative komponentene. Dialogboksen er modal, det vil si at alle andre komponenter på siden er deaktivert så lenge dialogboksen vises. Visuelt vises dette ved å legge et halveis gjennomsiktig lag over siden slik at en kan se at resten av siden er deaktivert. Selv om funksjonaliteten på mange måter overgår både Facebox og SimpleModal, er denne implementasjonen av dialogboksen på under halvparten så mange kodelinjer som begge alternativene.

Et eksempel på hvordan dialogboksen kan se ut finnes i figur 8.9, og koden for å vise dialogboksen er vist i listing 8.5.

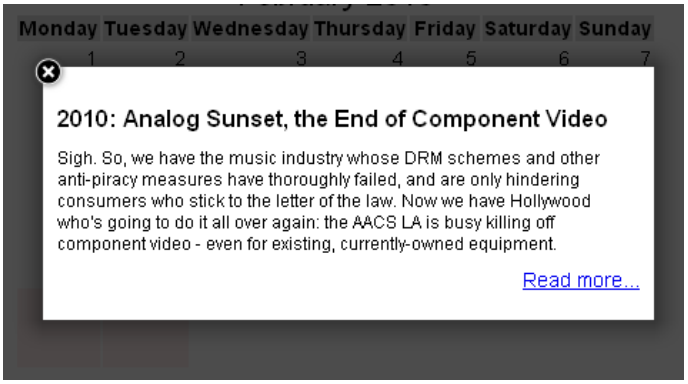
Listing 8.5: Vis dialogboksen.

```
// Hent DOM elementet som brukes til a vise dialogboksen
// (for a lage en overgangseffekt)
$trigger = $('#button');

// Vis dialogboksen
$($trigger).coolModalCreate($('<div>Innhold</div>'));
```

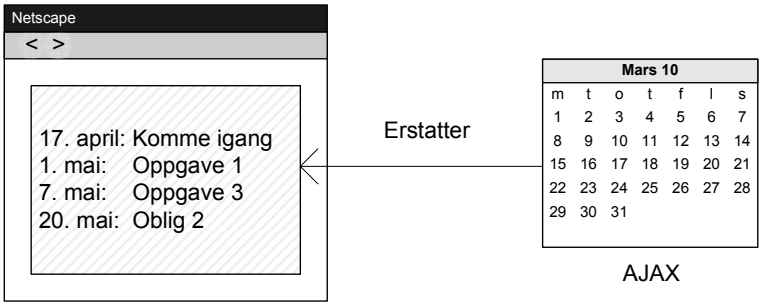
8.4 Kompatibilitet med nettlesere uten JavaScript-støtte

Selv om alle de store nettleservariantene (blant andre Mozilla Firefox, Microsoft Internet Explorer, Google Chrome, Apple Safari og Opera) har støtte for JavaScript, er det flere brukere som ikke benytter seg av dette. Dette kan være av forskjellige grunner som for eksempel at brukeren er bekymret for sikkerheten, eller at brukeren er blind, og dermed ikke bruker en vanlig skjerm. En annen *bruker* som ikke kjører JavaScript er webcrawlere [77]. Dette er programmer som går gjennom nettsider og indekserer dem, slik at det i ettertid er mulig å søke gjennom dem. Det blir derfor vanskelig å lage søk for en side som kun viser innhold ved hjelp av JavaScript.

I en slik situasjon er det viktig at brukeren fremdeles kan få tak i innholdet, om det så er i



Figur 8.9: Eksempel på dialogboksen.



Figur 8.10: Hvordan oppnå bakoverkompatibilitet med brukere som ikke støtter Javascript.

redusert form. De plugins som her er utviklet er laget på den måten at det er relativt enkelt å produsere presentasjoner som fungerer med nettlesere som ikke kjører JavaScript. Hvis man tar den kronologiske strømleseren som eksempel, så kan man bruke den tradisjonelle metoden for å utvikle en liste med meldiger ved hjelp av XSLT, og etter dette la JavaScript erstatte dette innholdet med den litt mer avanserte kronologiske nyhetsleseren. I figur se figur 8.10) erstatter JavaScriptet en tekstrepresentasjon av en kalender, med en visuell versjon. På denne måten vil da den samme informasjonen være tilgjengelig for alle, bare at brukere med nettlesere som har JavaScript tilgjengelig, vil få et mer dynamisk og stilig utseende.

8.5 Evaluering

I dette kapittelet har det blitt sett på hvordan man, med støtte fra serveren, kan utvikle programvare for rike klienter i DPG. Programvaren som er utviklet er standard jQuery plugins, og fungerer derfor i hvilken som helst sammenheng. Det er mulig å bruke dem, uten modifisering, på hvilken som helst nettside, så lenge siden har serverstøtte.

Alle utviklede jQuery-plugins er også konstruert slik at alle visningsegenskaper, som farger, skrifttyper og plassering, er konfigurerbare ved hjelp av stilark. Dette gjør det enklere å tilpasse dem til nye presentasjoner eller andre nettsider.

9

Evaluering, konklusjon og videre arbeid

9.1 Måloppnåelse

Det overodnede målet til denne masteroppgaven var å utvikle støtte for rike klienter i DPG. For å oppnå dette, måtte en rekke delmål oppfylles. Disse var:

1. Refaktorering av arkitekturen til DPG 2.0 for utviklingen av presentasjonsmønstre som støtter rike klienter.
2. Refaktorering av presentasjonsmønsterfikasjonen for å nå hovedmål.
3. Undersøke bruksområder for rike klienter i Kursmønsteret.
4. Utvikle klientprogramvare som kan utnytte serverstøtten utviklet i delmål 1.

Delmål 1 handlet om å gjøre arkitekturen til DPG i stand til å støtte rike klienter. Dette har i denne masteroppgaven blitt gjort, som beskrevet i kapittel 7, ved hjelp av pluginarkitekturen. For at pluginarkitekturen skulle være i stand til å utføre dette, var det nødvendig å refaktorere den, og legge til ny funksjonalitet. Dette har blitt gjort i samarbeid med Tobias Rusås Olsen [77] i kapitlene 3 og 4. Ved hjelp av den utvidede pluginarkitekturen har to plugins blitt utviklet, `List2Atom` og `FeedMerger`. Disse har til sammen oppnådd samtlige krav (satt i avsnitt 7.4.2) for serverstøtten på en tilfredstillende måte. Delmål 1 anses som oppnådd.

For å kunne oppnå hovedmålet var det ønskelig med en gjennomgang av DPG 2.0s presentasjonsmønsterspesifikasjon. Dette var nødvendig fordi presentasjonsmønsterspesifikasjonen

ikke var godt integrert med plugins. Kapittel 5 beskriver endringene som denne masteroppgaven har gjort med presentasjonsmønsterspesifikasjonen. Resultatet er at det har blitt enklere å bruke plugins, og at det er mulig å bruke plugins på måter som tidligere ikke var mulig (entitetslister). Delmål 2 anses som nådd. Målet er nådd i samarbeid med Tobias Rusås Olsen [77].

For å undersøke bruksområder for rike klienter i Kursmønsteret, som var delmål 3, har masteroppgaven forsøkt å finne områder hvor rike klienter har blitt benyttet i andre sammenhenger. Dette har blitt gjort i kapittel 8. Det ble vurdert at en kalender, en nyhetsticker og en kronologisk strømleser kan være nyttige komponenter i Kursmønsteret. Dette delmålet må anses som oppnådd.

For å oppnå delmål 4 var det nødvendig å utvikle klientprogramvare som skulle utnytte den nyervervede serverstøtten. Kapittel 6 forklarer hva slags JavaScript-biblioteker som kan brukes for å utvikle klientprogramvare, mens kapittel 8 viser hva slags programvare som er utviklet. Sistnevnet ble utviklet ut fra analysen som ble utført i sammenheng med delmål 3. På bakgrunn av disse kapitlene anses delmål 4 som løst.

DPG har, ved at alle delmålene har blitt oppnådd, blitt et internettinnholdssystem som støtter rike klienter. Hovedmålet med denne masteroppgaven må derfor anses som som oppnådd.

9.2 Konklusjon

Denne masteroppgaven beskriver vurderinger og arbeid rundt utviklingen av DPG 2.1. Det har for forfatteren vært en svært nyttig erfaring å videreutvikle et større system som DPG. Spesielt har det vært lærerikt å arbeide i team. I dette teamet har det blitt diskutert blant annet valg av teknologi og arkitektur, noe som har ført til god kjennskap til mange problemstillinger innen DPG og programvareutvikling generelt. Ved å jobbe i et team har det vært nødvendig å tilegne seg gode rutiner. Effektiv bruk av versjonskontrollprogramvare og optimalisering av kompilerings- og kjøremiljø har derfor vært uunværlig.

Forfatteren har gjennom utviklingsprosessen også fått kjennskap til flere ulike teknologier som blant annet AJAX, jQuery og Spring. Dette er teknologier som brukes i stor utstrekning i arbeidslivet, og kunnskap om disse er derfor verdifult å ta med videre.

Som nevnt i avsnitt 1.5 har forfatteren forsøkt å bruke praksisene fra den smidige systemutviklingsmetoden XP. Det er blant annet benyttet parprogrammering, refaktorering og noe testdrevet utvikling. En masteroppgave er et inviduelt arbeid, og flere av praksisene i XP har derfor ikke vært hensiktmessige å bruke.

DPG 2.1 har blitt til et utvidbart og solid system. Først og fremst har pluginarkitekturen gjort at det er enklere å legge til ny funksjonalitet. Koden til DPG har også blitt enklere å vedlikeholde på grunn av refaktorering av koden til store deler av systemet. I tillegg har det blitt enklere å utvikle presentasjonsmønstre, noe som legger et grunnlag for at også personer

uten tyngre kunnskap innen informatikk kan utvikle presentasjonsmønstre.

Arbeidet med DPG og presentasjonsmønstre har vært svært interessant. Det har vært spennende å jobbe med et nytt og innovativt konsept, og se hvordan det kan utvikles i nye retninger. Siden konseptet fremdeles er relativt nyetablert, har det vært lettere å tenke nytt, og forfatteren har derfor fått muligheten til å utfolde seg kreativt. Dette hadde vært vanskeligere innenfor et mer etablert konsept.

9.3 Videre arbeid

DPG har flere spennende utfordringer foran seg. I de neste avsnittene vil det bli presentert forslag for videre arbeid.

9.3.1 Hendelseshåndtering for DPG-plugins

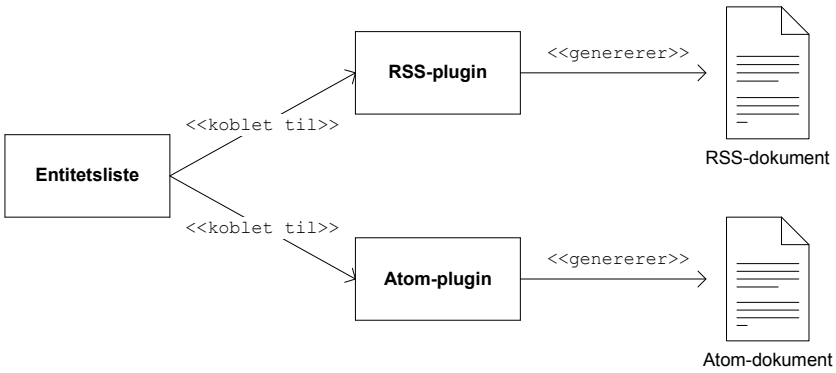
Som beskrevet i avsnitt 7.4.5, er inkonsekvens i de genererte ressursene til `List2Atom` og `FeedMerger` et problem. Dette er fordi plugins kun blir kjørt når en side blir rendrert eller når innholdet til et felt blir oppdatert i PCE-delsystemet. Det er ønskelig å få utviklet en generell mekanisme som kan kjøre DPG-plugins basert på hendelser i systemet. Eksempel på hendelser er når et visst klokkeslett intreffer og når en ny entitet blir lagt til i en entitetsliste. Dette systemet kan for eksempel basere seg på Observer-designmønsteret [68]. Spring rammeverket, som DPG bygger på, inneholder også støtte for hendelseshåndtering. For mer informasjon om dette, henvises det til avsnitt 3.4.5.

9.3.2 Standardisert plugindokumentasjon

Alle pluginforfattere har til nå utviklet sin egen måte å dokumentere plugins på. Det er ønskelig å få produsert en mal for hvordan plugins blir dokumentert, slik at det blir enklere å lese og å produsere dokumentasjonen. Denne dokumentasjonen kan for eksempel ta utgangspunkt i JavaDoc [16] som brukes til å dokumentere Java-kode. Eksempler på hvordan plugins blir dokumentert finnes i tillegg A og i masteroppgaven til Bjørn Ove Ingvaldsen, *Multimedia i Dynamic Presentation Generator 2.0* [59].

9.3.3 Skille mellom obligatoriske og valgfrie pluginparametre

Det finnes i DPG 2.1 kun én måte å fortelle omverdenen hva slags parametre en plugin kan behandle, og det er via `getParameters()`-metoden. Det er ikke alltid alle nødvendig å sette alle parametre for at et plugin skal kunne kjøre. Det er derfor ønskelig å kunne skille mellom obligatoriske og valgfrie parametre. DPG har da muligheten til å undersøke om de



Figur 9.1: To plugins koblet til samme liste.

obligatoriske parametrene er definert før den kjører plugins, slik at ikke plugins trenger å gjøre dette selv.

9.3.4 Mulighet for flere plugins på samme felt

Felter kan i dag kun ha én plugin koblet mot seg. I enkelte tilfeller er det ønskelig å kunne koble til flere plugins til samme felt. For eksempel kunne en liste ha hatt to plugins, der det første pluginet genererer en RSS-strøm, mens det andre genererer en Atom-strøm. Figur 9.1 viser hvordan en entitetliste er koblet mot to plugins som genererer hver sin type dokument.

9.3.5 Autorisert tilgang til eksterne ressurser

DPG-pluginet `FeedMerger` har ingen mulighet til å aksessere ressurser der det kreves autorisering for å få tilgang. Derfor er det for eksempel ikke mulig for `FeedMerger` å hente en strøm med de nyeste postene fra et forum som krever innlogging. Det hadde vært nyttig med en mekanisme som kan gi `FeedMerger` muligheten til å autorisere seg, slik at det er mulig å få tilgang til slike eksterne ressurser.

9.3.6 Tillate uautorisert tilgang til ressurser

Strømmene som blir generert av `FeedMerger` og `List2Atom` ligger beskyttet bak DPGs autoriseringssystem. Det er tilfeller der man ikke ønsker dette. For eksempel kan det være nyttig for studenter å abonnere på en strøm med meldinger med et program som Mozilla Firefox eller Mozilla Thunderbird. Dette er ikke mulig i dagens DPG. Det er derfor ønskelig med en mekanisme slik at *enkelte* ressurser ikke krever autorisert adgang.

9.3.7 Plugin for UML-tegning

Det nye elementet `<canvas>` i HTML 5 tilbyr et grensesnitt for å presentere 2D-grafikk på internettsider [90]. Det hadde vært spennende å bruke dette elementet til for eksempel UML-fremvisning. Foreleser kan da tegne UML-en i PCE-delsystemet, mens studentene ser på i PV-systemet. Dette kan foregå i sanntid ved å bruke AJAX-konseptet.

Elementet `<canvas>` har også mange andre interresante bruksområder i fjernundervisningssammenheng. Man kan for eksempel se for seg at det kan brukes til å produsere et alternativ til *Microsoft PowerPoint* [12] for bruk direkte i nettleseren.

A

Utviklede DPG-plugins

Dette tillegget inneholder dokumentasjon om DPG-plugins som har blitt utviklet i denne masteroppgaven. For mer informasjon om disse komponentene, henvises det til kapittel 7.

A.1 List2Atom

A.1.1 Beskrivelse

`List2Atom` er en plugin som produserer Atom-strømmer ut fra en liste med entiteter i DPG. For mer informasjon om `List2Atom`, se avsnitt 7.4.3.

A.1.2 Parametre

Alle parametre kan defineres på to måter. Enten kan parametre defineres ved å bruke en skråstrek (/) før verdien, eller så kan skråstreken utelates. Med skråstreken vil `List2Atom` sette verdien til parametret til å bli innholdet til elementet i listen med samme navn, mens det uten skråstreken kun vil bruke teksten. Hvis for eksempel parameteret `feedEntryTitle` blir definert som under vil `List2Atom` hente ut innholdet til elementet `overskrift`:

```
<param name="feedEntryTitle">/overskrift</param>
```

I dette eksempelet vil `List2Atom` sette tittelen til alle postene i strømmen til *Overskriften til posten*:

```
<param name="feedEntryTitle">Overskriften til posten</param>
```

Obligatoriske parametre

Parametrene under dette avsnittet er nødvendig å definere for at `List2Atom` skal kunne kjøre.

`feedEntryDescription`
Innholdet til en post.

`feedEntryTitle`
Tittelen til en post.

`feedFileName`
Hva filnavnet til den genrerte Atom-strømmen skal være i persistenslaget til DPG.

`feedTitle`
Tittelen til strømmen.

Valgfrie parametre

De følgende parametrene kan utelates.

`feedCategory`
Kategorien til strømmen.

Standardverdi: `(ingenting)`

`feedDescription`
Beskrivelse av strømmen.

Standardverdi: `(ingenting)`

`feedEntryDate`
Tidspunktet for når posten sist ble oppdatert.

Standardverdi: `(Datoen til når entiteten ble oppdatert i DPG)`

`feedEntryLink`
URL til hvor posten finnes.

Standardverdi: `(URL til hvor posten finnes i DPG)`

feedLenght
Begrenser lengden til strømmen. For eksempel kan man si at strømmen kun skal inneholde de ti nyeste postene ved å sette dette parameteret til `10`.

Standardverdi: ∞

A.2 FeedMerger

A.2.1 Beskrivelse

`FeedMerger` setter sammen flere forskjellige strømmer til én. Man kan også bruke `FeedMerger` til å laste ned strømmer fra eksterne kilder. For mer informasjon om `FeedMerger`, se avsnitt 7.4.4.

A.2.2 Parametre

`FeedMerger` sine parametre har samme egenskaper som `List2Atom` sine. Det henvises derfor til A.1.2 for dokumentasjon for hvordan man kan definere parametre.

Obligatoriske parametre

feedFileName
Hva filnavnet til den genrerte Atom-strømmen skal være i persistenslaget til DPG.

feedTitle
Tittelen til strømmen.

feedURL-*n*
Definerer URL-en til en strøm som `FeedMerger` skal lese inn. Strømmen kan være både RSS og Atom. *n* betyr et tall. `FeedMerger` vil begynne med parameteret `feedURL-1` og fortsette oppover så lenge den finner nye parametre. Postene fra alle strømmene vil til slutt bli flettet sammen og sortert etter når de sist ble oppdatert. URL-ene kan peke til eksterne eller interne ressurser. Hvis URL-en starter med `http://` eller `https://` vil `FeedMerger` lete etter ressursen eksternt. Hvis ikke vil den hente ressursen gjennom persistenslaget.

Valgfrie parametre

feedCategory-*n*
Setter kategorien til alle postene som inngår i strømmen `feedURL-n`. Nummeret *n* må korrespondere med nummeret *n* i parameteret `feedURL-n`

Standardverdi: `(Bruker kategorien som eventuelt finnes i posten)`

B

Utviklede jQuery-plugins

Dette tillegget inneholder dokumentasjon for alle jQuery plugins som ble utviklet i kapittel 8. `jFeed` er ikke tatt med på grunn av at bruken til den modifiserte versjonen som er utviklet i denne oppgaven er identisk med den orginale. For dokumentasjon til `jFeed` henvises det til [48].

B.1 CoolCalendar

B.1.1 Beskrivelse

Viser postene til en strøm merket av i en kalender. Komponenten tar utgangspunkt i når postene i strømmen sist ble oppdatert når den merker av datoer. For mer informasjon om kalenderkomponenten, se avsnitt 8.3.2. Figur B.1 viser en skjermdump av kalenderen.

B.1.2 Instillinger

`CoolCalendar` har følgende innstillingsmuligheter:

`atomFeed`
URL til strømmen som skal vises.

Standardverdi: `(ingenting)`

classPrefix
Prefiks til alle CSS-klassene som `CoolCalendar` skal bruke. Brukes hvis en ønsker å ha flere versjoner av kalenderen på samme side.

Standardverdi: `'cool'`

dialogFunction
En funksjon som forteller komponenten hvordan den skal vise innholdet til en dato. Hvis ingen funksjon blir definert vil den bruke `coolModal`.

Standardverdi: `(ingenting)`

month
Måneden som kalenderen skal starte i. 0 = januar, 11 = desember.

Standardverdi: `(gjeldende måned)`

updateEvery
Hvor ofte kalenderen skal sjekke for oppdaterte innhold i strømmen. Oppgis i millisekunder. `0` vil si at komponenten ikke sjekker for nye oppdateringer.

Standardverdi: `5000`

year
Året som kalenderen skal starte i.

Standardverdi: `(gjeldende år)`

B.1.3 Eksempel

```
$('.ticker').coolCalCreate({
    month : 10,
    year : 2010,
    atomFeed : 'messages.xml',
    dialogFunction : function($story){alert($story.text())},
    classPrefix: 'cool',
    updateEvery: 10000
});
```

B.2 CoolNewsTicker

B.2.1 Beskrivelse

Viser overskriftene til postene i en strøm. Overskriftene blir vist én og én, og skifter til neste overskrift etter et bestemt tidsinterval. For mer informasjon om denne komponenten, se avsnitt 8.3.3. Figur B.3 viser en skjermdump av komponenten.

B.2.2 Instillinger

`CoolNewsTicker` har følgende innstillingsmuligheter:

`atomFeed`
URL til strømmen som skal vises.

Standardverdi: `(ingenting)`

`classPrefix`
Prefiks til alle CSS-klassene som `CoolNewsTicker` skal bruke. Brukes hvis en ønsker å ha flere versjoner av nyhetstickeren på samme side.

Standardverdi: `'cool'`

`tickerTitle`
Tittelen til tickeren. Vises til venstre i komponenten.

Standardverdi: `'Ticker'`

`timer`
Hvor lenge hver overskrift skal vises før den skifter til den neste i rekken. Angis i millisekunder.

Standardverdi: `5000`

`updateEvery`
Hvor ofte tickeren skal sjekke for oppdaterte innhold i strømmen. Oppgis i millisekunder. `0` vil si at komponenten ikke sjekker for nye oppdateringer.

Standardverdi: `5000`

B.2.3 Eksempel

```
$('.ticker').coolTickerCreate({
    atomFeed : 'messages.xml',
```

```
    tickerTitle: 'Beskjeder',
    timer: 10000,
    classPrefix: 'cool',
    updateEvery: 10000
});
```

B.3 CoolDialog

B.3.1 Beskrivelse

Viser en dialogboks i midten av skjermen. Det er ikke mulig å bruke resten av nettsiden mens dialogboksen vises. Brukeren kan gå ut av dialogboksen ved å trykke *ESC*-knappen på tastaturet, trykke utenfor dialogboksen, eller ved å trykke på en lukkeknapp i dialogboksens øvre venstre hjørne. For mer informasjon om dialogboksen, se avsnitt 8.3.5. Figur B.2 viser en skjermdump av komponenten.

B.3.2 Instillinger

CoolDialog har ingen innstillingsmuligheter.

B.3.3 Eksempel

```
$().coolModalCreate($('<div>Innhold</div>'));
```

Med overgangseffekt:

```
$($fraElement).coolModalCreate($('<div>Innhold</div>'));
```

B.4 CoolMessages

B.4.1 Beskrivelse

Plugin som brukes til å vise postene til en strøm. Postene vil bli sortert kronologisk etter når de sist ble oppdatert. For mer informasjon om den kronologiske strømleseren, se avsnitt 8.3.4. Figur B.4 viser en skjermdump av kalenderen.

B.4.2 Instillinger

`CoolMessages` har følgende innstillingsmuligheter:

`atomFeed`
URL til strømmen som skal vises.

Standardverdi: `(ingenting)`

`classPrefix`
Prefiks til alle CSS-klassene som pluginet skal bruke. Brukes hvis en ønsker å ha flere versjoner av pluginet på samme side.

Standardverdi: `'cool'`

`numberOfPosts`
Hvor mange poster som blir vist på hver *side*.

Standardverdi: `5`

`startAt`
Forteller pluginen hva nummeret på den første posten skal være.

Standardverdi: `0`

`updateEvery`
Hvor ofte tickeren skal sjekke for oppdaterte innhold i strømmen. Oppgis i millisekunder. `0` vil si at komponenten ikke sjekker for nye oppdateringer.

Standardverdi: `5000`

B.4.3 Eksempel

```
$('.messages').coolMessageCreate({
    atomFeed: 'feed.xml',
    startAt : 0,
    numberOfPosts : 5,
    classPrefix : 'cool',
    updateEvery: 5000
});
```

Figur B.1: `CoolCalendar` komponenten.

Figur B.2: `CoolDialog` komponenten.

Figur B.3: `CoolTicker` komponenten.

Figur B.4: `CoolMessage` komponenten.

Bibliografi

[1] Json. http://www.json.org.

[2] Json in java. http://www.json.org/java.

[3] 11 Internet AG. qooxdoo. http://qooxdoo.org.

[4] AtomEnabled Alliance. Atomenabled.org. http://www.atomenabled.org.

[5] Karianne Berg. Persistensproblematikk i dynamic presentation generator, Juli 2008.

[6] Jostein Bjørge and Peder Skeidsvoll. Utvikling av presentasjonsmønstre for dpg 2.0. Technical report, Universitetet i Bergen, Institutt for Informatikk, 2009.

[7] RSS Advisory Board. Really simple syndication spesification, tutorials and discussion. http://www.rssboard.org.

[8] Bob Boiko. *Content Management Bible*. Wiley Publishing, 2005.

[9] N. Borenstein. Mime (multipurpose internet mail extensions) part one: Mechanisms for specifying and describing the format of internet message bodies, 1992.

[10] Microsoft Corporation. Microsoft silverlight. http://www.microsoft.com/silverlight.

[11] Microsoft Corporation. Microsoft visio. http://office.microsoft.com/en-us/visio.

[12] Microsoft Corporation. Powerpoint gome page. http://office.microsoft.com/en-us/powerpoint.

[13] Microsoft Corporation. Vbscript. http://msdn.microsoft.com/en-us/library/t0aew7h6.aspx.

[14] Microsoft Corporation. Xmlhttprequest object. http://msdn.microsoft.com/en-us/library/ms535874%28VS.85%29.aspx.

[15] Oracle Corporation. Applets. http://java.sun.com/applets.

[16] Oracle Corporation. Javadoc tool home page. http://java.sun.com/j2se/javadoc/.

[17] Oracle Corporation. Jsr 173: Streaming api for xml. http://www.jcp.org/en/jsr/detail?id=173.

[18] Dave Crane, Eric Pascarello, and Darren James. *Ajax in Action*. Manning Publications Co., Greenwich, CT, USA, 2005.

[19] Kevin Cruickshanks. Verkty for generering av xml-baserte presentasjonar: Jpgen - java

presentasjons generator, 2004.

[20] EMWeb. Wt. http://www.webtoolkit.eu/wt.

[21] Yngve Espelid. Dynamic presentation generator, 2004.

[22] Borealis Festival. Programme. http://borealisfestival.no/2010/programme.

[23] Internet Engineering Task Force. Date and time on the internet: Timestamps. http://www.ietf.org/rfc/rfc3339.txt.

[24] Internet Engineering Task Force. Rfc 4287. http://www.ietf.org/rfc/rfc4287.txt.

[25] Internet Engineering Task Force. Standard for the format of arpa internet text messages. http://www.ietf.org/rfc/rfc822.txt.

[26] Apache Foundation. Apache abderas hjemmeside. http://abdera.apache.org.

[27] Apache Software Foundation. Apache logging services - log4j. http://logging.apache.org/log4j/index.html.

[28] GNOME Foundation. Gtk+. http://www.gtk.org.

[29] Mozilla Foundation. Firebug. http://getfirebug.com.

[30] Mozilla Foundation. Javascript. https://developer.mozilla.org/en/JavaScript.

[31] Mozilla Foundation. Mozilla firefox. http://www.mozilla.com.

[32] The Apache Software Foundation. Apache jackrabbit. http://jackrabbit.apache.org.

[33] The Apache Software Foundation. The apache velocity project. http://velocity.apache.org.

[34] The Dojo Foundation. Dijit widget library. http://dojotoolkit.org/projects/dijit.

[35] The Dojo Foundation. Dojo toolkit. http://www.dojotoolkit.org.

[36] The Dojo Foundation. Dojo toolkit api documentation. http://api.dojotoolkit.org.

[37] The Eclipse Foundation. Eclipse. http://eclipse.org.

[38] The Eclipse Foundation. Javascript development tools. http://www.eclipse.org/webtools/jsdt.

[39] Wikimedia Foundation. Mediawiki. http://www.mediawiki.org.

[40] Wikimedia Foundation. Wikipedia. http://www.wikipedia.org.

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

[42] Thomas Fuchs. script.aculo.us - web 2.0 javascript. http://script.aculo.us.

[43] Fxdteam. Last.fm tag cloud. http://fxdteam.com/lastcloud/index.php?act=new.

[44] Erik Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley, 1995.

[45] Jesse James Garrett. Ajax: A new approach to web applications. http://www.adaptivepath.com/ideas/essays/archives/000385.php.

[46] Bryan Gullan. Bbc style news ticker. http://www.makemineatriple.com/2007/10/bbcnewsticker.

[47] David Heinemeier Hansson. Ruby on rails. http://www.rubyonrails.org.

[48] Jean-François Hovinne. jfeed homepage. http://www.hovinne.com.

[49] Adobe Systems Inc. Adobe flash. http://www.adobe.com/flashplatform.

[50] Adobe Systems Inc. Adobe photoshop. http://www.adobe.com/products/photoshop.

[51] AOL Inc. Aol-mail. http://mail.aol.com.

[52] Apple Inc. Quicktime. http://www.apple.com/quicktime.

[53] Google Inc. Gmail. http://gmail.com.

[54] Google Inc. Google analytics. http://analytics.google.com.

[55] Google Inc. Google documents. http://docs.google.com.

[56] Google Inc. Google web toolkit. http://code.google.com/webtoolkit.

[57] MapQuest Inc. Mapquest. http://www.mapquest.com.

[58] Yahoo! Inc. Yui library. http://developer.yahoo.com/yui.

[59] Bjørn Ove Ingvaldsen. Multimedia i dynamisk presentasjons generator 2.0, September 2008.

[60] Michael Leitner Johann Schrammel and Manfred Tscheligi. Semantically structured tag clouds: An empirical evaluation of clustered presentation approaches, 2009.

[61] The jQuery Project. jquery. http://www.jquery.com.

[62] The jQuery Project. jquery api documentation. http://api.jquery.com.

[63] Håkon Wium Lie and Bert Bos. *Cascading Style Sheets, designing for the Web*. Addison Wesley, 2 edition, 1999.

[64] Last.fm Ltd. Last.fm. http://last.fm.

[65] Wireless Application Protocol Forum Ltd. Wap forum specifications. http://www.wapforum.org/what/technical.htm, 2002.

[66] Kristian Skønberg Løvik. Webucator 3.0 - brukerhåndtering og aksesskontroll for dpg 2.0, 2008.

[67] Eric Martin. Simplemodal. http://www.ericmmartin.com/projects/simplemodal.

[68] Robert C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall, 2002.

[69] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, Upper Saddle River, NJ, USA, 2008.

[70] Ryan Boren Matt Mullenweg and Donncha Caoimh. Wordpress.

http://www.wordpress.org.

[71] Sun Microsystems. Swing. http://java.sun.com/docs/books/tutorial/uiswing.

[72] Sun Microsystems. Code conventions for the java programming language. http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html, 1999.

[73] Gian Carlo Mingati. jquery liscroll - a jquery news ticker. http://www.gcmingati.net/wordpress/wp-content/lab/jquery/newsticker/jq-liscroll/scrollanimate.html.

[74] Khalid A. Mughal. Presentation patterns: Composing web-based presentations. Technical report, Universitetet i Bergen, Institutt for Informatikk, 2003.

[75] Khalid A. Mughal. Annotations. http://www.ii.uib.no/~khalid, 2007.

[76] Ira R. Forman og Nate Forman. *Java Reflection in Action*. Manning Publications Co., 2004.

[77] Tobias Rusås Olsen. Interaksjon og søk i dynamic presentation generator, 2010.

[78] Valerio Proietti. Mootools - a compact javascript framework. http://mootools.net.

[79] Valerio Proietti. Slickspeed selectors test. http://mootools.net/slickspeed.

[80] Bjørn Christian Sebak. Dynamic presentation generator 2.0 - utvikling av dynamisk presentasjonsmønstergenerator og presentasjonsmønsterspesifikasjon, August 2008.

[81] Steve Souders. High-performance web sites. *Commun. ACM*, 51(12):36–41, 2008.

[82] SpringSource. Springsource. http://www.springsource.org.

[83] Prototype Core Team. Prototype api documentation. http://api.prototypejs.org.

[84] Prototype Core Team. Prototype javascript framework. http://www.prototypejs.org.

[85] ROME Developer Team. Romes prosjektside. https://rome.dev.java.net.

[86] Bergens Tidende. Bergenpuls. http://www.bt.no/kultur/bergenpuls.

[87] Twitter. Javazone (javazone) on twitter. http://twitter.com/javazone.

[88] Twitter. Twitter. http://twitter.com.

[89] Universitetet i Bergen. *Rossini, Alessandro and Liberati, Graziano*. Institutt for informatikk, 2006.

[90] The World Wide Web Consortium (W3C). Html5. http://dev.w3.org/html5/spec/Overview.html.

[91] The World Wide Web Consortium (W3C). Platform for internet content selection. http://www.w3.org/PICS.

[92] The World Wide Web Consortium (W3C). Xmlhttprequest. http://www.w3.org/TR/XMLHttpRequest.

[93] The World Wide Web Consortium (W3C). Xsl transformations (xslt). http://www.w3.org/TR/xslt.

[94] Chris Wanstrath. Facebox 2.1. http://famspam.com/facebox.

[95] Curbera Francisco Leymann Frank Storey Tony Ferguson Weerawarana, Sanjiva and Donald F. *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall, 2005.